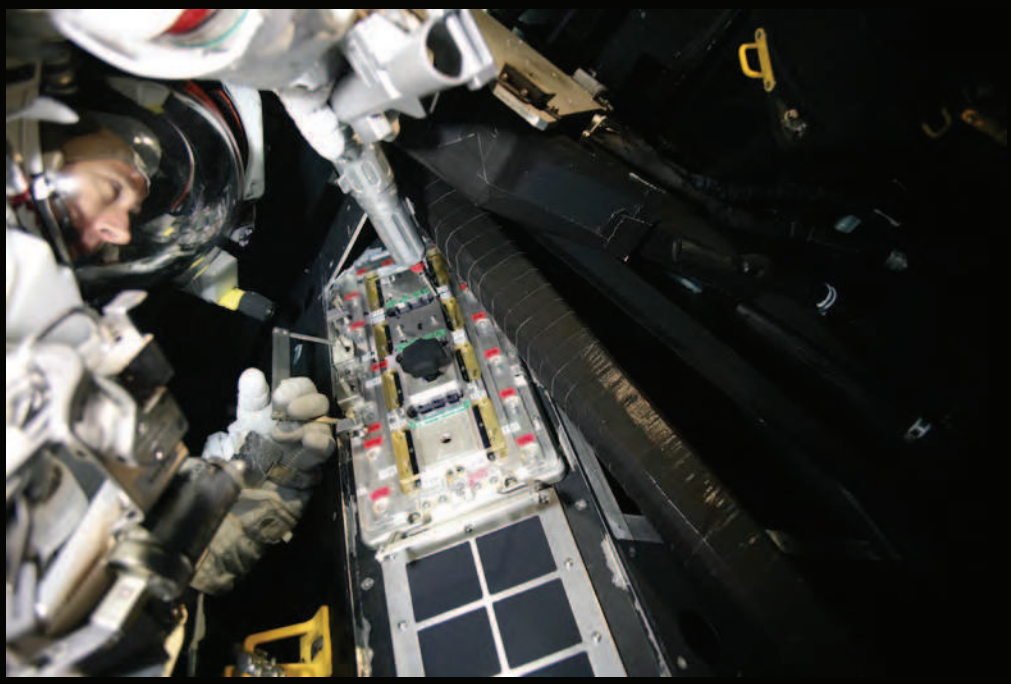
Approximation Algorithms
for Model-Based Diagnosis

Aleksandar Feldman



Approximation Algorithms for Model-Based Diagnosis

Approximation Algorithms for Model-Based Diagnosis

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op maandag 17 mei 2010 om 15.00 uur
door

Aleksandar Beniaminov FELDMAN

informatica ingenieur
geboren te Varna, Bulgarije.

Dit proefschrift is goedgekeurd door de promotor:

Prof.dr.ir. A.J.C. van Gemund

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. A.J.C. van Gemund	Technische Universiteit Delft, promotor
Prof.dr. G. Provan	University College Cork
Prof.dr. C. Witteveen	Technische Universiteit Delft
Prof.dr.ir. H.J. Sips	Technische Universiteit Delft
Prof.dr. K.G. Langendoen	Technische Universiteit Delft
Dr. J. de Kleer	Palo Alto Research Center
Dr. P.J.F. Lucas	Radboud Universiteit Nijmegen

This research was supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW under award DES.07015.

ISBN 978-90-9025023-6

Cover: Astronaut Andrew Feustel practices installing the Fastener Capture Plate on an underwater mockup of the Advanced Camera for Surveys at the Neutral Buoyancy Laboratory in Houston. Photo used with permission of NASA.

Printed by Wöhrmann Print Service, Loskade 4, 7202 CZ Zutphen

Author email: a.b.feldman@tudelft.nl

Dedication

To my father Beniamin Feldman (1939 – 2008).

Contents

List of Abbreviations

AC	Alternating Current
AI	Artificial Intelligence
ARP	Aerospace Recommended Practice
ATMS	Assumption-Based Truth Maintenance System
ATPG	Automated Test Pattern Generation
BCP	Boolean Constraint Propagation
CAD	Computer-Aided Design
CAN	Controller-Area Network
CBA	Cost-Based Abduction
CDRW	Conflict-Directed Random Walk
CNF	Conjunctive Normal Form
COTS	Commercial Off-The-Shelf
DA	Diagnostic Algorithm
DC	Direct Current
DES	Discrete Event Systems
DNF	Disjunctive Normal Form
EPS	Electrical Power System
FDI	Fault Detection and Isolation

FMEA	Failure Modes and Effects Analysis
FOL	First-Order Logic
GDE	General Diagnostic Engine
IBDM	International Berthing and Docking Mechanism
ILS	Iterated Local Search
LCP	Least Cost Proof
LTMS	Logic-Based Truth Maintenance System
MAP	Maximum A Posteriori
MBD	Model-Based Diagnosis
MBR	Model-Based Reasoning
MBT	Model-Based Testing
MC	Minimal Cardinality
MDH	Minimal Diagnosis Hypothesis
MFMC	Max-Fault Min-Cardinality
MPE	Most Probable Explanation
MSMC	Max-Size Min-Cardinality
MSS	Maximum Satisfiable Subset
PIM	Paper Input Module
QBF	Quantified Boolean Formula
RSA	Repetitive Simulated Annealing
SA	Simulated Annealing
SAT	satisfiability
SLS	Stochastic Local Search
TIPP	Tightly Integrated Parallel Printer
VLSI	Very Large-Scale Integration

Chapter 1

Introduction

Even the most carefully designed and tested devices fail to function as intended at some point of their lifetime. Before any corrective action can be taken, one needs to diagnose the malfunctioning device, i.e., to provide an explanation of the observed faulty behavior in terms of its components. Traditionally, myopic investment strategies have favored manual diagnosis, as it offsets initial costs into the maintenance phase. Increasing device complexity [Moore, 2000], however, quickly renders the cost of manual troubleshooting prohibitive, and necessitates the development of methods for *automated* diagnosis [Heckerman et al., 1995].

Automated diagnosis is not only needed for decreasing the cost-of-ownership but it also has no alternative in achieving autonomy. For example, manual troubleshooting of a deep-space probe is impossible [Muscettola et al., 1998], sending a human to a faulty nuclear reactor may be dangerous, and man may fail to correctly and swiftly identify the root cause of failure in a complex device, comprised of thousands of components.

Automated diagnosis is challenging because modern systems are complex, there is lack of knowledge on their functioning, observability is often limited, there are different types of faults, sensors provide noisy readings, etc. Approaches to automated diagnosis differ significantly depending on the type of the diagnosed system. Diagnosis of VLSI circuits [Waicukauski and Lindbloom, 1989], for example, is different from software diagnosis [Abreu, 2009] or diagnosis of hybrid software and hardware systems [Mikaelian et al., 2005].

Expert systems [Duda and Shortliffe, 1983] have been traditionally used to solve diagnostic problems (for a classification of influential diagnostic methods cf. Fig. 1.1). Expert systems incorporate a large body of approaches that apply some kind of automated reasoning to knowledge bases. Rule-based systems [Hayes-Roth, 1985] form a well-known class in this category. Rule-based systems encode "shallow" system knowledge in the form of rules (or material implications) and use basic forward propagation methods for inference. A notable example of a

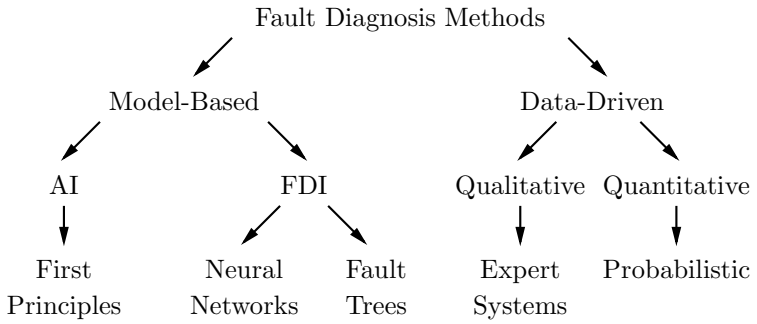


Figure 1.1: *Classification of diagnostic methods*

rule-based system for human diagnostics (although of moderate practical success) is MYCIN [Buchanan and Shortliffe, 1984]. MYCIN has been designed to identify the cause of infections and prescribe antibiotics.

Probabilistic methods treat diagnosis as a belief state estimation problem. Computing diagnosis is readily generalized as a Maximum A Posteriori (MAP) problem and can be solved with methods from Bayesian reasoning [Pearl, 1988]. Computing MAPs is known to be computationally very hard [Park, 2002]. Another method that treats diagnosis as a belief state estimation problem is Particle Filters [de Freitas, 2002].

The above diagnostic methods use expert and symptomatic knowledge about the system behaviour. *Model-based* methods, on the other hand, use deeper system knowledge as a basis for the diagnostic inference process. In literature, the term model is ambiguous and undefined, varying from a set of equations predicting one or more sensor signals to a formal logic description of the topology and behavior of a system. The emphasis of this thesis is on algorithms, hence we use a very restricted logic formulation of a model, but our approach is loosely applicable to qualitative models from first principles [Forbus and de Kleer, 1993], i.e., a class of qualitative physical system models.

Model-based methods for fault diagnosis originate either from control engineering, referred to as Fault Detection and Isolation (FDI) methods [Isermann, 1997], or from Artificial Intelligence (AI). FDI methods perform diagnosis by using classification methods such as neural networks [Ayoubi, 1996] or reasoning methods such as fault-tree analysis [Lee et al., 1985]. One major difference between FDI and AI model-based methods is that AI methods use logical inference mechanisms such as resolution, while FDI methods mostly rely on change detection, residual analysis and other numerical methods. This thesis concerns exclusively the AI model-based methods, hence, from now on we will omit the AI qualifier.

Model-based systems [Reiter, 1987] use an explicit model of the system structure (topology) and behavior to guide the diagnostic inference. One of the first implementations of a model-based reasoning system is the General Diagnostic En-

gine (GDE) [de Kleer and Williams, 1987]. Model-based systems rely on models from first principles, i.e., models that represent the basic, real-world, physical properties of the diagnosed system.

Model-based diagnostic problems are mostly optimization problems as opposed to decision procedures [Krentel, 1986]. This optimization view naturally maps many diagnostic problems into a cost-minimization framework. For example, a diagnostician may be interested in computing the "smallest" explanation of a fault (i.e., the minimal or most cost-optimal set of components that have to be repaired), the minimal reconfiguration of a system exposing a given fault, etc.

Many non-linear optimization problems (for example Max-SAT [Hoos and Stützle, 2004]) are hard to solve. Algorithm designers use *approximation algorithms* [Hochbaum, 1997] to find suboptimal solutions of high quality (i.e., solutions close to the optimal solution), trading optimality for drastic cost reduction. Approximation algorithms have been successful in providing better cost tradeoffs in solving many optimization problems, hence the idea of applying them to automated diagnosis, which is the focus of this thesis.

1.1 Model-Based Diagnosis

Model-Based Diagnosis (MBD) [de Kleer and Williams, 1987; Reiter, 1987] is an abductive reasoning approach that takes a model of an artifact and an actual observation as inputs, and computes one or more diagnoses as an output. Models (system descriptions) are usually specified in some formal language such as First-Order Logic (FOL), propositional logic, etc.

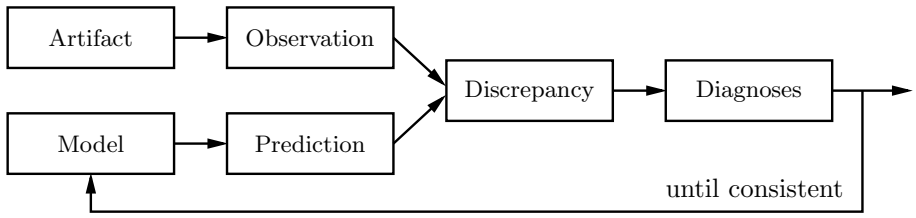


Figure 1.2: *Model-based diagnosis*

Typical to MBD is that it discerns models from observations. Models are known in advance, they are subject to compilation [Darwiche and Marquis, 2002] while observations are acquired at run-time. Observations are expressed as assignments to some observable (or manifestation) variables.

As it is visible from Fig. 1.2, an MBD reasoner uses a model to make predictions about the real-world. Some model variables can be distinguished as component variables (also known as assumable or health variables). A diagnosis is a health assignment (i.e., an assignment to all component variables) that is

consistent with a model and an observation. Alternatively, a diagnosis is a set of components that must be simultaneously faulty to explain an observation. Often, a model cannot predict the faulty behavior of a system because the faulty behavior is unknown or too complex to describe. Such models are called *weak*-fault models or models with ignorance of abnormal behavior [de Kleer et al., 1992a]. Models that capture faulty behavior are called *strong*-fault models.[1] As we will see in this thesis, model weakness has profound consequences for the diagnostic inference process.

There are many situations in which an MBD reasoner returns *multiple* diagnostic candidates: incomplete modeling, sensor-lean systems, weak-fault models, etc. Often the number of diagnoses is exponential in the number of components. This poses a problem for many applications as the residual diagnostic work that has to be performed by the diagnostician to uniquely isolate the faulty components becomes prohibitive. The application of *lex parsimoniae*, however, makes some diagnoses preferred over the others. To first compute the preferred diagnoses, a diagnostic algorithm may impose a (partial) ordering by employing some minimality criterion such as the number of faulty components involved in a diagnosis. The number of faulty components determines the *cardinality* of a diagnosis and cardinality-minimality is a common minimality criterion. Another minimality criterion is subset-minimality–preferred are diagnoses that do not "contain" other diagnoses. In some cases components have *a priori probability of failure* that can be obtained, for example, from reliability analysis [Ebeling, 1997]. The application of diagnosis converts this a priori probability to *a posteriori probability* (the probability of failure given the model and the observation).

Applying the cardinality minimality criterion results in *cardinality-minimal* diagnoses, i.e., diagnoses that minimize the number of simultaneously faulty components. The subset-minimality criterion leads to *subset-minimal* diagnoses. If we consider a diagnosis as a set of faulty components, subset-minimal diagnoses are minimal under set inclusion. Finally, *probability-minimal* diagnoses minimize the a posteriori probability of a fault. The a posteriori probability is usually computed by updating prior failure probabilities with the help of Bayes' rule [de Kleer and Williams, 1987].

A minimality criterion alone is often inadequate to optimally reduce the set of diagnoses (the ideal result is one, possibly multiple-fault, diagnosis that is the true diagnosis). To further reduce diagnostic uncertainty, a diagnostician can apply MBD multiple times, acquiring additional information at each step. This process is called *sequential diagnosis*. One approach to sequential diagnosis is to measure unobserved variables in order to decrease the diagnostic uncertainty. This process is known as *probing* [de Kleer and Williams, 1987]. In addition to probing, one may actively manipulate a system (or apply tests) to reduce the number of

[1] Contrary to intuition, strong-fault models are not always preferable over weak-fault ones. MBD knows relatively little research on model "correctness" and from experience, strong-fault models are more likely to contain incorrect constraints leading to contradictions or wrong predictions.

diagnoses. The latter approach for decreasing the diagnostic uncertainty is known as *active testing*.

1.2 Problem Statement

Consider a system of n components (n can be thousands) where each component can be either healthy or faulty. An algorithm computing diagnosis in such a system has to consider a search space of size $O(2^n)$. There exist sets of restrictions under which computing the first subset-minimal diagnosis is in P (for example, weak-fault models allowing polynomial-time consistency checking), but even in those, computing the first cardinality-minimal diagnosis is *NP*-hard [Selman and Levesque, 1990]. Computing the second minimal diagnosis is always *NP*-hard, regardless of the minimality criterion [Bylander et al., 1991]. Sequential MBD is as complex or more complex than the above, combinational MBD, hence all complexity problems of combinational MBD are carried over in the sequential MBD domain.

The overall cost C of automated diagnosis can be expressed as

$$C = T + W \tag{1.1}$$

where T is the cost of computing the set of diagnostic candidates (in terms of CPU time and memory consumption) and W is the residual work (often in terms of manual labor) of finding the actual diagnosis within the list of candidates.

T and W are related. Typically, an exponential increase in T leads to a small decrease in W. In computing diagnoses, the decrease in W is according to the law of diminishing returns. The reason for that is that computing each separate diagnosis may lead to a combinatorial blowup, while the contribution of a single diagnosis in deriving the a posteriori fault-state probability distribution function (pdf) of each component is small.

Diagnoses of small cardinality contribute most to decreasing W, given their relatively high probability of being the actual fault state. As a result, deterministic algorithms like CDA* [Williams and Ragno, 2007], NGDE [de Kleer, 2009], and RODON [Bunus et al., 2009] generate candidates in order of increasing cardinality. This approach is slow with high-cardinality faults. For example, in a system with 1 000 components, if the leading diagnosis is of cardinality 5, there are approximately 4×10^{12} different health assignments containing 4 faults or less that may have to be considered first.

Some deterministic algorithms increase their performance by using *conflicts* (a conflict is a set of components that cannot be simultaneously healthy given a system description and an observation). GDE [de Kleer and Williams, 1987], for example, computes all minimal conflicts prior to computing diagnoses, while CDA* improves the performance of the diagnostic search by learning conflicts of small size. However, the performance problem remains, as minimal conflicts are

dual to minimal diagnoses and hence, computing them is of the same, prohibitive computational complexity.

Due to the nature of most MBD problems the return in W diminishes when increasing the number and the cardinality of diagnoses. As a result, deterministic algorithms trade increasingly smaller decreases in W for a large investment in T. A key disadvantage of the majority of deterministic MBD algorithms is that they do not efficiently exploit the large, continuous solution subspaces[2] inherent to many MBD problems. Extreme examples are the diagnosis spaces in weak-fault models where continuity is fundamental, but near-continuous subspaces exist in almost all diagnostic models. Continuous subspaces also appear in sequential MBD problems, for example when searching for optimal test vectors.

Our thesis is that in exploiting continuous subspaces, it is more natural to have an "inverted" view on cardinality. A greedy stochastic algorithm starting from high-cardinality diagnoses is the most intuitive implementation of this backwards search. The concept and its advantage over forward, deterministic algorithms are illustrated in Fig. 1.3 (where C is the minimal-cardinality, C' is the stochastic search *approximation* and M is the maximal-cardinality).

As it is visible from Fig. 1.3, a deterministic algorithm such as CDA* or HA* [Feldman and van Gemund, 2006] searches a significant fraction of the search space before finding a minimal-cardinality diagnosis, while a stochastic, backwards search progresses swiftly to a diagnosis of cardinality *close* to the desired one (hence, an approximation algorithm). As will be shown in this thesis, the probability of a greedy search to find a diagnosis of specific cardinality is heavily skewed towards those of small cardinality, yielding a significant reduction of W at very low cost.

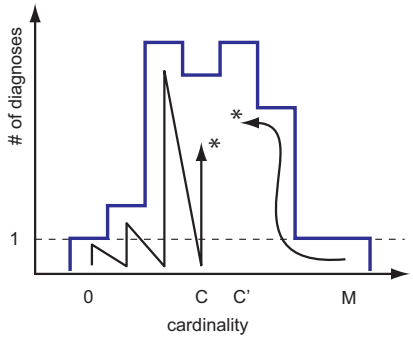


Figure 1.3: *Exhaustive vs random search*

Apart from a large initial increase in W for a budgeted T, another advantage of approximation algorithms based on "inverted" search is that their performance

[2] A continuous subspace is loosely defined as a subset of the true-points of a Boolean function, such that for each true-point it includes another true-point, different in one bit exactly.

degrades more smoothly than the performance of deterministic algorithms, i.e., approximation algorithms compute results (although sub-optimal) even in the cases in which a deterministic MBD algorithm would run out of computational resources. That is, approximation algorithms show *strong anytime behavior*, and will return results after any non-trivial inference time.

Given the potential of our approximation approach to MBD, the main research question addressed in this thesis is formulated as follows:

> *What are the cost/performance trade-offs of using approximation algorithms for MBD?*

In addition to the main research issue above, this thesis also addresses a number of peripheral questions:

- What are the cost/performance trade-offs of using a *greedy* approximation algorithm? Can we analytically or programmatically model the performance of the approximation algorithms and is it possible to establish bounds on the optimality of the approximation algorithms by using these models?
- Often MBD algorithms consider multiple observations (tests) to reduce the diagnostic ambiguity. Can a greedy approximation algorithm be used to compute a *sequence* of control settings that reduces near-optimally the number of diagnoses?
- An important performance characteristic of diagnostic algorithms is the time to compute a high-cardinality solution. However, observations leading to such high-cardinality solutions are extremely hard to determine. Can a greedy approximation algorithm be used to compute observations that *maximize* the cardinality of the minimal-cardinality diagnoses?
- As mentioned earlier, a key premise to our approximation approach is search space continuity. Until now, it was suspected that continuity exists in weak-fault models only. We have noticed that parts of strong-fault models expose topologically-dependent continuous properties. Which specific strong-fault model/observation properties affect the performance of greedy approximation algorithms?

1.3 Contribution

To our knowledge, we are the first to apply approximation algorithms to MBD. Our contributions are described in the following two sections.

1.3.1 Theory

1. We propose a greedy approximation algorithm, called SAFARI (StochAstic Fault diagnosis AlgoRIthm), to solve the fundamental MBD problem of computing minimal diagnoses. We model the progress of SAFARI and establish

bounds on its performance. We consider the distribution of the cardinalities of the diagnoses computed by SAFARI and establish favorable properties of SAFARI in terms of this distribution. We compare the performance and optimality of SAFARI to, amongst others, that of a family of MBD algorithms based on stochastic Max-SAT. We empirically demonstrate, using the 74XXX and ISCAS85 suites of benchmark combinatorial circuits, that SAFARI achieves several orders-of-magnitude speedup over both deterministic algorithms and algorithms based on stochastic Max-SAT.

2. We combine approaches from sequential diagnosis [Shakeri, 1996], passive monitoring [Pietersma and van Gemund, 2006], probing [de Kleer and Williams, 1987], and Automated Test Pattern Generation (ATPG), into an MBD framework called FRACTAL (FRamework for ACtive Testing ALgorithms). We propose an algorithm, called FRACTAL^{G}, that approximates the computationally hard problem of computing optimal control assignments (as defined in FRACTAL). FRACTAL^{G} offers a low-cost, greedy, stochastic approach that maintains exponential decay of the number of cardinality-minimal diagnoses computed by SAFARI. We compare the performance of FRACTAL^{G} to FRACTAL^{A} (ATPG-based sequential diagnosis) and FRACTAL^{P} (probing). FRACTAL^{G} overcomes limitations of FRACTAL^{A} and offers a trade-off in computational complexity versus optimality in reducing the diagnostic uncertainty. We use FRACTAL^{P} to show lower bound on the decay of SAFARI diagnoses, achievable for a model extended with unlimited control circuitry. We present extensive empirical data on FRACTAL experiments with models of the 74XXX/ISCAS85 combinational circuits.

3. We define the so called Max-Fault Min-Cardinality (MFMC) problem of finding an observation that leads to a minimum-cardinality diagnosis that has the highest possible number of faults given a system description, and use MFMC observation vectors for benchmarking the performance of combinational MBD algorithms. We present a near-optimal, stochastic algorithm, called MIRANDA (Max-fault mIn-caRdinAlity observatioN Deduction Algorithm), that computes observations leading to diagnoses of large cardinality. Experiments show that MIRANDA delivers optimal results on the 74XXX circuits, as well as good MFMC cardinality estimates on the larger ISCAS85 circuits.

1.3.2 Application

The MBD approximation algorithms discussed in this thesis are validated in terms of their practical significance. We have applied them, amongst others, to a model of the Electrical Power System (EPS) testbed in the ADAPT lab at NASA Ames Research Center [Poll et al., 2007]. ADAPT EPS has been a part of the First International Diagnostic Competition (DXC'09) where SAFARI has shown several

orders-of-magnitude speedup over deterministic algorithms while preserving the optimality of the diagnostic candidates [Feldman et al., 2010].

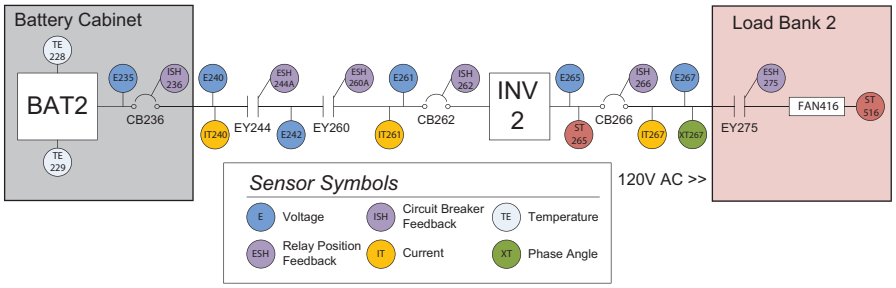


Figure 1.4: *ADAPT-Lite (diagram courtesy of NASA Ames Research Center)*

Systems similar to ADAPT (part of ADAPT is shown in Fig. 1.4) are typical for the aerospace industry, and the good diagnostic results with approximation algorithms indicate that implementations of the latter can be used for a variety of both off-line and on-board, real-time reasoning tasks.

A key advantage of the MBD approach is that it is domain independent. To demonstrate the wide applicability of our algorithms, we introduce a qualitative model of a Paper Input Module (PIM), part of a heavy-duty printer. While the ADAPT EPS consists of electrical components only, the PIM model is a combination of electrical, mechanical and pneumatic ones.

The PIM case study demonstrates the use of FRACTAL^{G} for solving real-world sequential MBD problems. We have shown that control settings computed by FRACTAL^{G} significantly reduce the initial set of SAFARI diagnoses even with a coarse-grained device model.

Typically, the implementations of our approximation algorithms are significantly shorter compared to the implementations of deterministic algorithms. This results in lower certification costs (the cost of approving mission-critical software is usually measured per line of code) and lower overall costs for using MBD algorithms in mission critical applications.

1.4 Thesis Outline

A separate chapter of this thesis is dedicated to each theoretical contribution listed in Sec. 1.3. Chapter 2 discusses technical background, classification of strong-fault models, and metrics. Chapters 3, 4, and 5 present SAFARI, MIRANDA, and FRACTAL, respectively. Chapter 6 summarizes this thesis and provides recommendations for future work. Appendix A introduces the modeling language LYDIA. Appendix B discusses the ADAPT and ADAPT-Lite models, while Appendix C presents the PIM model.

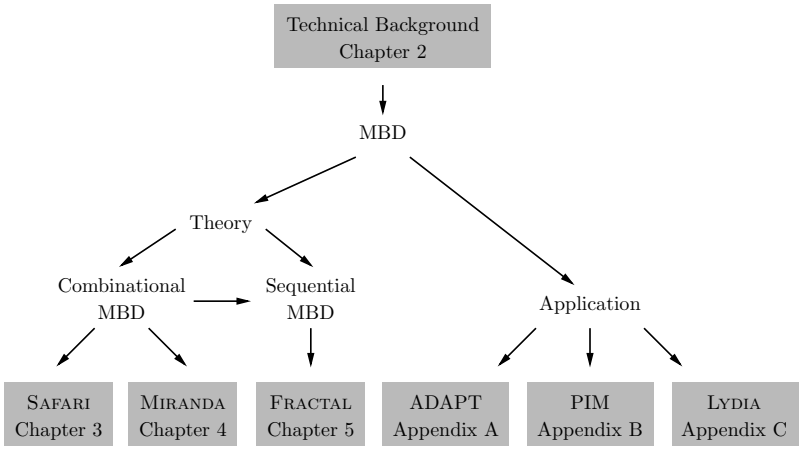


Figure 1.5: *Overview*

Figure 1.5 illustrates the relations between the various MBD topics discussed in this thesis, as well as the content of each chapter. Each chapter in this thesis is directly based on at least two international, peer-reviewed publications.

1.5 Origin of Chapters

All of the publications cited in this section have been co-authored with Van Gemund. The publications of Sec. 2.1 - Sec. 2.2, Chapters 3 - 5, have been co-authored with Provan.

Chapter 2 has been partly published (Sec. 2.1 - Sec. 2.2) in the *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)* [Feldman et al., 2009c].

Chapter 3 has been submitted to the Journal of Artificial Intelligence Research. A short version appeared in the *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08)* [Feldman et al., 2008c] and a proof-of-concept has been published in *Proceedings of the Seventh Symposium on Abstraction, Reformulation, and Approximation (SARA'07)* [Feldman et al., 2007b].

Chapter 4 has been published in *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08)* [Feldman et al., 2008b]. A preliminary version appeared in the *Proceedings of the Eighteenth International Workshop on Principles of Diagnosis (DX'07)* [Feldman et al., 2007a].

Chapter 5 has been submitted to the Journal of Artificial Intelligence Research. A preliminary version has been published in the *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09)* [Feldman et al., 2009b] and a proof-of-concept appeared in the *Proceedings of the First International Conference on Prognostics and Health Management (PHM'08)* [Feldman et al., 2008a].

Appendix A originally appeared as a Technical Report [Feldman and van Gemund, 2007b].

Appendix B is submitted to the International Journal of Prognostics and Health Management.

Appendix C originally appeared as a Technical Report [Feldman and van Gemund, 2009].

Chapter 2

Model-Based Diagnosis

Model-based diagnosis, as formulated in terms of logic [Reiter, 1987], focuses on determining whether an assignment of health/failure states to a set of behavioral mode variables is consistent with a system description and an observation (e.g., of sensor values). Hence, the diagnostic process consists of taking an observation (or an observation vector), and then inferring the failure-mode assignment (diagnosis) consistent with that observation(s).

2.1 Concepts and Definitions

Our discussion continues by formalizing some MBD notions. This thesis uses the traditional diagnostic definitions [de Kleer and Williams, 1987], except that we use propositional logic terms (conjunctions of literals) instead of sets of failing components.

Central to MBD, a *model* of an artifact is represented as a Well-Formed Propositional Formula (**Wff**) over some set of variables. We discern subsets of these variables as *assumable* and *observable*.[1]

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

Throughout this thesis we will assume that $\mathrm{OBS} \cap \mathrm{COMPS} = \emptyset$ and $\mathrm{SD} \not\models \bot$.

Let $\mathrm{COMPS} = \{h_1, h_2, \cdots, h_n\}$. This mnemonics corresponds to the convention that a positive sign of a "health" variable $(h_1, h_2, \cdots, h_n)$ denotes a nominal

[1] In the MBD literature the assumable variables are also referred to as "component", "failure-mode", or "health" variables. Observable variables are also called "measurable", or "control" variables.

(healthy) component state. Faulty components are denoted as $\neg h_1, \neg h_2, \cdots, \neg h_n$. Other authors use "ab" for abnormal or "ok" for healthy.

Not all propositional theories used as system descriptions are of interest to MBD. Diagnostic systems can be characterized by a restricted set of models, the restriction making the problem of computing diagnosis amenable to algorithms like the ones presented in this thesis. We consider two main classes of models.

Definition 2 (Weak-Fault Model)**.** A diagnostic system DS = ⟨SD, COMPS, OBS⟩ belongs to the class **WFM** iff SD is equivalent to $(h_1 \Rightarrow F_1) \wedge (h_2 \Rightarrow F_2) \wedge \cdots \wedge (h_n \Rightarrow F_n)$ such that for $1 \leq i \leq n$, $\{h_i\} \subseteq$ COMPS, $F_i \in$ **Wff**, and COMPS$\cap V' = \emptyset$, where V' is the set of all variables appearing in $F_1, F_2, \cdots, F_n$.

Weak-fault models are sometimes referred to as models with *ignorance of abnormal behavior* [de Kleer et al., 1992a], or *implicit fault systems*. Alternatively, a model may specify faulty behavior for its components. In the following definition, with the aim of simplifying the formalism throughout this thesis, we adopt a slightly restrictive representation of faults, allowing only a single fault mode per assumable variable. This can be easily generalized by introducing multi-valued logic or suitable encodings [Feldman et al., 2006; Hoos, 1999].

Definition 3 (Strong-Fault Model)**.** A diagnostic system DS = ⟨SD, COMPS, OBS⟩ belongs to the class **SFM** iff SD is equivalent to $(h_1 \Rightarrow F_{1,1}) \wedge (\neg h_1 \Rightarrow F_{1,2}) \wedge \cdots \wedge (h_n \Rightarrow F_{n,1}) \wedge (\neg h_n \Rightarrow F_{n,2})$ such that $1 \leq i, j \leq n, k \in \{1, 2\}$, $\{h_i\} \subseteq$ COMPS, $F_{\{j,k\}} \in$ **Wff**, and none of h_i appears in $F_{j,k}$.

Membership testing for the **WFM** and **SFM** classes can be performed efficiently in many cases, for example, when a model is represented explicitly as in Def. 2 or Def. 3.

2.1.1 A Running Example

We will use the Boolean circuit shown in Fig. 2.1 as a running example for illustrating many notions and algorithms in this thesis. The subtractor, shown there, consists of seven components: an inverter, two or-gates, two xor-gates, and two and-gates. The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models the normative (healthy) behavior of an inverter, where the variables i, o, and h represent input, output and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2)$ and an or-gate by $h \Rightarrow (o \Leftrightarrow i_1 \vee i_2)$. Finally, an xor-gate is specified as $h \Rightarrow [o \Leftrightarrow \neg (i_1 \Leftrightarrow i_2)]$.

The above propositional formulae are copied for each gate in Fig. 2.1 and their variables renamed in such a way as to properly connect the circuit and disambiguate the assumables, thus obtaining a propositional formula for the Boolean subtractor, given by:

$$\begin{aligned} \mathrm{SD}_w = &\{h_1 \Rightarrow [i \Leftrightarrow \neg (y \Leftrightarrow p)]\} \wedge \{h_2 \Rightarrow [d \Leftrightarrow \neg (x \Leftrightarrow i)]\} \wedge \\ &\wedge [h_3 \Rightarrow (j \Leftrightarrow y \vee p)] \wedge [h_4 \Rightarrow (m \Leftrightarrow l \wedge j)] \wedge \\ &\wedge [h_5 \Rightarrow (b \Leftrightarrow m \vee k)] \wedge [h_6 \Rightarrow (x \Leftrightarrow \neg l)] \wedge [h_7 \Rightarrow (k \Leftrightarrow y \wedge p)] \end{aligned} \tag{2.1}$$

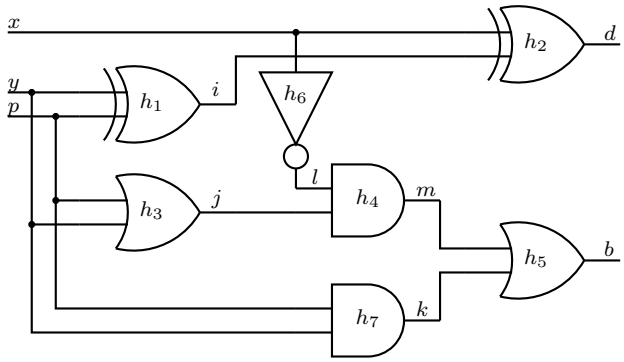


Figure 2.1: *A subtractor circuit*

A strong-fault model for the Boolean circuit shown in Fig. 2.1 is constructed by assigning fault-modes to the different gate types. We will assume that, when malfunctioning, the output of an xor-gate has the value of one of its inputs, an or-gate can be stuck-at-one, an and-gate can be stuck-at-zero, and an inverter behaves like a buffer. This gives us the following strong-fault model formula for the Boolean subtractor circuit:

$$\begin{aligned} \mathrm{SD}_s = \mathrm{SD}_w \wedge [\neg h_1 \Rightarrow (i \Leftrightarrow y)] \wedge [\neg h_2 \Rightarrow (d \Leftrightarrow x)] \wedge (\neg h_3 \Rightarrow j) \wedge \\ \wedge (\neg h_4 \Rightarrow \neg m) \wedge (\neg h_5 \Rightarrow b) \wedge [\neg h_6 \Rightarrow (x \Leftrightarrow l)] \wedge (\neg h_7 \Rightarrow \neg k) \end{aligned} \tag{2.2}$$

For both models (SD_s and SD_w), the set of assumable variables is COMPS = $\{h_1, h_2, \cdots, h_7\}$ and the set of observable variables is OBS = $\{x, y, p, d, b\}$.

2.1.2 Diagnosis and Minimal Diagnosis

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

Definition 4 (Health Assignment)**.** Given a system DS = ⟨SD, COMPS, OBS⟩, an assignment ω to all variables in COMPS is defined as a health assignment.

A health assignment ω is represented as a conjunction of propositional literals. In some cases it is convenient to use the set of negative or positive literals in ω. These two sets are denoted as $Lit^{-}(\omega)$ and $Lit^{+}(\omega)$, respectively.

In our example, the "all nominal" assignment is $\omega_1 = h_1 \wedge h_2 \wedge \cdots \wedge h_7$. The health assignment $\omega_2 = h_1 \wedge h_2 \wedge h_3 \wedge \neg h_4 \wedge h_5 \wedge h_6 \wedge \neg h_7$ means that the two and-gates from Fig. 2.1 are malfunctioning.

What follows is a formal definition of consistency-based diagnosis.

Definition 5 (Diagnosis)**.** Given a diagnostic system DS = ⟨SD, COMPS, OBS⟩, an observation α over some variables in OBS, and a health assignment ω, ω is a diagnosis iff $\mathrm{SD} \wedge \alpha \wedge \omega \not\models \bot$.

Traditionally, other authors [de Kleer and Williams, 1987] arrive at minimal diagnosis by computing a minimal hitting set of the minimal conflicts (broadly, minimal health assignments incompatible with the system description and the observation), while this thesis makes no use of conflicts, hence the equivalent, direct definition above.

There is a total of 96 possible diagnoses given SD_w and an observation $\alpha_1 = x \wedge y \wedge p \wedge b \wedge \neg d$. Example diagnoses are $\omega_3 = \neg h_1 \wedge h_2 \wedge \cdots \wedge h_7$ and $\omega_4 = h_1 \wedge \neg h_2 \wedge h_3 \wedge \cdots \wedge h_7$. Trivially, given a weak-fault model, the "all faulty" health assignment (in our example $\omega_a = \neg h_1 \wedge \cdots \wedge \neg h_7$) is a diagnosis for any instantiation of the observable variables in OBS (cf. Def. 2).

In the MBD literature, a range of types of "preferred" diagnosis has been proposed. This turns the MBD problem into an optimization problem. In the following definition we consider the common subset-ordering.

Definition 6 (Minimal Diagnosis)**.** A diagnosis $\omega^{\subseteq}$ is defined as minimal, if no diagnosis $\tilde{\omega}^{\subseteq}$ exists such that $Lit^{-}(\tilde{\omega}^{\subseteq}) \subset Lit^{-}(\omega^{\subseteq})$.

Consider the weak-fault model SD_w of the circuit shown in Fig. 2.1 and an observation $\alpha_2 = \neg x \wedge y \wedge p \wedge \neg b \wedge d$. In this example, two of the minimal diagnoses are $\omega_5^{\subseteq} = \neg h_1 \wedge h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$ and $\omega_6^{\subseteq} = \neg h_1 \wedge h_2 \wedge \cdots \wedge h_5 \wedge \neg h_6 \wedge \neg h_7$. The diagnosis $\omega_7 = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$ is non-minimal as the negative literals in $\omega_5^{\subseteq}$ form a subset of the negative literals in ω_7.

Note that the set of all minimal diagnoses characterizes all diagnoses for a weak-fault model, but that does not hold in general for strong-fault models [de Kleer et al., 1992a]. In the latter case, faulty components may "exonerate" each other, resulting in a health assignment containing a proper superset of the negative literals of another diagnosis not to be a diagnosis. In our example, given SD_s and $\alpha_3 = \neg x \wedge \neg y \wedge \neg p \wedge b \wedge \neg d$, it follows that $\omega_8^{\subseteq} = h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge \cdots \wedge h_7$ is a diagnosis, but $\omega_9 = h_1 \wedge h_2 \wedge \neg h_3 \wedge \neg h_4 \wedge \cdots \wedge h_7$ is not a diagnosis, despite the fact that the set of negative literals in ω_9 ($\{\neg h_3, \neg h_4\}$) forms a superset of the set of negative literals in $\omega_8^{\subseteq}$ ($\{\neg h_3\}$).

Definition 7 (Number of Minimal Diagnoses)**.** Let the set $\Omega^{\subseteq}(\mathrm{SD} \wedge \alpha)$ contain all minimal diagnoses of a system description SD and an observation α. The number of minimal diagnoses, denoted as $|\Omega^{\subseteq}(\mathrm{SD} \wedge \alpha)|$, is defined as the size of $\Omega^{\subseteq}(\mathrm{SD} \wedge \alpha)$.

Continuing our running example, $|\Omega^{\subseteq}(\mathrm{SD}_w \wedge \alpha_2)| = 8$ and $|\Omega^{\subseteq}(\mathrm{SD}_s \wedge \alpha_3)| = 2$. The number of non-minimal diagnoses of $\mathrm{SD}_w \wedge \alpha_2$ is 61.

Definition 8 (Cardinality of a Diagnosis)**.** The cardinality of a diagnosis, denoted as $|\omega|$, is defined as the number of negative literals in ω.

Diagnosis cardinality gives us another partial ordering: a diagnosis is defined as *minimal cardinality* iff it minimizes the number of negative literals.

Definition 9 (Minimal-Cardinality Diagnosis)**.** A diagnosis $\omega^{\leq}$ is defined as minimal-cardinality if no diagnosis $\tilde{\omega}^{\leq}$ exists such that $|\tilde{\omega}^{\leq}| < |\omega^{\leq}|$.

The cardinality of a minimal-cardinality diagnosis computed from a system description SD and an observation α is denoted as $\mathit{MinCard}(\mathrm{SD} \wedge \alpha)$. For our example model SD_w and an observation $\alpha_4 = x \wedge y \wedge p \wedge \neg b \wedge \neg d$, it follows that $\mathit{MinCard}(\mathrm{SD}_w \wedge \alpha_4) = 2$. Note that in this case all minimal diagnoses are also minimal-cardinality diagnoses.

A minimal cardinality diagnosis is a minimal diagnosis, but the opposite does not hold. There are minimal diagnoses that are not minimal-cardinality diagnoses. Consider the example SD_w and α_2 given earlier in this section, and the two resulting minimal diagnoses $\omega_5^{\subseteq}$ and $\omega_6^{\subseteq}$. From these two, only $\omega_5^{\subseteq}$ is a minimal-cardinality diagnosis.

Definition 10 (Number of Minimal-Cardinality Diagnoses)**.** Let the set $\Omega^{\leq}(\mathrm{SD} \wedge \alpha)$ contain all minimal-cardinality diagnoses of a system description SD and an observation α. The number of minimal-cardinality diagnoses, denoted as $|\Omega^{\leq}(\mathrm{SD} \wedge \alpha)|$, is defined as the cardinality of $\Omega^{\leq}(\mathrm{SD} \wedge \alpha)$.

Computing the number of minimal-cardinality diagnoses for the running example results in $|\Omega^{\leq}(\mathrm{SD}_w \wedge \alpha_2)| = 2$, $|\Omega^{\leq}(\mathrm{SD}_s \wedge \alpha_3)| = 2$, and $|\Omega^{\leq}(\mathrm{SD}_w \wedge \alpha_4)| = 4$.

2.2 Diagnostic Model Taxonomy

Within MBD, two broad classes of model types have been specified: weak-fault models **WFM** [de Kleer et al., 1992a] and strong-fault models **SFM** [Struss and Dressler, 1989]. Traditionally, **WFM** has been considered to be computationally simple, and **SFM** computationally hard. Weak-fault models describe a system only in terms of its normal (non-faulty) behaviour, whereas strong-fault models include a definition of some aspects of abnormal behaviour. Strong-fault models can avoid violating physical rules (cf. [Struss and Dressler, 1989]), but at the cost of increased complexity: moving from a binary-valued model with n components (which is adequate for weak-fault models) to one with $m+1$ possible faulty values increases the maximum number of failure candidates from 2^n to $(m+1)^n$.

In terms of worst-case complexity, finding the first minimal diagnosis for a Horn model in **WFM** can be done in polynomial time, but finding the next minimal diagnosis is NP-complete [Friedrich et al., 1990]. In contrast, inference in strong-fault models entails computing kernel diagnoses [de Kleer et al., 1992a], which is a Σ_2^P-hard task and is known to be computationally intensive in practice; for example, kernel diagnoses are given by the prime implicants of the minimal conflicts [de Kleer et al., 1992a]. Further, the average case complexity of reasoning in **WFM** versus **SFM** increases from poly-time in n (**WFM**) to exponential in n (**SFM**) [de Kleer et al., 1992a].

We show that, by closer examination of **SFM**, there is a spectrum of model types, and, possibly, corresponding inference complexities. We identify two main

categories of **SFM**, which we call literal-based SFM, **lSFM**, and function-based SFM, **fSFM**. In contrast to previous work discussing strong fault models, e.g., [Console and Torasso, 1991; Mozetič and Holzbaur, 1994; Struss and Dressler, 1989], we propose sub-classes of strong-fault models that occur in practice, such as stuck-at circuit models, and that are amenable to the algorithms introduced in this thesis.

We now introduce our classification of fault models. As mentioned earlier, given a health variable h_i and an arbitrary **Wff** F_i, normal behaviour for component i is denoted using the clause $h_i \Rightarrow F_i$, and abnormal behaviour by $\neg h_i \Rightarrow F_i$. Hence a weak-fault model, as depicted in row 2 of Table 2.1, is given by $\bigwedge_{i=1}^{n} (h_i \Rightarrow F_i)$; a strong-fault model consists of clauses denoting both normal and abnormal behaviour, i.e., $\bigwedge_{i=1}^{n} (h_i \Rightarrow F_{i,1}) \wedge (\neg h_i \Rightarrow F_{i,2})$.

Table 2.1: *Model classification [Feldman et al., 2009c]*

Notation	Model Class	System Description	Restrictions
M	abduction models	propositional **Wff**	
WFM	weak-fault models	$\bigwedge_{i=1}^{n} (h_i \Rightarrow F_i)$	none of h_i appears in F_j, $h_i \in$ COMPS $(1 \leq i, j \leq n)$
SFM	strong-fault models	**SFM** = **lSFM** $\cup$ **hSFM** $\cup$ **fSFM**	
lSFM	literal-based strong-fault models	$\bigwedge_{i=1}^{n} (h_i \Rightarrow F_i) \wedge (\neg h_i \Rightarrow l_i)$	none of h_i appears in F_j, $h_i \in$ COMPS, $l_i \in V$, $(\neg l_i \in V)$, $l_i \notin$ COMPS, $(1 \leq i, j \leq n)$
hSFM	dual Horn strong-fault models	$\bigwedge_{i=1}^{n} (h_i \Rightarrow F_i) \wedge (\neg h_i \Rightarrow H_i)$	none of h_i appears in F_j, $H_i = x_{i,1} \vee x_{i,2} \vee \cdots \vee x_{i,n_i}$ is a disjunction with at most one negative literal $(1 \leq i, j \leq n)$
fSFM	functional strong-fault models	$\bigwedge_{i=1}^{n} (h_i \Rightarrow F_{i,1}) \wedge (\neg h_i \Rightarrow F_{i,2})$	none of h_i or $\neg h_i$ appears in $F_{j,1}$ and $F_{j,2}$ for $1 \leq i, j \leq n$
nlSFM	negative literal strong-fault models	$\bigwedge_{i=1}^{n} (h_i \Rightarrow F_i) \wedge (\neg h_i \Rightarrow \neg F_i)$	none of h_i appears in F_j, $h_i \in$ COMPS $(1 \leq i, j \leq n)$

Continuing the running example from Sec. 2.1.1, it is clear that $\mathrm{SD}_w \in$ **WFM**, $\mathrm{SD}_s \in$ **SFM**, $\mathrm{SD}_s \notin$ **lSFM**, $\mathrm{SD}_s \notin$ **hSFM**, $\mathrm{SD}_s \notin$ **nlSFM**.

We generalize the well-known class of *stuck-at* models using the strong-fault class **lSFM**, in which each clause denoting abnormal behaviour is given by $(\neg h_i \Rightarrow l_i)$ for some literal l_i; for example, this captures component i (with output l_i being

stuck-at-0) as given by $[(h_i = \text{stuck-at-0}) \Rightarrow (l_i = 0)]$.

The class **hSFM** of dual Horn strong-fault models has the consequent **Wff** F_i restricted to dual Horn clauses (clauses with at most one negative literal). The class **fSFM** of functional strong-fault models allows the consequent **Wff** F_i to take on any form. Finally, the class **nlSFM** of negative-literal strong-fault models has the consequent **Wff** F_i restricted to defining the *negation* of the normal behaviour of component i given h_i.

In the above classification we do not address the problem of recognizing if a model SD belongs to a specific class. This problem is expected to be hard in the general case (arbitrary **Wff** and class). In some specific cases (e.g., an explicit model representation from Table 2.1), recognition can be done efficiently.

Determining the model class has consequences for the choice of MBD reasoning algorithm and for preprocessing the model. Models belonging to **lSFM**, **hSFM**, and **nlSFM** can be relaxed by "stripping" the "stuck-at" constraints and solved with a diagnostic solver optimized for **WFM**. The strong-part of the model can be solved subsequently, and the set of diagnoses circumscribed [Feldman et al., 2009c].

2.3 Converting Propositional Formulae to Clausal Form

Diagnosis is related to SAT, and often uses SAT or Max-SAT solvers and related approaches. Although there are exceptions [Thiffault et al., 2004], SAT solvers predominantly work on input in Conjunctive Normal Form (CNF). Converting arbitrary propositional **Wff** to CNF can be computationally hard and is considered harmful for the performance of many reasoning algorithms [Ramesh et al., 1997]. The MBD algorithms represented in this thesis also need CNF input for a number of complexity and implementation issues. Converting a propositional **Wff** to CNF can be done with [Tseitin, 1983] or without [Forbus and de Kleer, 1993] the introduction of intermediate variables. In both cases important structural information is lost, which may lead to performance degradation when checking if a **Wff** is consistent or when computing a satisfiable solution.

Lemma 1. *A fault-model* $\text{SD} = F_1 \wedge F_2 \wedge \ldots \wedge F_n$ *(*$\text{SD} \in$ **WFM** *or* $\text{SD} \in$ **SFM***) with* $|\text{COMPS}|$ *component variables can be converted to CNF in time* $O\left(\sum_{i=1}^{n} \zeta_i\right)$ *where* ζ_i *is the time for converting subformula* F_i $(1 \leq i \leq n)$ *to CNF.*

Proof (Sketch). If all F_i are converted to CNF, the resulting SD would be in CNF. □

Lemma 1 restates the worst-case complexity of converting a propositional **Wff** to CNF as a parameter of $|\text{COMPS}|$, and effectively defines the class of diagnostic models that have concise CNF representation. These are models whose component models have concise CNF representations. Particular diagnostic algorithms could use a non-CNF SAT solver, but for practical reasons we have constrained our reasoning to diagnostic models with concise CNF encodings.

Consider, for example, the formula $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \cdots \vee (x_n \wedge y_n)$, which is in Disjunctive Normal Form[2] (DNF) and, converted to CNF, has 2^n clauses. Although similar examples of propositional **Wff** having exponentially many clauses in their CNF representations are easy to find, they are artificial and are rarely encountered in MBD (cf. Manolios and Vroon [2007] for benchmark results on Boolean circuit **Wff** to CNF conversion). Furthermore, the Boolean circuits with which we have tested the performance of our algorithms, do not show exponential blow-up when converted to CNF.

2.4 Benchmarks

We have experimented on the well-known benchmark models of ISCAS85 circuits [Brglez and Fujiwara, 1985]. As models derived from the ISCAS85 circuits are computationally intensive (from a diagnostic perspective), we have also considered four medium-sized circuits from the 74XXX family [Hansen et al., 1999]. Without loss of generality, in our models only gates are allowed to fail (as opposed to ATPG where gates typically do not fail but wires can be "stuck-at-zero" or "stuck-at-one").

Table 2.2: *An overview of the* 74XXX/ISCAS85 *circuits (V is the total number of variables and C is the number of clauses)*

Name	Description	|IN|	|OUT|	|COMPS|	V	C
74182	4-bit CLA	9	5	19	47	150
74L85	4-bit comparator	11	3	33	77	236
74283	4-bit adder	9	5	36	81	244
74181	4-bit ALU	14	8	65	144	456
c432	27-channel interrupt controller	36	7	160	356	514
c499	32-bit SEC circuit	41	32	202	445	714
c880	8-bit ALU	60	26	383	826	1 112
c1355	32-bit SEC circuit	41	32	546	1 133	1 610
c1908	16-bit SEC/DEC	33	25	880	1 793	2 378
c2670	12-bit ALU	233	140	1 193	2 695	3 269
c3540	8-bit ALU	50	22	1 669	3 388	4 608
c5315	9-bit ALU	178	123	2 307	4 792	6 693
c6288	32-bit multiplier	32	32	2 416	4 864	7 216
c7552	32-bit adder	207	108	3 512	7 232	9 656

Table 2.2 provides a summary of the 74XXX/ISCAS85 circuits. Many (5 out of 14) circuits are Arithmetic Logic Units (ALUs). The number of inputs, outputs

[2]Note that all DNF formulae are also propositional **Wff**s.

and components are given in the third, fourth, and fifth column of Table 2.2, respectively. The sixth and seventh columns show the number of variables and clauses, respectively, in the CNF representation of the circuits.

2.5 Diagnostic Metrics

In this section we assume that each diagnostic experiment (a diagnostic scenario) defines a fault injection $\omega^\star$ of minimal-cardinality and an associated observation α. Note that computing $\langle \alpha, \omega^\star \rangle$ pairs such that $|\omega^\star| > 1$ is not trivial (this problem is addressed in Chapter 4).

We have defined several diagnostic metrics which are computed from the set of diagnostic candidates Ω and the injected fault ω^* (classification errors, and utility metrics). Consider a single diagnostic candidate $\omega \in \Omega$. Both the candidate ω and the injected fault $\omega^\star$ 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 2.2 shows how ω and $\omega^\star$ partition the set of all components in four.

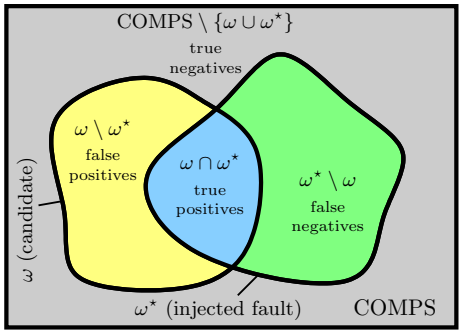


Figure 2.2: *The diagnostic candidate ω and the injected fault $\omega^\star$ partition* COMPS *in four sets*

False positives and false negatives in this context relate to individual candidates, i.e., misclassified components in a single diagnostic candidate. For brevity we use the notation in Table 2.3 for the Fig. 2.2 sets.

2.5.1 Classification Errors and Isolation Accuracy

The *classification errors* metric is defined as:

$$M_{\text{err}} = \sum_{\omega \in \Omega} W(\omega)(|\omega \ominus \omega^\star|) \tag{2.3}$$

Table 2.3: *Notation for sizes of some frequently used sets*

Variable	Set	Description
N	$\|\text{COMPS}\|$	all components
n_{fn}	$\|\omega^\star \setminus \omega\|$	false negatives
N_{fn}	$\|\text{COMPS} \setminus \omega\|$	the set of healthy components from the viewpoint of the diagnostic algorithm
n_{fp}	$\|\omega \setminus \omega^\star\|$	false positives
N_{fp}	$\|\omega\|$	the set of faulty components from the viewpoint of the diagnostic algorithm

where $W(\omega)$ is the weight of a diagnosis ω such that:

$$\sum_{\omega \in \Omega} W(\omega) = 1 \tag{2.4}$$

In Eq. (2.3), $\omega \ominus \omega^\star$ denotes the symmetric difference of the ω and $\omega^\star$ sets, i.e., the number of misclassified components. Note that $|\omega \ominus \omega^\star| = n_{\text{fn}} + n_{\text{fp}}$ and $N = N_{\text{fn}} + N_{\text{fp}}$.

The isolation accuracy metric that we show next is very similar to the classification errors metric.

$$M_{\text{ia}} = \sum_{\omega \in \Omega} W(\omega)(N - |\omega \ominus \omega^\star|) \tag{2.5}$$

One can see that M_{ia} and M_{err} are duals, i.e.:

$$\frac{M_{\text{ia}}}{N} + \frac{M_{\text{err}}}{N} = 1 \tag{2.6}$$

Consider the isolation accuracy (m_{ia}) of a single diagnostic candidate $\omega \in \Omega$:

$$m_{\text{ia}} = N - |\omega \ominus \omega^\star| \tag{2.7}$$

Eq. 2.7 defines a plane in the $(n_{\text{fn}}, n_{\text{fp}}, m_{\text{ia}})$-space (cf. Fig 2.3).

The m_{ia} metric "penalizes" a Diagnostic Algorithm (DA) for each misclassified component. As it is visible from Fig. 2.3, the penalty is applied linearly.

The isolation accuracy metric M_{ia} originates in the automotive industry [Committee E-32, 2008]. The Aerospace Recommended Practice (ARP) computes the closely related probability of correct classification in the following way. For each component we compute a confusion matrix. The probability of correct classification is the sum of the main diagonal divided by the total number of classifications (cf. the referenced ARP [Committee E-32, 2008] for details and examples).

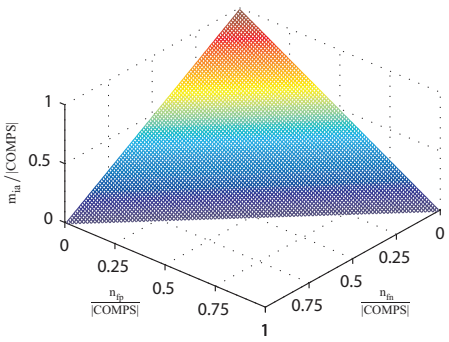


Figure 2.3: *m_{ia} as a function of n_{fn} and n_{fp}*

It can be shown that the probability of correct classification, as defined in the above ARP, is equivalent to M_{ia}, if both fault and nominal component modes are used for the computation of the confusion matrices. The probability of correct classification is conditioned on the fault probability while the probability measured by M_{ia} is not. The latter is purely a metric design consideration. The fact that we use nominal modes for computing M_{ia} leads to higher correlation of M_{ia} with the detection accuracy metrics defined later in this section.

If more than one predicted mode vector is reported by a DA, (meaning that the diagnostic output consists of a set of candidate diagnoses), then the isolation accuracy and the classification errors are calculated for each predicted component mode vector and weighted by the candidate probabilities reported by the DA as it is seen in Eq. (2.3) and Eq. (2.5). M_{ia} and M_{err} are very useful for single diagnoses but with multiple candidates they are less intuitive. The metric that follows is loosely based on the concept of "repair effort" and partly remedies this problem [Feldman et al., 2010].

2.5.2 Utilities

In what follows we show the derivations of three utility metrics: system repair utility M_{sru}, diagnosis repair utility M_{dru}, and utility M_{utl}.

The *utilities* metrics measure the work for correctly identifying all false negatives and false positives in a diagnostic candidate. Alternatively, the utilities metrics measure the expected number of calls to a testing oracle that always determines correctly the health state of a component. Note that these metrics assume an equal cost for fixing a false negative and a false positive.

System Repair Utility

Consider an injected fault $\omega^\star$ ($\omega^\star$ is a set of faulty components) and a diagnostic candidate ω (also a set of what the DA considers faulty components). The num-

ber of truly faulty components that are improperly diagnosed by the diagnostic algorithm as healthy (false negatives) is $n_{\text{fn}} = |\omega^\star \setminus \omega|$ (cf. Fig. 2.2). In general a diagnostician has to perform extra work to verify a diagnostic candidate ω, which must be reflected in the system repair utility. We assume that he or she has access to a test oracle that states if a component c is healthy or faulty.

We first determine what the expected number of tests a diagnostician has to perform to test all components in $\omega^\star \setminus \omega$ (the false negatives) if the diagnostician chooses untested components at random with uniform probability. In the worst case, the diagnostician has to test all the remaining $\text{COMPS} \setminus \omega$ components (the diagnostic algorithm has already determined the state of all components in ω). Consider the average situation. Recall that $N_{\text{fn}} = |\text{COMPS} \setminus \omega|$, where N_{fn} is the size of the "population" of components to be tested.

The probability of observing $s-1$ successes (faulty components) in $k+s-1$ trials (i.e., k oracle tests) is given by the direct application of the hypergeometric distribution:

$$p(k, s-1) = \frac{\binom{n_{\text{fn}}}{s-1}\binom{N_{\text{fn}}-n_{\text{fn}}}{k}}{\binom{N_{\text{fn}}}{k+s-1}} \tag{2.8}$$

The probability $p(k, s)$ of then observing a faulty component in the next oracle test is simply the number of remaining false negatives $n_{\text{fn}} - (s-1)$ divided by the size of the remaining population $(N_{\text{fn}} - (s+k-1))$:

$$p(k, s) = \frac{n_{\text{fn}} - s + 1}{N_{\text{fn}} - k - s + 1} \tag{2.9}$$

and the probability of having exactly k oracle faults up to the $s^{\text{-th}}$ test, is then the product of these two probabilities:

$$p'(k, s, n_{\text{fn}}, N_{\text{fn}}) = \frac{\binom{n_{\text{fn}}}{s-1}\binom{N_{\text{fn}}-n_{\text{fn}}}{k}(n_{\text{fn}} - s + 1)}{\binom{N_{\text{fn}}}{k+s-1}(N_{\text{fn}} - k - s + 1)} \tag{2.10}$$

The formula above is the probability mass of the inverse hypergeometric distribution that, in our case, yields the probabilities for testing k healthy components before we find s faulty components out of the population (no repetitions). The expected value $\text{E}'[k]$ of $p'(k, s, n_{\text{fn}}, N_{\text{fn}})$ (from the definition of a first central moment of a random variable) is:

$$\text{E}'[k] = \sum_{x=0}^{n_{\text{fn}}} x p'(x, s, n_{\text{fn}}, N_{\text{fn}}) \tag{2.11}$$

Replacing $p'(k, s, n_{\text{fn}}, N_{\text{fn}})$ in (2.11) and simplifying gives us the mean of the inverse hypergeometric distribution:[3]

$$\text{E}'[k] = \frac{s(N_{\text{fn}} - n_{\text{fn}})}{n_{\text{fn}} + 1} \tag{2.12}$$

[3] For a detailed derivation of the negative hypergeometric mean, cf. [Schuster and Sype, 1987].

As we are interested in finding $s = n_{\text{fn}}$ faulty components, the expected value $E'(n_{\text{fn}}, N_{\text{fn}})$ becomes:

$$\mathrm{E}'[k] = \frac{n_{\text{fn}}(N_{\text{fn}} - n_{\text{fn}})}{n_{\text{fn}} + 1} \tag{2.13}$$

The expected number of tests $\mathrm{E}[t_{\text{fn}}]$ (as opposed to the expected number of faulty components $\mathrm{E}'[k]$) then becomes:

$$\mathrm{E}[t_{\text{fn}}] = \frac{n_{\text{fn}}(N_{\text{fn}} - n_{\text{fn}})}{n_{\text{fn}} + 1} + n_{\text{fn}} = \frac{n_{\text{fn}}(N_{\text{fn}} + 1)}{n_{\text{fn}} + 1} \tag{2.14}$$

The expected number of tests $\mathrm{E}[t_{\text{fn}}]$ is then normalized by the number of components N and flipped alongside the y axis to give the system repair utility:

$$m_{\text{sru}} = 1 - \frac{n_{\text{fn}}(N_{\text{fn}} + 1)}{N(n_{\text{fn}} + 1)} \tag{2.15}$$

Plotting the system repair utility m_{sru} against a variable number of false negatives is shown in Fig. 2.4. One can see that unlike m_{err} which changes linearly, m_{sru} "penalizes" improperly diagnosed components exponentially.

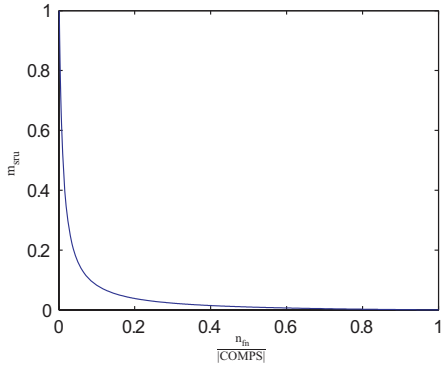


Figure 2.4: m_{sru} *as a function of* n_{fn}

The system repair utility for a set of diagnoses is defined as:

$$M_{\text{sru}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{sru}}(\omega^\star, \omega) \tag{2.16}$$

All weights $W(\omega)$, $\omega \in \Omega$, are computed by the DA.

Diagnosis Repair Utility

Using $\mathrm{E}[t_{\text{fn}}]$ in a metric is not enough as it only captures the effort to "eliminate" (test) all false negatives. The size of the set of false positives is $n_{\text{fp}} = |\omega \setminus \omega^\star|$ (cf.

Fig. 2.2 and Table 2.3). To find all false positives, the diagnostician has to test in the worst case all components in ω. Hence, the general population is $N_{\text{fp}} = |\omega|$. Repeating the argument for $\text{E}[t_{\text{fn}}]$ we determine the expected number of tests for testing all false positives $\text{E}[t_{\text{fp}}]$:

$$\text{E}[t_{\text{fp}}] = \frac{n_{\text{fp}}(N_{\text{fp}} + 1)}{n_{\text{fp}} + 1} \tag{2.17}$$

Similarly, the diagnostic repair utility m_{dru} is the normalized $\text{E}[t_{\text{fp}}]$:

$$m_{\text{dru}} = 1 - \frac{n_{\text{fp}}(N_{\text{fp}} + 1)}{N(n_{\text{fp}} + 1)} \tag{2.18}$$

The system repair utility for a set of diagnoses is defined as:

$$M_{\text{dru}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{dru}}(\omega^\star, \omega) \tag{2.19}$$

Utility

The utility metric (per candidate) is a combination of m_{sru} and m_{dru}:

$$m_{\text{utl}} = 1 - \frac{\text{E}[t_{\text{fn}}] + \text{E}[t_{\text{fp}}]}{N} = 1 - \frac{n_{\text{fn}}(N_{\text{fn}} + 1)}{N(n_{\text{fn}} + 1)} - \frac{n_{\text{fp}}(N_{\text{fp}} + 1)}{N(n_{\text{fp}} + 1)} \tag{2.20}$$

The utility metric (per scenario[4]) is

$$M_{\text{utl}} = \sum_{\omega \in \Omega} W(\omega) m_{\text{utl}}(\omega^\star, \omega) \tag{2.21}$$

Figure 2.5 plots m_{utl} for varying numbers of false negatives and false positives in a (symmetric) case where the cardinality of the injected fault is half the number of components. Normally, the number of injected faulty components $|\omega^\star|$ is small compared to the total number of components f, which leads to an asymmetric m_{utl} plot. In such cases, $N_{\text{fp}} \ll N_{\text{fn}}$, hence the role of the false positives is small. In Fig. 2.5, there is a global optimum $m_{\text{utl}} = 1$ for $n = 0$ and $\bar{n} = 0$, i.e., all components in ω are classified correctly.

2.5.3 Consistency

Consider a model SD and an observation α. The set of consistent diagnoses is defined as:

$$\Omega^\top = \{\omega \in \Omega : \text{SD} \wedge \alpha \wedge \omega \not\models \bot\} \tag{2.22}$$

[4] We assume one observation per diagnostic scenario, unless specified otherwise.

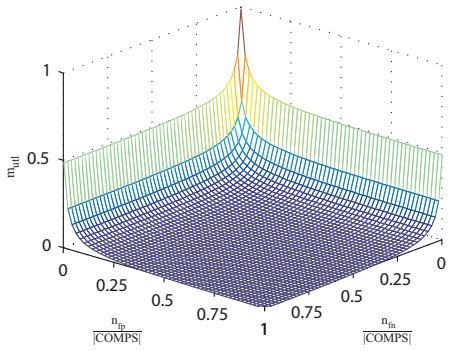


Figure 2.5: *m_{utl} as a function of n_{fn} and n_{fp}*

where Ω is the set of all diagnostic candidates computed by a DA (some candidates in Ω may not be diagnoses).

The set $\Omega^{\top}$ can be computed from SD, α, and Ω by using a DPLL-solver [Davis et al., 1962]. The *consistency* metric can be computed from $\Omega^{\top}$, W and the injected fault $\omega^{\star}$:

$$M_{\text{sat}} = \sum_{\omega \in \Omega^{\top}} W(\omega) \tag{2.23}$$

M_{sat} is a measure of how much probably mass a DA associates with diagnoses consistent with the observations.

2.5.4 Computational Metrics

The two metrics that follow measure the use of computational resources of a DA.

CPU Load

The *CPU load* during an experiment is computed as:

$$M_{\text{cpu}} = C_s + \sum_{c \in C} c \tag{2.24}$$

where C_s is the amount of CPU time spent by a DA during startup and C is a vector with the actual CPU time spent by the DA at each time step (we create the vector C by querying the operating system at equal intervals of time). The CPU load is reported in milliseconds.

Memory Load

The *memory load* is defined as:

$$M_{\text{mem}} = \max_{m \in M} m \tag{2.25}$$

where M is a vector with the maximum memory size allocated at each step of the diagnostic session. During an experiment, we create the vector M by repetitively measuring the maximum memory consumption. The memory load is reported in Kb.

2.5.5 System Metrics

For each system we consider a number of diagnostic scenarios. Throughout this thesis, we assume that a diagnostic scenario consists of a single fault injection $\omega^\star$ (a set of faulty components) that leads to a single observation α.

The metrics M_{err}, M_{ia}, M_{sru}, M_{dru}, M_{utl}, M_{sat}, and M_{mem} are based on a single scenario. To receive "per system" results we combine the metrics of each scenario using unweighted average. For example, if a system SD is tested with scenarios $\mathbf{S} = \{S_1, S_2, \cdots, S_n\}$, the "per system" utility of SD is computed as:

$$\bar{M}_{\text{utl}} = \sum_{S \in \mathbf{S}} \frac{1}{|\mathbf{S}|} M_{\text{utl}}(\text{SD}, S) \tag{2.26}$$

where $M_{\text{utl}}(\text{SD}, S)$ is the "per scenario" utility of system SD and scenario S.

The rest of the "per system" metrics ($\bar{M}_{\text{err}}$, $\bar{M}_{\text{ia}}$, $\bar{M}_{\text{sru}}$, $\bar{M}_{\text{dru}}$, $\bar{M}_{\text{sat}}$, and $\bar{M}_{\text{mem}}$) are defined in a way analogous to $\bar{M}_{\text{utl}}$.

Chapter 3

Approximate Model-Based Diagnosis Using Greedy Stochastic Search

In this chapter, we propose a StochAstic Fault diagnosis AlgoRIthm, called SAFARI, which trades off guarantees of computing minimal diagnoses for computational efficiency. We empirically demonstrate, using the 74XXX and ISCAS85 suites of benchmark combinatorial circuits, that SAFARI achieves several orders-of-magnitude speedup over two well-known deterministic algorithms, CDA* and HA*, for multiple-fault diagnoses; further, SAFARI can compute a range of multiple-fault diagnoses that CDA* and HA* cannot. We also prove that SAFARI is optimal for a range of propositional fault models, such as the widely-used weak-fault models (models with ignorance of abnormal behavior). We discuss the optimality of SAFARI in a class of strong-fault circuit models with stuck-at failure modes. By modeling the algorithm itself as a Markov chain, we provide exact bounds on the minimality of the diagnosis computed. SAFARI also displays strong anytime behavior, and will return a diagnosis after any non-trivial inference time.

3.1 Introduction

Model-based diagnosis is an area of non-monotonic logic that uses a system model, together with observations about system behavior, to isolate sets of faulty components (diagnoses) that explain the observed behavior, according to some minimality criterion. The standard MBD formalization [Reiter, 1987] frames a diagnostic problem in terms of a set of logical clauses that include mode-variables describing the nominal and fault status of system components; from this the diagnostic status

of the system can be computed given an observation of the system's sensors. MBD provides a sound and complete approach to enumerating multiple-fault diagnoses, and exact algorithms can guarantee finding a diagnosis optimal with respect to the number of faulty components, probabilistic likelihood, etc.

The biggest challenge is the computational complexity of the MBD problem. The MBD problem of determining if there exists a diagnosis with at most k faults is NP-hard for the arbitrary propositional models we consider in this thesis [Bylander et al., 1991; Friedrich et al., 1990]. Computing the set of all diagnoses is harder still, since there are possibly exponentially many such diagnoses. Since almost all proposed MBD algorithms have been complete and exact, with some authors proposing possible trade-offs between completeness and faster consistency checking by employing methods such as BCP [Williams and Ragno, 2007], the complexity problem still remains a major challenge to MBD.

To overcome this complexity problem, we propose a novel *approximation approach* for multiple-fault diagnosis, based on a stochastic algorithm. SAFARI (StochAstic Fault diagnosis AlgoRIthm) sacrifices guarantees of optimality, but for diagnostic systems in which faults are described in terms of an arbitrary deviation from nominal behavior SAFARI can compute diagnoses several orders of magnitude faster than competing algorithms.

Our contributions are as follows. (1) This chapter introduces an approximation algorithm for computing diagnoses within an MBD framework, based on a greedy stochastic algorithm. (2) We show that we can compute minimal-cardinality diagnoses for weak fault models in poly-time (calling an incomplete SAT-solver that implements Boolean Constraint Propagation[1] (BCP) only), and that more general frameworks (such as a sub-class of strong fault models) are also amenable to this class of algorithm. (3) We model SAFARI search as a Markov chain to show the performance and optimality trade-offs that the algorithm makes. (4) We apply this algorithm to a suite of benchmark combinatorial circuits, demonstrating order-of-magnitude speedup over two state-of-the-art deterministic algorithms, CDA* and HA*, for multiple-fault diagnoses. (5) We compare the performance of SAFARI against a range of Max-SAT algorithms for our benchmark problems. Our results indicate that, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality. SAFARI is of great practical significance, as it can compute a large fraction of minimal-cardinality diagnoses for discrete systems too large or complex to be diagnosed by existing deterministic algorithms.

3.2 Related Work

This chapter (1) generalizes Feldman et al. [2008c], (2) introduces important theoretical results for strong-fault models, (3) extends the experimental results there,

[1] With formulae in CNF, BCP is implemented through the unit resolution rule.

and (4) provides a comprehensive optimality analysis of SAFARI.

On a gross level, one can classify the types of algorithms that have been applied to solve MBD as being based on search or compilation. The search algorithms take as input the diagnostic model and an observation, and then search for a diagnosis, which may be minimal with respect to some minimality criterion. Examples of search algorithms include A*-based algorithms, such as CDA* [Williams and Ragno, 2007] and hitting set algorithms [Reiter, 1987]. Compilation algorithms pre-process the diagnostic model into a form that is more efficient for on-line diagnostic inference. Examples of such algorithms include the ATMS [de Kleer, 1986a] and other prime-implicant methods [Kean and Tsiknis, 1993], DNNF [Darwiche, 1998], and OBDD [Bryant, 1992]. To the best of our knowledge, all of these approaches adopt exact methods to compute diagnoses; in contrast, SAFARI adopts a stochastic approach to computing diagnoses.

At first glance, it seems like MBD could be efficiently solved using an encoding as a SAT [Jin et al., 2005], constraint satisfaction [Freuder et al., 1995] or Bayesian network [Kask and Dechter, 1999] problem. However, one needs to take into account the increase in formula size (over a direct MBD encoding), in addition to the underconstrained nature of MBD problems.

SAFARI has resemblance to Max-SAT [Hoos and Stützle, 2004] and we have conducted extensive experimentation with both complete Max-SAT (partial and weighted) and Max-SAT based on Stochastic Local Search (SLS). Empirical evidence shows that although Max-SAT can compute diagnoses in many of the cases, the performance of Max-SAT degrades when increasing the circuit size or the cardinality of the injected faults. In particular, SAFARI outperforms Max-SAT at least an order-of-magnitude for the class of diagnostic problems we have considered. In the case of SLS-based Max-SAT, the optimality of Max-SAT diagnoses is significantly worse than that of SAFARI.

We show that SAFARI exploits a particular property of MBD problems, called diagnostic continuity, which improves the optimality of SAFARI compared to, for example, straightforward ALLSAT encodings [Jin et al., 2005]. We experimentally confirm this favorable performance and optimality of SAFARI. Although SAFARI has close resemblance to Max-SAT, SAFARI exploits specific landscape properties of the diagnostic problems which allow (1) simple termination criteria and (2) optimality bounds. Due to the hybrid nature of SAFARI (the use of LTMS and SAT), SAFARI avoids getting stuck in local optima and performs better than Max-SAT based methods. Incorporating approaches from Max-SAT, and in particular SAPS [Hutter et al., 2002] in future versions of SAFARI may help in solving more general abduction problems which may not expose the continuous properties of the models we have considered.

Stochastic algorithms have been discussed in the framework of constraint satisfaction [Freuder et al., 1995] and Bayesian network inference [Kask and Dechter, 1999]. The latter two approaches can be used for solving suitably translated MBD problems. It is often the case, though, that these encodings are more difficult for search than specialized ones.

MBD is an instance of constraint optimization, with particular constraints over failure variables, as we will describe. MBD has developed algorithms to exploit these domain properties, and our proposed approach differs significantly from almost all MBD algorithms that appear in the literature. While most advanced MBD algorithms are deterministic, SAFARI borrows from SLS algorithms that, rather than backtracking, may randomly flip variable assignments to determine a satisfying assignment. Complete MBD algorithms typically make use of preferences, e.g., fault-mode probabilities, to improve search efficiency; SAFARI uses this technique on top of its stochastic search over the space of diagnoses.

A closely-related diagnostic approach is that of Fijany et al. [2003], who map the minimal-hitting set problem into the problem of finding an assignment with bounded weight satisfying a monotone SAT problem, and then propose to use efficient SAT algorithms for computing diagnoses. The approach of Fijany et al. [2003] has shown speedups in comparison with other diagnosis algorithms; the main drawback is the number of extra variables and clauses that must be added in the SAT encoding, which is even more significant for strong fault models and multi-valued variables. In contrast, our approach works directly on the given diagnosis model and requires no conversion to another representation.

Our work bears the closest resemblance to preference-based or Cost-Based Abduction (CBA) [Charniak and Shimony, 1994; Santos Jr., 1994]. Of the algorithmic work in this area, the primary paper that adopts stochastic local search is Abdelbar et al. [2006]. In this paper, Abdelbar et al. [2006] present a hybrid two-stage method that is based on Iterated Local Search (ILS) and Repetitive Simulated Annealing (RSA). The ILS stage of the algorithm uses a simple hill-climbing method (randomly flipping assumables) for the local search phase, and tabu search for the perturbation phase. RSA repeatedly applies Simulated Annealing (SA), starting each time from a random initial state. The hybrid method initially starts from an arbitrary state, or a greedily-chosen state. It then applies the ILS algorithm; if this algorithm fails to find the optimal solution after a fixed number τ of hill-climbing steps[2] or after a fixed number $\mathcal{R}$ of repetitions of the perturbation-local search cycle,[3] ILS-based search is terminated and the RSA algorithm is run until the optimal solution is found.

Our work differs from that of Abdelbar et al. [2006] in several ways. First, our initial state is generated using a random SAT solution. The hill-climbing phase that we use next is similar to that of Abdelbar et al. [2006]; however, we randomly restart should hill-climbing not identify a "better" diagnosis, rather than applying tabu-search or simulated annealing. Our approach is simpler than that

[2]Hill-climbing proceeds as follows: given a current state s with a cost of $f(s)$, a neighbouring state s' is generated by flipping a randomly chosen assumable hypothesis. If $f(s')$ is better than $f(s)$, then s' becomes the current state; otherwise, it is discarded. If τ iterations elapse without a change in the current state, the local search exits.

[3]Perturbation-local search, starting from a current state s with a cost of $f(s)$, randomly chooses an assumable variable h, and applies tabu-search to identify a better state by flipping h based on its tabu status.

of Abdelbar et al. [2006], and for the case of weak fault models is guaranteed to be optimal; in future work we plan to compare our approach to that of Abdelbar et al. [2006] for strong fault models.

In 2009 SAFARI competed against the diagnostic algorithms NGDE [de Kleer, 2009] and RODON [Bunus et al., 2009] in the synthetic track of the first diagnostic competition DXC'09 [Feldman et al., 2010]. The CPU and memory performance of SAFARI were an order of magnitude better than the competing algorithms despite the fact that NGDE and RODON performed better than the complete algorithms discussed in this section.

3.3 Complexity of Diagnostic Inference

This section discusses the complexity of the problems in which we are interested, namely the problem of computing a single or the set of all minimal diagnoses, using two minimality criteria, subset-minimality ($\subseteq$) and cardinality-minimality ($\leq$). We assume as input a CNF formula defined over a variable set V, of which $\gamma = |\text{COMPS}|$ are assumable (or fault) variables. Table 3.1 introduces the notation that we use to define these 4 types of diagnosis.

Table 3.1: *Summary of definitions of types of diagnosis of interest*

Symbol	Diagnoses	Preference Criterion
$\omega^{\subseteq}$	1	$\subseteq$ (subset-minimality)
$\omega^{\leq}$	1	$\leq$ (cardinality-minimality)
$\Omega^{\subseteq}$	all	$\subseteq$ (subset-minimality)
$\Omega^{\leq}$	all	$\leq$ (cardinality-minimality)

The complexity of computing the set of all diagnoses is harder than computing a single diagnosis, since the number of diagnoses is, in the worst case, exponential in the input size (number of components). This problem is bounded from below by the problem of counting the number of diagnoses. This problem has been shown to be #co-*NP*-Complete [Hermann and Pichler, 2007]. These results indicate that the MBD problems we are interested in, i.e., computing the set of minimal-cardinality diagnoses over propositional models, are intractable.

If we restrict our clauses to be Horn or definite Horn, then we can reduce the complexity of the problems that we are solving, at the expense of decreased model expressiveness. Under a Horn-clause restriction, for SD $\in$ **WFM**, determining if a first minimal diagnosis exists is in P. Under the same restriction, for SD $\in$ **SFM**, deciding if a first minimal diagnosis exists is *NP*-hard [Friedrich et al., 1990]. In both cases (SD $\in$ **WFM**, **SFM**) deciding if a next diagnosis exists is *NP*-hard.

The diagnosis problems of interest in this article are intractable in the worst-case. The complexity of a closely-related problem, Propositional Abduction Prob-

lems (PAPs), has been studied by Eiter and Gottlob [1995]. Eiter and Gottlob show that for a propositional PAP, the problem of determining if a solution exists is Σ_2^P-complete. Computing a minimal diagnosis is a search problem, and hence it is more difficult to pose a decision question for proving complexity results. Consequently, one can just note that computing a diagnosis minimal with respect to $\subseteq$ / $\leq$ requires $O(\log|\text{COMPS}|)$ calls to an *NP* oracle [Eiter and Gottlob, 1995], asking the oracle at each step if a diagnosis containing at most k faulty components exists.

Results on abduction problems indicate that the task of approximate diagnosis is intractable. Roth [1996] has addressed the problems of abductive inference, and of approximating such inference. Roth focuses on counting the number of satisfying assignments for a range of AI problems, including some instances of PAPs. In addition, Roth shows that approximating the number of satisfying assignments for these problems is intractable.

Abdelbar [2004] has studied the complexity of approximating Horn abduction problems, showing that even for a particular Horn restriction of the propositional problem of interest, the approximation problem is intractable. In particular, for an abduction problem with costs assigned to the assumables (which can be used to model both the preference-orderings $\subseteq, \leq$), he has examined the complexity of finding the Least Cost Proof (LCP) for the evidence (OBS), where the cost of a proof is taken to be the sum of the costs of all hypotheses that must be assumed in order to complete the proof. For this problem he has shown that it is *NP*-hard to approximate an LCP within a fixed ratio r of the cost of an optimal solution, for any $r > 0$.

SAFARI approximates the intractable problems denoted in Table 3.1. We show that for **WFM**, SAFARI can efficiently compute a single diagnosis that is minimal under $\subseteq$ by using a satisfiability oracle. For SD $\in$ **SFM**, SAFARI generates a sound but possibly sub-optimal diagnosis (or set of diagnoses). We have referred to papers indicating that it is intractable to approximate, within a fixed ratio, a minimal diagnosis. In the following, we adopt a stochastic approach that cannot provide fixed-ratio guarantees. However, SAFARI trades off optimality for efficiency and can compute most diagnoses with high likelihood.

3.4 Stochastic MBD Algorithm

In this section we discuss an algorithm for computing multiple-fault diagnoses using stochastic search.

3.4.1 A Simple Example (Continued)

Consider the Boolean subtractor shown in Fig. 2.1, its weak-fault model SD_w given by (2.1), and the observation α_4 from the preceding section. The four minimal diagnoses associated to $\text{SD}_w \wedge \alpha_4$ are: $\omega_1 = \neg h_1 \wedge h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$,

$\omega_2 = h_1 \wedge \neg h_2 \wedge h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge h_7$, $\omega_3 = \neg h_1 \wedge h_2 \wedge \ldots \wedge h_6 \wedge \neg h_7$, and $\omega_4 = h_1 \wedge \neg h_2 \wedge h_3 \wedge \ldots \wedge h_6 \wedge \neg h_7$.

A naïve deterministic algorithm would check the consistency of all the $2^{|\mathrm{COMPS}|}$ possible health assignments for a diagnostic problem, 128 in the case of our running example. Furthermore, most deterministic algorithms first enumerate health assignments of small cardinality but with high a priori probability, which renders these algorithms impractical in situations when the minimal diagnosis is of a higher cardinality. Such performance is not surprising even when using state-of-the art MBD algorithms which utilize, for example conflict learning, or partial compilation, considering the bad worst-case complexity of finding all minimal diagnoses (cf. Sec. 3.3).

In what follows, we will show a two-step diagnostic process that requires fewer consistency checks. The first step involves finding a random non-minimal diagnosis as a starting point (cf. Sec. 3.4.2 for details on computing random SAT solutions with equal likelyhood). The second step attempts to minimize the fault cardinality of this diagnosis by repeated modification of the diagnosis.

The first step is to find one random, possibly non-minimal diagnosis of $\mathrm{SD}_w \wedge \alpha_4$. Such a diagnosis we can obtain from a classical DPLL solver after modifying it in two ways: (1) not only determine if the instance is satisfiable but also extract the satisfying solution and (2) find a *random* satisfiable solution every time the solver is invoked. Both modifications are trivial, as DPLL solvers typically store their current variable assignments and it is easy to choose a random variable and value when branching instead of deterministic ones. The latter modification may possibly harm a DPLL variable or value selection heuristics, but later in this chapter we will see that this is of no concern for the type of problems we are considering as diagnostic systems are typically underconstrained.

In the subtractor example we call the DPLL solver with $\mathrm{SD}_w \wedge \alpha_4$ as an input and we consider the random solution (and obviously a diagnosis) $\omega_5 = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge h_5 \wedge \neg h_6 \wedge \neg h_7$ ($|\omega_5| = 4$). In the second step of our stochastic algorithm, we will try to minimize ω_5 by repetitively choosing a random negative literal, "flipping" its value to positive (thus obtaining a candidate with a smaller number of faults), and calling the DPLL solver. If the new candidate is a diagnosis, we will try to improve further this newly discovered diagnosis, otherwise we will mark the attempt a "failure" and choose another negative literal. After some constant number of "failures" (two in this example), we will terminate the search and will store the best diagnosis discovered so far in the process.

After changing the sign of $\neg h_7$ in ω_5 we discover that the new health assignment is not consistent with $\mathrm{SD}_w \wedge \alpha_4$, hence it is not a diagnosis and we discard it. Instead, the algorithm attempts changing $\neg h_6$ to h_6 in ω_5, this time successfully obtaining a new diagnosis $\omega_6 = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge h_5 \wedge h_6 \wedge \neg h_7$ of cardinality 3. Next the algorithm tries to find a diagnosis of even smaller cardinality by randomly choosing $\neg h_1$ and $\neg h_7$ in ω_6, respectively, and trying to change their sign, but both attempts return an inconsistency. Hence the "climb" is aborted and ω_6 is stored as the current best diagnosis.

Repeating the process from another random initial DPLL solution, gives us a new diagnosis $\omega_7 = \neg h_1 \wedge \neg h_2 \wedge h_3 \wedge \neg h_4 \wedge h_5 \wedge h_6 \wedge \neg h_7$. Changing the sign of $\neg h_7$, again, leads to inconsistency, but the next two "flips" (of $\neg h_4$ and $\neg h_2$) lead to a double-fault diagnosis $\omega_8 = \neg h_1 \wedge h_2 \wedge \ldots \wedge h_6 \wedge \neg h_7$. The diagnosis ω_8 can not be improved any further as it is minimal. Hence the next two attempts to improve ω_8 fail and ω_8 is stored in the result.

This process is illustrated in Fig. 3.1, the search for ω_6 is on the left and for ω_8 on the right. Gates which are shown in solid black are "suspected" as faulty when the health assignment they participate in is tested for consistency, and inconsistent candidates are crossed-out.

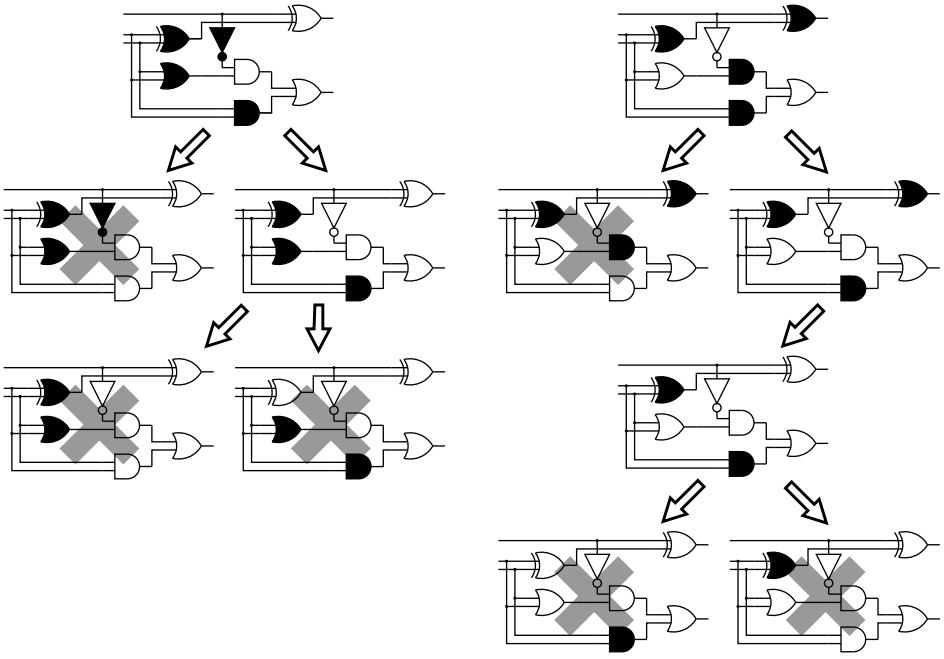


Figure 3.1: *An example of a stochastic diagnostic process*

Let us consider the result. We have found two diagnoses: ω_6 and ω_8, where ω_6 is not a minimal diagnosis. This we have done at the price of 11 calls to a DPLL subroutine. The suboptimal diagnosis ω_6 is of value as its cardinality is near the one of a minimal diagnosis. Hence this is an example of a way to find approximations of minimal diagnoses, while drastically reducing the number of consistency checks in comparison to a deterministic algorithm, sacrificing optimality. Next we will formalize our experience into an algorithm, the behavior of which we will analyze extensively in the section that follows.

Diagnosing a strong-fault model is known to be strictly more difficult than

a weak-fault model [Friedrich et al., 1990]. In many diagnostic instances this problem is alleviated by the fact that there exist, although without a guarantee, continuities in the diagnostic search space similar to the one in the weak-fault models. Let us discuss the process of finding a minimal diagnosis of the subtractor's strong-fault model SD_s and the observation α_2 (both from Sec. 2.1.1).

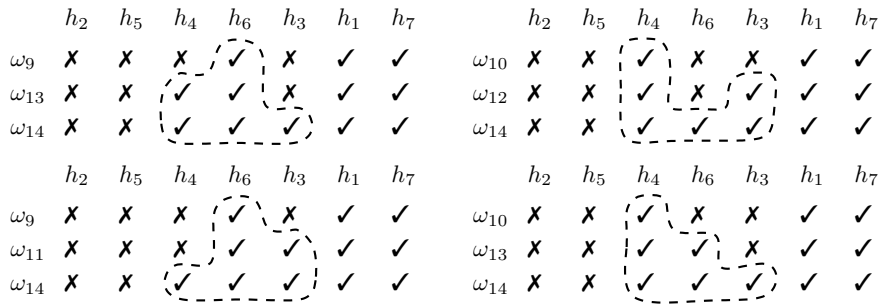


Figure 3.2: *Diagnoses of a strong-fault model*

Figure 3.2 illustrates the notion of continuity in a strong-fault model. In the four scenarios, a diagnostician starts from ω_{14} and improves the cardinality by changing the sign of a single literal in each diagnosis. At each step we receive one of the six distinct diagnoses of $\mathrm{SD}_s \wedge \alpha_2$ (these are $\omega_9, \omega_{10}, \ldots, \omega_{14}$ as shown in Fig. 3.2). Of these six diagnoses only ω_9 and ω_{10} are minimal such that $|\omega_9| = |\omega_{10}| = 3$. It is visible in Fig. 3.2 that component variables h_2 and h_5 are false in all diagnoses, while h_1 and h_7 are always true (healthy). Hence, any satisfying assignment of $\mathrm{SD}_s \wedge \alpha_2$ would contain $h_1 \wedge \neg h_2 \wedge \neg h_5 \wedge h_7$. Starting from the maximal-cardinality diagnosis ω_{14} (note that the all-faulty assignment in a strong-fault model is not necessarily a diagnosis), we must "flip" the variables h_3, h_4, and h_6 in order to reach the two minimal diagnoses. The key insight is that, as shown in Fig. 3.2, this is always possible by "flipping" a single literal at a time from health to faulty and receiving another consistent assignment (diagnosis).

In what follows we will formalize our experience so far in a stochastic algorithm for finding minimal diagnoses.

3.4.2 A Greedy Stochastic Algorithm

The randomized search process performed by SAFARI (cf. Alg. 1) has two parameters, M and N. There are N independent searches that start from randomly generated starting points. The algorithm tries to improve the cardinality of the initial diagnoses (while preserving their consistency) by randomly "flipping" fault literals. The change of a sign of literal is done in one direction only: from faulty to healthy.

Algorithm 1 SAFARI: A greedy stochastic hill climbing algorithm for approximating the set of minimal diagnoses.

```
1: function SAFARI(DS, α, M, N) returns a trie
       inputs: DS = ⟨SD, COMPS, OBS⟩, diagnostic system
               α, term, observation
               M, N, integers, climb restart limit and number of tries
       local variables: SD_cnf, CNF
                        m, n, integers, H, set of terms, improvements history
                        ω, ω′, terms, R, set of terms (initially ∅), result
2:     SD_cnf ← WFFTOCNF(SD)
3:     for n = 1, 2, ..., N do
4:         ω ← RANDOMDIAGNOSIS(SD_cnf, α)
5:         m ← 0, H ← ∅
6:         while m < M do
7:             ω′ ← IMPROVEDIAGNOSIS(ω, H)
8:             H ← H ∪ {ω′}
9:             if SD_cnf ∧ α ∧ ω′ ⊭ ⊥ then
10:                ω ← ω′, m ← 0
11:            else
12:                m ← m + 1
13:            end if
14:        end while
15:        unless ISSUBSUMED(R, ω) then
16:            ADDTOTRIE(R, ω)
17:            REMOVESUBSUMED(R, ω)
18:        end unless
19:    end for
20:    return R
21: end function
```

Each attempt to find a minimal diagnosis terminates after M unsuccessful attempts to "improve" the current diagnosis stored in ω. Thus, increasing M will lead to a better exploration of the search space and, possibly, to diagnoses of lower cardinality, while decreasing it will improve the overall speed of the algorithm.

SAFARI uses a number of utility functions. WFFTOCNF converts a propositional **Wff** in SD to CNF (cf. Sec 2.3). The IMPROVEDIAGNOSIS subroutine takes a term ω as an argument and changes the sign of a random negative literal in ω. If there are no negative literals, the function returns its original argument. IMPROVEDIAGNOSIS also uses the history of improvements H to avoid repetitions in randomly choosing the next literal. The history H is reset in each run of SAFARI (second assignment of line 5 in Alg. 1). Note that H never contains more than M terms (or flipped variables).

The implementation of RandomDiagnosis uses a modified DPLL solver returning a random SAT solution of $\mathrm{SD} \wedge \alpha$. Consider the original DPLL algorithm [Davis and Putnam, 1960] without the unit resolution rule. One can show that if, in the event of branching, the algorithm chooses polarity with equal probability, the DPLL algorithm is equally likely to compute any satisfiable solution (if such exists). Note that the order in which variables are assigned does not matter. Of course, the DPLL algorithm may end-up with a partial assignment, i.e., some of the variables are "don't care". This is not a problem because the partial assignment can be extended to a full satisfiable assignment by randomly choosing the signs of the unassigned variables from a uniform distribution. Taking the unit resolution rule into consideration does not change the likelihood of the modified DPLL solver finding a particular solution because it only changes the order in which variables are assigned. As most up-to-date SAT solvers are based on DPLL, creating a randomized DPLL solver that computes any satisfiable solution with equal probability is not difficult. Of course, random polarity decisions may negatively effect branching heuristics [Marques-Silva, 1999] but such analysis is beyond the scope of this chapter.

Similar to deterministic methods for MBD, Safari uses a SAT-based procedure for checking the consistency of $\mathrm{SD} \wedge \alpha \wedge \omega$. To increase the implementation efficiency of Safari, we combine a BCP-based LTMS engine [McAllester, 1990] and a full-fledged DPLL solver in two-stage consistency checking. Experimentation shows that combining LTMS and DPLL in such a way allows an order-of-magnitude Safari speed-up compared to pure DPLL, while the soundness and completeness properties of consistency checking are preserved.

We have implemented the two-stage consistency checking as follows. First, Safari calls a BCP-based LTMS [Forbus and de Kleer, 1993] to check if $\mathrm{SD} \wedge \alpha \wedge \omega \models \bot$. If the result is UNSAT then the candidate ω is not a diagnosis.[4] If the LTMS result is not UNSAT, it means that the consistency of the candidate is unknown and a call to a complete DPLL engine is needed. For the full DPLL checking we use POSIT [Freeman, 1995] or MiniSat [Eén and Sörensson, 2003].

Safari benefits from the two-stage SAT procedure because a typical MBD instance involves a large number of consistency checks ($O(|\mathrm{COMPS}|^2)$ for $N = 1, M = |\mathrm{COMPS}|$). As $\mathrm{SD} \wedge \alpha$ does not change during the search and each time only a small number of assumption clauses have to be updated, the incremental nature of LTMS greatly improves the search efficiency. Even though the DPLL running time per instance is the same as LTMS (DPLL performs BCP when doing unit propagation), DPLL construction is expensive and should be avoided when possible. DPLL initialization is typically slow as it involves building data structures for clauses and variables, counting literals, initializing conflict databases, etc. On the other hand, our implementation of LTMS is both incremental (does not have to be reinitialized before each consistency check) and efficient as it main-

[4] It can be shown that with a number of assumptions (well-formed Boolean circuits, **WFM** or **SFM**, full observations, etc.) if a BCP consistency check of $\mathrm{SD} \wedge \alpha \wedge \omega$ returns UNSAT, then the formula is UNSAT (the opposite is not necessarily true).

tains only counters for each clause. Each counter keeps the number of unassigned literals. Assigning a value to a variable requires decrementing some or all of the clause counters. If a counter becomes zero, a contradiction handler is signaled.

There is no guarantee that two diagnostic searches, starting from random diagnoses, would not lead to the same minimal diagnosis. To prevent this, we store the generated diagnoses in a trie R [Forbus and de Kleer, 1993], from which it is straightforward to extract the resulting diagnoses by recursively visiting its nodes. A diagnosis ω is added to the trie R by the function AddToTrie, iff no subsuming diagnosis is contained in R (the IsSubsumed subroutine checks on that condition). After adding a diagnosis ω to the resulting trie R, all diagnoses contained in R and subsumed by ω are removed by a call to RemoveSubsumed.

3.4.3 Basic Properties of the Greedy Stochastic Search

Before we continue with the topics of completeness and optimality we show that Safari is sound.

Lemma 2 (Soundness)**.** *All diagnoses returned by* Safari *are sound.*

Proof (Sketch). The consistency check in line 8 of Alg. 1 guarantees that only terms ω for which it holds that $\mathrm{SD} \wedge \alpha \wedge \omega \not\models \bot$ will be added to the result set R. According to Def. 5 these terms ω are diagnoses. □

One of the key factors in the success of the proposed algorithm is the exploitation of the continuity of the search-space of diagnosis models, where by continuity we mean that we can monotonically reduce the cardinality of a non-minimal diagnosis. Through the exploitation of this continuity property, Safari can be configured to guarantee finding a minimal diagnosis in weak fault models in polynomial number of calls to a satisfiability oracle.

The hypothesis which comes next is well studied in prior work [de Kleer et al., 1992a] as it determines the conditions in which minimal diagnoses represent all diagnoses of a model and an observation. This chapter is interested in the hypothesis from the computational viewpoint: it defines a class of models for which it is possible to establish a theoretical bound on the optimality and performance of Safari.

Hypothesis 1 (Minimal Diagnosis Hypothesis)**.** Let $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$ be a diagnostic system and ω' a diagnosis for an arbitrary observation α. The Minimal Diagnosis Hypothesis (MDH) holds in DS iff for any health assignment ω such that $Lit^{-}(\omega) \supset Lit^{-}(\omega')$, it holds that ω is also a diagnosis.

It is easy to show that MDH holds for all weak-fault models. There are other theories $\mathrm{SD} \notin \mathbf{WFM}$ for which MDH holds (e.g., one can directly construct a theory as a conjunction of terms for which MDH holds). Unfortunately, no necessary condition is known for MDH to hold in an arbitrary SD. The lemma which comes next is a direct consequence of MDH and weak-fault models.

Lemma 3. *Given a diagnostic system* DS = ⟨SD, COMPS, OBS⟩, SD ∈ **WFM**, *and a diagnosis* ω *for some observation* α, *it follows that* ω *is non-minimal iff another diagnosis* ω' *can be obtained by changing the sign of exactly one negative literal in* ω.

Proof (Sketch). From Def. 2 and SD ∈ **WFM**, it follows that if ω is a minimal diagnosis, any diagnosis ω' obtained by flipping one positive literal in ω is also a diagnosis. Applying the argument in the other direction gives us the above statement. □

SAFARI operates by performing subset flips on non-minimal diagnoses, attempting to compute minimal diagnoses. We next formalize this notion of flips, in order to characterize when SAFARI will be able to compute a minimal diagnosis.

Definition 11 (Subset Flip $\Phi_{\Downarrow}$)**.** Given a diagnostic system DS = ⟨SD, COMPS, OBS⟩ and a health assignment ω with a non-empty set of negative literals ($Lit^{-}(\omega) \neq \emptyset$), a subset flip $\Phi_{\Downarrow}$ turns one of the negative literals in ω to a positive literal, i.e., it creates a health assignment ω' with one more positive literal.

We next characterize flips based on whether they produce consistent models after the flip.

Definition 12 (Valid Subset Flip)**.** Given a system DS = ⟨SD, COMPS, OBS⟩, an observation α, and a non-minimal diagnosis ω, a subset flip $\Phi_{\Downarrow}$ is valid iff $\Phi_{\Downarrow}$ creates a health assignment that is also a diagnosis.

Given these notions, we can define continuity of the diagnosis search space in terms of literal flipping.

Definition 13 (Continuity)**.** A system model SD and an observation α satisfy the continuity property with respect to the set of diagnoses $\Omega^{\subseteq}(\text{SD} \wedge \alpha)$, iff for any diagnosis $\omega_k \in \Omega(\text{SD} \wedge \alpha)$ there exists a sequence $\Phi = \langle \omega_1, \omega_2, \cdots, \omega_{k-1}, \omega_k, \omega_{k+1}, \cdots, \omega_n \rangle$, such that for $i = 1, 2, \cdots, n-1$, it is possible to go from ω_i to ω_{i+1} via a valid subset flip, $\omega_i \in \Omega(\text{SD} \wedge \alpha)$, and $\omega_n \in \Omega^{\subseteq}(\text{SD} \wedge \alpha)$.

Given this definition of continuity, we can easily show the following two lemmata:

Lemma 4. *If* SD *satisfies MDH, then it satisfies the continuity property.*

Proof. Follows directly from Hypothesis 1 and Def 13. □

Lemma 5. SD ∈ **WFM** *satisfies the continuity property.*

Proof (Sketch). It is straightforward to show that if SD ∈ **WFM** then SD satisfies MDH. Then from Lemma 4 it follows that SD satisfies the continuous property. □

Our greedy algorithm starts with an initial diagnosis and then randomly flips faulty assumable variables. We now use the MDH property to show that, starting with a non-minimal diagnosis ω, the greedy stochastic diagnosis algorithm can monotonically reduce the size of the "seed" diagnosis to obtain a minimal diagnosis through appropriately flipping a fault variable from faulty to healthy; if we view this flipping as search, then this search is continuous in the diagnosis space.

Proposition 1. *Given a diagnostic system* $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$*, an observation* α*, and* $\mathrm{SD} \in \mathbf{WFM}$*,* SAFARI *configured with* $M = |\mathrm{COMPS}|$ *and* $N = 1$ *returns one minimal diagnosis.*

Proof. The diagnosis improvement loop starts, in the worst case, from a health assignment ω which is a conjunction of negative literals only. Necessarily, in this case, ω is a diagnosis as $\mathrm{SD} \in \mathbf{WFM}$. A diagnosis ω' that is subsumed by ω would be found with at most M consistency checks (provided that ω' exists) as M is set to be equal to the number of literals in ω and there are no repetitions in randomly choosing of which literal to flip next. If, after trying all the negative literals in ω, there is no diagnosis, then from Lemma 3 it follows that ω is a minimal diagnosis.

Through a simple inductive argument, we can continue this process until we obtain a minimal diagnosis. □

From Proposition 1 it follows that there is an upper bound of $O(|\mathrm{COMPS}|^2)$ consistency checks for finding a single minimal diagnosis. In most of the practical cases, however, we are interested in finding an approximation to *all* minimal-cardinality diagnoses. As a result the complexity of the optimally configured SAFARI algorithm becomes $O(|\mathrm{COMPS}|^2 S)$, where S is the number of minimal-cardinality diagnoses for the given observation. Section 3.6 discusses in more detail the computation of multiple minimal-cardinality diagnoses.

The number of assumable variables in a system of practical significance may exceed thousands, rendering an optimally configured SAFARI computationally too expensive. In Sec 3.5 we will see that while it is more computationally efficient to configure $M < |\mathrm{COMPS}|$, it is still possible to find a minimal diagnosis with high probability.

It is simple to show that flip-based search algorithms are complete for continuous diagnosis search spaces given weak fault models, i.e., $\mathrm{SD} \in \mathbf{WFM}$, and models that follow MDH, i.e., Lemma 3. We can formally characterize the guarantee of finding a minimal diagnosis with SAFARI in terms of a continuous diagnosis space. Note that this is a sufficient, but not necessary, condition; for example, we may configure SAFARI to flip multiple literals at a time to circumvent problems of getting trapped in discontinuous diagnosis spaces.

Theorem 1. *Given a diagnostic system* $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$*, and a starting diagnosis* ω*,* SAFARI *configured with* $M = |\mathrm{COMPS}|$ *and* $N = 1$ *is guaranteed to compute a minimal diagnosis if the diagnosis space is continuous.*

Proof. Given an initial diagnosis ω, SAFARI attempts to compute a minimal diagnosis by performing subset flips. If the diagnosis space is continuous, then we know that there exists a sequence of valid flips leading to a minimal diagnosis. Hence SAFARI is guaranteed to find a minimal diagnosis from ω. □

Finally, we show that SAFARI is guaranteed to compute all minimal diagnoses.

Theorem 2. *Consider a diagnostic system* $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$*, an observation* α*, and a continuous diagnosis space* $\Omega(\text{SD}, \alpha)$*. We denote the event of* SAFARI *computing all minimal diagnoses of* $\text{SD} \wedge \alpha$ *after* N *runs with* $\Upsilon(N)$*. It follows that* $\lim_{N \to \infty} Pr(\Upsilon(N)) = 1$*.*

Proof (Sketch). Since (1) the search space is continuous, (2) at each step there is a non-zero probability of flipping any unflipped literal, and (3) there is a polynomial upper bound of steps (|COMPS|) for computing a diagnosis, SAFARI can compute any non-minimal diagnosis with non-zero probability. Hence as $N \to \infty$, SAFARI will compute all minimal diagnoses. □

3.4.4 Complexity of Inference Using Greedy Stochastic Search

We next look at the complexity of SAFARI, and its stochastic approach to computing sound but incomplete diagnoses. We show that the primary determinant of the inference complexity is the consistency checking. SAFARI randomly computes a partial assignment π, and then checks if π can be extended to create a satisfying assignment during each consistency check, i.e., it checks the consistency of π with SD. This is solving the satisfiability problem (SAT), which is NP-complete [Cook, 1971]. We will show how we can use incomplete satisfiability checking to reduce this complexity, at the cost of completeness guarantees.

In the following, we call Ψ the complexity of a consistency check, and assume that there are γ components that can fail, i.e., $\gamma = |\text{COMPS}|$.

Lemma 6. *Given a diagnostic system* $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$ *with* $\text{SD} \in$ **WFM***, the worst-case complexity of finding any minimal diagnosis is* $O(\gamma^2 \Psi)$*, where* Ψ *is the cost of a consistency check.*

Proof. From Proposition 1 it follows that there is an upper bound of γ consistency checks for finding a single minimal diagnosis, since at each step (there is a maximum of γ steps for computing the "all healthy" diagnosis), the algorithm must flip at most γ literals. The total complexity is hence $O(\gamma^2 \Psi)$, since we perform a consistency check after each flip. □

The complexity of BCP is well-known allowing us to get more precise bounds on the worst-case complexity of computing one minimal-diagnosis with SAFARI. In what follows we will assume that SD is in CNF (cf. Sec. 2.3).

Lemma 7. *Given a diagnostic system* $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$, $\text{SD} \in \mathbf{WFM}$, *and* SD *having* c *clauses and* n *variables, the worst-case complexity under* **WFM** *of finding any minimal diagnosis is* $O(\gamma^2 cn)$ *when using BCP for consistency checks.*[5]

Proof (Sketch). An implementation of BCP [Forbus and de Kleer, 1993] maintains a total of c counters for the number of unsatisfied literals in each clause. A consistency check requires decrementing some or all counters for each of the n variables in SD. This gives us an upper bound of $O(cn)$ on the execution time of BCP. Combining the complexity of BCP with Lemma 6 gives us the desired result. □

In most practical cases, however, we are interested in finding an approximation to *all* minimal-cardinality diagnoses. As a result the complexity of the optimally configured SAFARI algorithm becomes $O\left(\gamma\binom{|\omega|}{\gamma}\Psi\right)$, where $|\omega|$ is the cardinality of the minimal-cardinality diagnoses for the given observation (cf. Sec. 3.7.7).

3.5 Optimality Analysis (Single Diagnosis)

In contrast to deterministic algorithms, in the SAFARI algorithm there is no absolute guarantee that the optimum solution (minimal diagnosis) is found for arbitrary values of the configuration parameter M ($M < |\text{COMPS}|$). Below we will provide an intuition behind the performance of the SAFARI algorithm by means of an approximate, analytical model that estimates the probability of reaching a diagnostic solution of specific minimality.

3.5.1 Optimality of SAFARI in Weak-Fault Models

We will start by considering a single run of the algorithm without retries where we will assume the existence of only one minimal diagnosis. Next, we will extend the model by considering retries.

Basic Model

Consider a diagnostic system $\text{DS} = \langle \text{SD}, \text{COMPS}, \text{OBS} \rangle$ such that $\text{SD} \in \mathbf{WFM}$ and an observation α such that α manifests only one minimal diagnosis ω. For the argument that follows we will configure SAFARI with $M = 1$, $N = 1$, and we will assume that the starting solution is the trivial "all faulty" diagnosis.

When SAFARI randomly chooses a faulty variable and flips it, we will be saying that it is a "success" if the new candidate is a diagnosis and, a "failure" otherwise. Let k denote the number of steps that the algorithm successfully traverses in the direction of the minimal diagnosis of cardinality $|\omega|$. Thus k also measures the

[5] More efficient implementations of BCP exist [Zhang and Stickel, 1996].

number of variables whose values are flipped from faulty to healthy in the process of climbing.

Let $f(k)$ denote the probability distribution function (pdf) of k. In the following we derive the probability $p(k)$ of successfully making a transition from k to $k+1$. A diagnosis at step k has k positive literals and still $|\text{COMPS}| - k$ negative literals. The probability of the next variable flip being successful equals the probability that the next negative to positive flip out of the $|\text{COMPS}| - k$ negative literals does not conflict with a negative literal belonging to a diagnosis solution ω. Consequently, of the $|\omega| - k$ literals only $|\text{COMPS}| - |\omega| - k$ literals are allowed to flip, and therefore the success probability equals:

$$p\,(k) = \frac{|\text{COMPS}| - |\omega| - k}{|\text{COMPS}| - k} = 1 - \frac{|\omega|}{|\text{COMPS}| - k} \tag{3.1}$$

The search process can be modeled in terms of the Markov chain depicted in Fig. 3.3, where k equals the state of the algorithm. Running into an inconsistency is modeled by the transitions to the state denoted "fail".

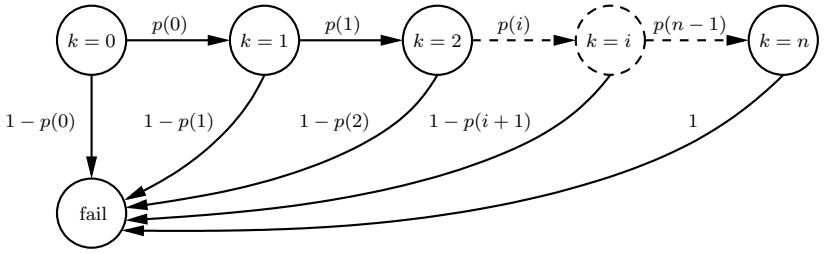


Figure 3.3: *Model of a* SAFARI *run for* $M = 1$ *and a single diagnosis* ω *(*$n = |\text{COMPS}| - |\omega|$*)*

The probability of exactly attaining step k (and subsequently failing) is given by:

$$f(k) = (1 - p(k+1)) \prod_{i=0}^{k} p(i) \tag{3.2}$$

After substituting (3.1) in (3.2) we receive the pdf of k:

$$f(k) = \frac{|\omega|}{|\text{COMPS}| - k + 1} \prod_{i=0}^{k} \left[1 - \frac{|\omega|}{|\text{COMPS}| - i} \right] \tag{3.3}$$

At the optimum goal state $k = |\text{COMPS}| - |\omega|$ the failure probability term in (3.3) is correct as it equals unity.

If p were independent of k, f would be according to a geometric distribution, which implies that chances of reaching the goal state $k = |\text{COMPS}| - |\omega|$ are slim. However, the fact that p decreases with k moves probability mass to the tail of the

distribution, which works in favor of reaching higher-k solutions. For instance, for single-fault solutions ($|\omega| = 1$) the distribution becomes uniform. Figure 3.4 shows the pdf for problem instances with $|\text{COMPS}| = 100$ for an increasing fault cardinality $|\omega|$. In order to decrease sampling noise, the empirical $f(k)$ values in Fig. 3.4 are computed by taking the average over 10 samples of k.

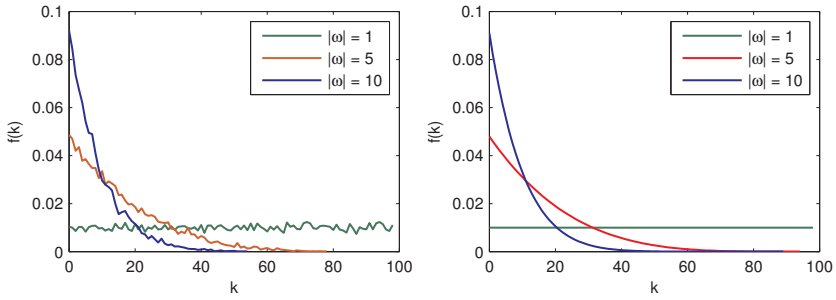


Figure 3.4: *Empirical (left) and analytic (right) $f(k)$ for no retries and a single diagnosis*

In the next section we show that retries will further move probability mass towards the optimum, providing the increasing distribution tail, needed for (almost always) reaching optimality.

Modeling Retries

In this section we extend the model to account for retries, which has a profound effect on the resulting pdf of f. Again, consider the transition between step k and $k+1$ where the algorithm can spend up to $m = 1, \ldots, M$ retries before bailing out. As can be seen by the algorithm (cf. Alg. 1), when a variable flip produces an inconsistency a retry is executed while m is incremented.

From elementary combinatorics we can compute the probability of having a diagnosis after flipping any of M different negative literals at step k. Similar to (3.1), at stage k there are $|\text{COMPS}| - k$ faulty literals from which M are chosen (as variable "flips" leading to inconsistency are recorded and not attempted again, there is no difference between choosing the M variables in advance or one after another). The probability of advancing from stage k to stage $k+1$ becomes:

$$p'(k) = 1 - \frac{\binom{|\omega|}{M}}{\binom{|\text{COMPS}|-k}{M}} \tag{3.4}$$

The progress of SAFARI can be modeled for values of $M > 1$ as a Markov chain, similar to the one shown in Fig. 3.3 with the transition probability of p replaced

by p'. The resulting pdf of the number of successful steps becomes:

$$f'(k) = \frac{\binom{|\omega|}{M}}{\binom{|\text{COMPS}|-k+1}{M}} \prod_{i=0}^{k} \left[1 - \frac{\binom{|\omega|}{M}}{\binom{|\text{COMPS}|-i}{M}} \right] \tag{3.5}$$

It can be seen that (3.3) is a private case of (3.5) for $M = 1$.

The retry effect on the shape of the pdf is profound. Whereas for single-fault solutions the shape for $M = 0$ is uniform, for $M = 1$ most of the probability mass is already located at the optimum $k = |\text{COMPS}| - |\omega|$. Fig. 3.5 plots f for a number of problem instances with increasing M. As expected, the effect of M is extremely significant. Note that in case of the real system, for $M = |\text{COMPS}|$ the pdf would consist of a single, unit probability spike at $|\text{COMPS}| - |\omega|$.

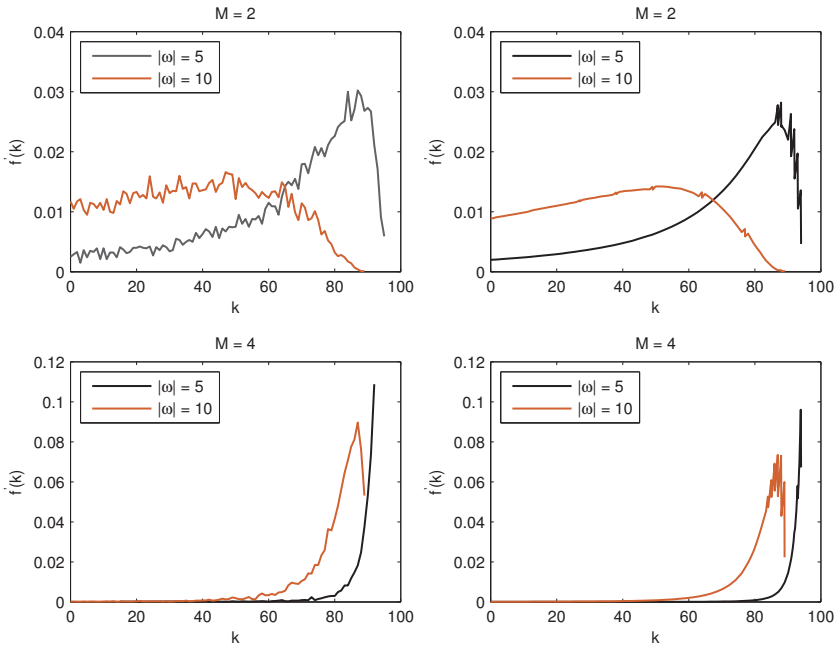


Figure 3.5: *Empirical (left) and analytic (right) $f'(k)$ for multiple retries and a single diagnosis*

Although we were unable to find an analytic treatment of the transition model above, the graphs immediately show that for large M the probability of moving to $k = |\text{COMPS}| - |\omega|$ is very large. Hence, we expect the pdf to have a considerable probability mass located at $k = |\text{COMPS}| - |\omega|$, depending on M relative to $|\text{COMPS}|$.

3.5.2 Optimality of SAFARI in Strong-Fault Models

From the above analysis we have seen that in **WFM** it is easy, starting from a non-minimal diagnosis, to reach a subset minimal diagnosis. As will be discussed in more detail below, this is not necessarily the case for strong-fault models. In many practical cases, however, strong-fault models exhibit, at least partially, behavior similar to MDH, thus allowing greedy algorithms like SAFARI to achieve results that are close to the optimal values.

Partial Continuity in Strong-Fault Stuck-At Models

In what follows we will restrict our attention to a large subclass of **SFM**, called **SFSM** [Struss and Dressler, 1992].

Definition 14 (Strong-Fault Stuck-At Model)**.** A system DS = ⟨SD, COMPS, OBS⟩ belongs to the class **SFSM** iff SD is equivalent to $(h_1 \Rightarrow F_1) \wedge (\neg h_1 \Rightarrow l_1) \wedge \cdots \wedge (h_n \Rightarrow F_n) \wedge (\neg h_n \Rightarrow l_n)$ such that $1 \leq i, j \leq n$, $\{h_i\} \subseteq$ COMPS, $F_j \in$ **Wff**, none of h_i appears in F_j, and l_j is a positive or negative literal in F_j.

MDH (cf. Hypothesis 1) does not hold for **SFSM** models. Consider an adder whose inputs and outputs are all zeroes, and whose gate models are all stuck-at-1 when faulty. In this case, the "all nominal" assignment is a diagnosis, but, for example, a stuck-at-1 output gate is not a diagnosis (there is a contradiction with the zero output).

Many practical observations involving **SFSM** models, however, lead to partial continuity. This means that there are groups of diagnoses that differ in at most one literal, i.e., a flip based search can improve the cardinality of a diagnosis. We next formalize this notion.

Definition 15 (Partial Continuity)**.** A system model SD and an observation α satisfy the partial continuity property with respect to a diagnosis ω_k, iff there exists a sequence $\Phi = \langle \omega_1, \omega_2, \cdots, \omega_{k-1}, \omega_k, \omega_{k+1}, \cdots, \omega_n \rangle$, such that for $i = 1, 2, \cdots, n-1$, it is possible to go from ω_i to ω_{i+1} via a valid subset flip, and $\{\omega_1, \omega_2, \cdots, \omega_n\} \subseteq \Omega(\text{SD} \wedge \alpha)$.

The differences between continuity (cf. Def. 13) and partial continuity are: (1) in Def. 13, the flipping leads to a minimal diagnosis, while in Def. 15, the flipping leads to any (not necessarily minimal) diagnosis, and (2) continuity holds with respect to any diagnosis, while partial continuity holds with respect to a specific diagnosis ω_k.

Note that the continuous property is trivally satisfied with respect to any diagnosis $\omega_k \in \Omega(\text{SD} \wedge \alpha)$, i.e., there always exists a sequence containing ω_k only ($\Phi = \langle \omega_k \rangle$). We are only interested in the non-trivial cases, for which $|\Phi| > 1$.

Consider a system SD and an observation α that satisfy the partial continuity property with respect to some diagnosis ω_k. We say that the diagnoses in the flip sequence Φ that contains ω_k form a continuous subspace. Alternatively, given a

diagnostic system SD and an observation α, a continuous diagnostic subspace of SD$\wedge\alpha$ is a set of diagnoses $\bar{\Omega} \subseteq \Omega(\text{SD}\wedge\alpha)$ with the property that, for any diagnosis $\omega \in \bar{\Omega}$, there is another diagnosis $\bar{\omega} \in \bar{\Omega}$ such that $|Lit^{-}(\omega)| - |Lit^{-}(\bar{\omega})| = \pm 1$.

Unfortunately, in the general **SFSM** case, we cannot derive bounds for the sizes of the continuous subspaces, and hence, for the optimality of SAFARI. In what follows, and with the help of a few examples, we illustrate the fact that partial continuity depends on the model and the observation and then we express the optimality of SAFARI as a function of this topologically-dependent property. Later, in Sec. 3.7, we collect empirical data that continuous subspaces leading to near-optimal diagnoses exist for a class of benchmark **SFSM** circuits.

Our first example illustrates the notion of partial discontinuity (dual to partial continuity). We show a rare example of a model and and an observation leading to a set of diagnoses that contains diagnoses of cardinality m and $m+q$ ($q>1$), but has no diagnoses of cardinality $m+1, m+2, \cdots, m+q-1$.

A Partial Discontinuity Example Consider, for example, the Boolean circuit shown in Fig. 3.6 and modeled by the propositional formula:

$$\text{SD}_d = \begin{cases} [h_1 \Rightarrow (y \Leftrightarrow \neg x)] \wedge [\neg h_1 \Rightarrow (y \Leftrightarrow x)] \\ [h_2 \Rightarrow (y \Leftrightarrow \neg x)] \wedge [\neg h_2 \Rightarrow (y \Leftrightarrow x)] \end{cases} \tag{3.6}$$

and an observation $\alpha_d = x \wedge \neg y$. Note, that $\text{SD}_d \notin$ **SFSM**. There are exactly two diagnoses of $\text{SD}_d \wedge \alpha_d$: $\omega_{15} = h_1 \wedge h_2$ and $\omega_{16} = \neg h_1 \wedge \neg h_2$. Note that this model cannot have single faults. As only ω_{15} is minimal, $|\omega_{15}| = 0$, and $|\omega_{16}| = 2$, if the algorithm starts from ω_{16} it is not possible to reach the minimal diagnosis ω_{15} by performing single flips. Similarly we can construct models which impose an arbitrarily bad bound on the optimality of SAFARI. Such models, however, are not common and we will see that the greedy algorithm performs well on a wide class of strong-fault models.

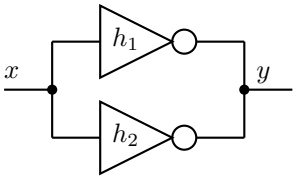


Figure 3.6: *A two inverters circuit*

Obviously, continuity in the distribution of the cardinalities in a set of diagnoses is a necessary (but not sufficient) condition for SAFARI to progress. Such models impose arbitrary difficulty to SAFARI, leading to suboptimal diagnoses of any cardinality.

An Example of Partial Continuity We continue the example started in Sec. 2.1.1. First, we create a system description SD_{sa} for a **SFSM** model. Let $\mathrm{SD}_{sa} = \mathrm{SD}_w \wedge \mathrm{SD}_f$, where SD_w is given by (2.1). The second part of SD_{sa}, the strong fault description SD_f, specifies that the output of a faulty gate must be stuck-at-1:

$$\begin{aligned}\mathrm{SD}_f = &(\neg h_1 \Rightarrow i) \wedge (\neg h_2 \Rightarrow d) \wedge (\neg h_3 \Rightarrow j) \wedge (\neg h_4 \Rightarrow m) \wedge \\ &\wedge (\neg h_5 \Rightarrow b) \wedge (\neg h_6 \Rightarrow l) \wedge (\neg h_7 \Rightarrow k)\end{aligned} \tag{3.7}$$

It is clear that $\mathrm{SD}_{sa} \in$ **SFSM**. We next compute the diagnoses of $\mathrm{SD}_{sa} \wedge \alpha_1$ ($\alpha_1 = x \wedge y \wedge p \wedge b \wedge \neg d$). There is one minimal diagnosis of $\mathrm{SD}_{sa} \wedge \alpha_1$ and it is $\omega_5^{\subseteq} = \neg h_1 \wedge h_2 \wedge h_3 \wedge \cdots \wedge h_7$ (cf. Fig. 3.7). If we choose the two literals h_3 and h_4 from $\omega_5^{\subseteq}$ and change the signs of h_3 and h_4, we create two new health assignments: $\omega_{15} = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge h_5 \wedge h_6 \wedge h_7$ and $\omega_{16} = \neg h_1 \wedge h_2 \wedge h_3 \wedge \neg h_4 \wedge h_5 \wedge h_6 \wedge h_7$. It can be checked that both ω_{15} and ω_{16} are diagnoses, i.e., $\mathrm{SD}_{sa} \wedge \alpha_1 \wedge \omega_{15} \not\models \bot$ and $\mathrm{SD}_{sa} \wedge \alpha_1 \wedge \omega_{16} \not\models \bot$. Note that ω_{15} and ω_{16} are diagnoses of the weak-part of the model, i.e., $\{\omega_{15}, \omega_{16}\} \subset \Omega(\mathrm{SD}_w \wedge \alpha_1)$. This follows from MDH and the fact that $\omega_5^{\subseteq}$ is a minimal diagnosis of $\mathrm{SD}_w \wedge \alpha_1$. Furthermore, ω_{15} is also a diagnosis in the strong-fault stuck-at model ($\omega_{15} \in \Omega(\mathrm{SD}_{sa} \wedge \alpha_1)$) because $\mathrm{SD}_w \wedge \alpha_1 \wedge \neg h_3$ does not lead to a contradictory value for j in the strong-fault part SD_f. A similar argument applies to ω_{16}: $\mathrm{SD}_w \wedge \alpha_1 \wedge \neg h_4$ does not contradict m in SD_f. Equivalently, if negating h_3 in $\omega_5^{\subseteq}$, which makes j stuck-at-1, results in a diagnosis, and negating h_4 in $\omega_5^{\subseteq}$, which makes m stuck-at-1, also results in a diagnosis, negating *both* h_3 and h_4 in $\omega_5^{\subseteq}$ will also result in a diagnosis (consider the fact that the fault mode of h_4 sets m only, but does not impose constraints on j). The above argument can be extended similarly to h_5, h_6, and h_7. Hence, any assignment of COMPS containing $\neg h_1 \wedge h_2$ is a diagnosis of $\mathrm{SD}_{sa} \wedge \alpha_1$, no matter what combination of signs we take for h_3, h_4, h_5, h_6, and h_7. Note that a health assignment containing $\neg h_4$ is a diagnosis conditioned on $k = 1$.

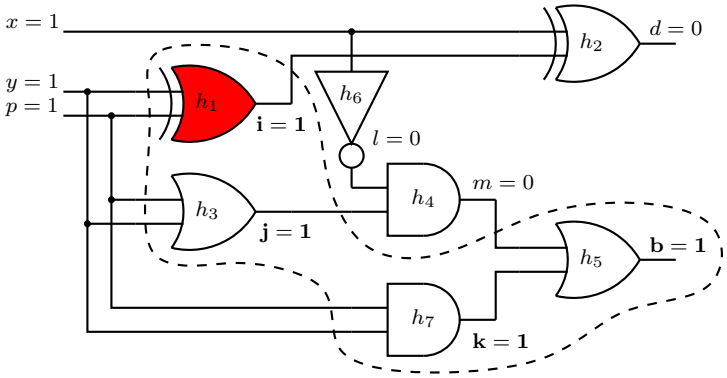


Figure 3.7: *Continuous subspace in a strong-fault, stuck-at-*1 *model of a subtractor*

Consider an alternative way of computing a set of ambiguous diagnoses of

$\mathrm{SD}_{sa} \wedge \alpha_1$. Given $\mathrm{SD}_{sa} \wedge \alpha_1 \wedge \omega_5^{\subseteq}$, we can compute a consistent assignment to all internal variables (for example by propagation). There is exactly one such assignment ϕ and it is $\phi = i \wedge j \wedge k \wedge \neg l \wedge \neg m$, $\mathrm{SD}_{sa} \wedge \alpha_1 \wedge \omega_5^{\subseteq} \wedge \phi \not\models \bot$ (cf. Fig. 3.7). Note that for components h_1, h_3, h_5, and h_7, a change in the state of a component (healthy or faulty) does not lead to a different output value. For example the output j of the h_3 or-gate is 1 because the gate is healthy and its inputs are 1 but j would also be 1 for a stuck-at-1 or-gate ($\neg h_3$). As a result, no diagnostic reasoner can determine if the components in the dashed region of Fig. 3.7 are healthy or faulty (stuck-at-1). Equivalently, one can change the signs of h_3, h_5, and h_7 in the diagnosis $\omega_5^{\subseteq}$ and the resulting assignments are still diagnoses. We call the set of components modeled by h_1, h_3, h_5, and h_7 an ambiguity group. Clearly, SAFARI can start from a diagnosis $\omega_{17} = \neg h_1 \wedge h_2 \wedge \neg h_3 \wedge h_4 \wedge \neg h_5 \wedge h_6 \wedge \neg h_7$ ($|\omega_{17}| = 4$) and reach $\omega_5^{\subseteq}$ ($|\omega_5^{\subseteq}| = 1$) by performing valid subset flips.

We hypothesize that the existence of component ambiguity groups, and hence, continuous subspaces can be also explained with the concept of cones [de Kleer, 2008; Siddiqi and Huang, 2007]. Loosely, a cone in ATPG [Bushnell and Agrawal, 2000] is a set of gates in a circuit, having one circuit output only.

To make our reasoning precise, we restrict the class of **SFSM** models to exclude malformed circuits such as ones having disconnected inputs or outputs, etc. Furthermore, we assume that each component has exactly one output (the set of all component output variables is denoted as COUT). The latter is not a big restriction as multi-output component models can be replaced by multiple components, each having a single output.[6]

Definition 16 (Well-Formed Diagnostic System (**Wfds**))**.** The diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$ is well-formed ($\mathrm{DS} \in$ **Wfds**) iff for any observation α such that $\mathrm{SD} \wedge \alpha \not\models \bot$, and for any diagnosis ω ($\omega \in \Omega(\mathrm{SD} \wedge \alpha)$), there is exactly one assignment ϕ to all component outputs COUT such that $\mathrm{SD} \wedge \alpha \wedge \omega \wedge \phi \not\models \bot$.

At this point we have each component in a model producing either a nominal output or a stuck-at value. It is easy to define the subset of components for which these two values coincide.

Definition 17 (Component Ambiguity Group)**.** Given a system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, $\mathrm{SD} \in$ **SFSM**, $\mathrm{SD} \in$ **Wfds**, an observation α, and a diagnosis $\omega \in \Omega(\mathrm{SD} \wedge \alpha)$, the component ambiguity group $\mathcal{U}(\mathrm{DS}, \alpha, \omega)$, $\mathcal{U} \subseteq \mathrm{COMPS}$ is defined as the set of components whose output values are the same as the stuck-at value of SD.

Finally, we show that a component ambiguity group leads to a continuous subspace. In the general case we cannot say much about the size of the component ambiguity groups. From experimentation, we have noticed that it is difficult to assign the inputs of an **SFSM** to values that generate small continuous subspaces

[6] Any multi-output Boolean function can be replaced by a composition of single-output Boolean functions.

(either SD $\wedge \alpha \models \perp$, or SD $\wedge \alpha$ leads to large component ambiguity groups). Of course, it is possible to consider an adder, or a multiplier, for example, whose inputs are all zeroes and whose gate models are all stuck-at-1 when faulty, but the number of such inputs/circuit combinations is small.

Proposition 2. *A diagnostic system* SD, SD $\in$ **SFSM**, SD $\in$ **Wfds**, *and an observation* α *entail continuous diagnostic subspaces.*

Proof. From Def. 16 and the fact that SD $\in$ **Wfds** it follows that the output values of a subset of the components have the same sign as the model's stuck-at value. We denote this set as COMPS$'$, COMPS$' \subseteq$ COMPS. Any health assignment $\bar{\omega}$ that differs only in signs of components belonging to COMPS$'$ is also a diagnosis. If the set of diagnoses of SD$\wedge\alpha$ contains all possible assignments to the assumables in COMPS$'$ then those diagnoses form a continuous space (cf. Def. 17). □

To best illustrate Proposition 2, consider the or-gate modeled by h_3 in Fig. 3.7. Its output is 1 either because the gate is healthy and one of the gate's inputs is 1 or because the gate is stuck-at-1. In this situation, it is not possible to determine if the component is healthy or faulty.

Clearly, $|\mathcal{U}(\text{DS}, \alpha, \omega)|$ is a lower bound for the progress of SAFARI in stuck-at models. It can be shown that if SAFARI starts from a diagnosis ω of maximum cardinality for the given subspace, SAFARI is guaranteed (for $M = |\text{COMPS}|$) to improve the cardinality of ω by at least $|\mathcal{U}(\text{DS}, \alpha, \omega)|$. In practice, SAFARI can proceed even further as the stuck-at ambiguity groups are only one factor of diagnostic uncertainty. A stuck-at component effectively "disconnects" inputs from outputs, hence gates from the fan-in region are not constrained. For instance, continuing our example, for $\neg h_5$, all predecessors in the cone of $\neg h_5$ (components $\neg h_3$, $\neg h_4$, $\neg h_5$, $\neg h_6$, and $\neg h_7$) constitute a continuous health subspace. Contrary to a component ambiguity group, this set is conditional on the health state of another component. A thorough study of stuck-at continuity is outside the scope of this thesis but as we shall see in Sec. 3.7, continuous subspaces justify SAFARI experiments on stuck-at models.

Performance Modeling with Stuck-At Models

To further study the optimality of SAFARI in strong-fault models, we first define a case in which the algorithm cannot improve a non-minimal diagnosis by changing the sign of a faulty literal. Note that the existence of such cases is not a sufficient condition for SAFARI to be suboptimal, as it is possible to reach a minimal diagnosis by first changing the sign of some other faulty literal, thus "circumventing" the missing diagnosis.

From the preceding section we know that the number of "invalid flips" does not depend on k, i.e., it is determined by the observation vector and the fault

modes. The probability of SAFARI to progress from any non-minimal diagnosis becomes

$$p(k) = 1 - \frac{\binom{|\omega|+|X|}{M}}{\binom{|\text{COMPS}|-k}{M}} \tag{3.8}$$

where $|X|$ is the number of "invalid flips". The ratio of the number of "invalid flips" $|X|$ to |COMPS| we will call **SFM** density d. The density d gives the average probability of trying an "invalid flip" throughout the diagnostic search. An approximation of the probability of success of SAFARI is:

$$p(k) = 1 - \frac{\binom{|\omega|}{M}}{\binom{|\text{COMPS}|-k}{M}} - d \tag{3.9}$$

Plugging p into (3.2) allows us to predict $f(k)$ for the **SFM** models for which our assumptions hold. This pdf, both measured from an implementation of SAFARI and generated from (3.2) and (3.9) is shown in Fig. 3.8 for different values of the density d.

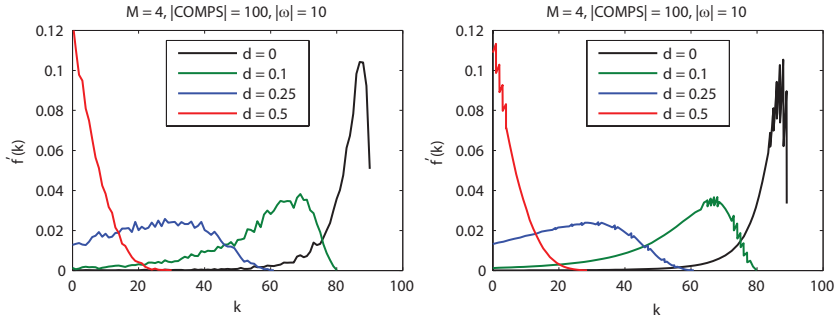


Figure 3.8: *Empirical (left) and analytic (right) $f'(k)$ for various diagnostic densities, multiple retries and a single diagnosis*

From Fig. 3.8 it is visible that increasing the density d leads to a shift of the probability density of the length of the walk k to the left. The effect however is not that profound even for large values of d and is easily compensated by increasing M as discussed in the preceding sections.

It is interesting to note that bounds on d can be computed from SD (independent of α) and these bounds can be used further for increasing the performance of SAFARI.

3.5.3 Validation

In the preceding sections we have illustrated the progress of SAFARI with synthetic circuits exposing specific behavior (diagnoses). In the remainder of this section

we will plot the pdf of the greedy search on one of the small benchmark circuits (for more information on the 74181 model cf. Sec. 3.7).

The progress of SAFARI with a weak-fault model of the 74181 circuit is shown in Fig. 3.9. We have chosen a difficult observation leading to a minimal diagnosis of cardinality 7 (left) and an easy observation leading to a single fault diagnosis (right). Both plots show that the probability mass shifts to the right when increasing M and the effect is more profound for the smaller cardinality.

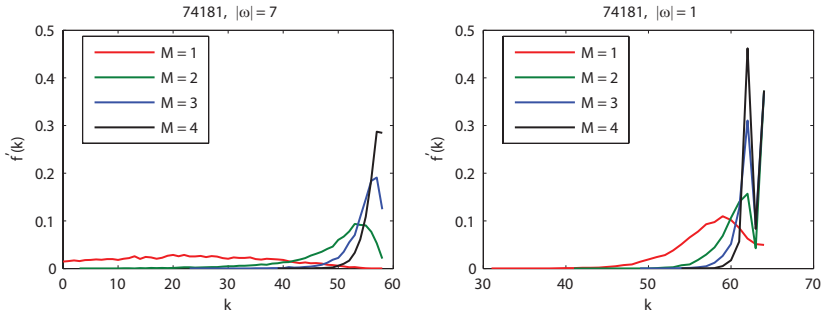


Figure 3.9: *Empirical $f'(k)$ for a weak-fault model of the 74181 circuit with observations leading to two different minimal-cardinality diagnoses and various M*

The effect of the stuck-at-0 and stuck-at-1 fault modes (**SFM**) on the probability of success of SAFARI is shown in Fig. 3.10.

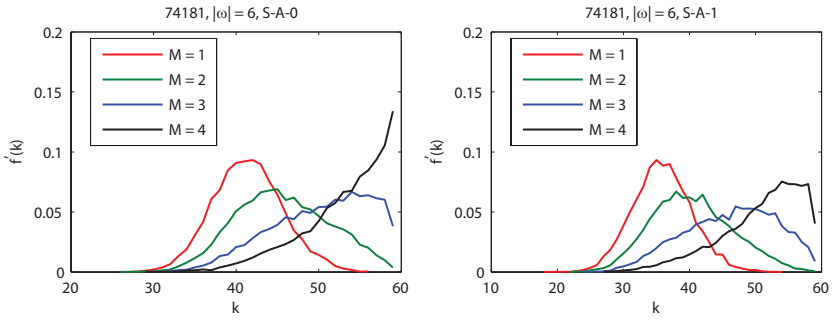


Figure 3.10: *Empirical $f'(k)$ for stuck-at-0 and stuck-at-1 strong-fault models of the 74181 circuit with various M*

Obviously, in this case the effect of increasing M is smaller, although still depending on the difficulty of the observation vector. Last, even for small values of M, the absolute probability of SAFARI finding a minimal diagnosis is sizeable, allowing the use of SAFARI as a practical *anytime* algorithm which always re-

turns a diagnosis, the optimality of which depends on the time allocated to its computation.

3.6 Optimality Analysis (Multiple Diagnoses)

The preceding section described the process of computing one diagnosis with SAFARI ($N = 1$). In this section we discuss the use of SAFARI in computing (or counting) all minimal-cardinality diagnoses ($N > 1$). For the rest of the section we will assume that SAFARI is configured with $M = |\text{COMPS}|$.

Consider a system description SD (SD $\in$ **WFM**) and an observation α. The number of minimal diagnoses $|\Omega^{\subseteq}(\text{SD} \wedge \alpha)|$ can be exponential in $|\text{COMPS}|$. Furthermore, in practice, diagnosticians are interested in sampling from the set of minimal-cardinality diagnoses $\Omega^{\leq}(\text{SD} \wedge \alpha)$ (recall that $\Omega^{\leq}(\text{SD} \wedge \alpha) \subseteq \Omega^{\subseteq}(\text{SD} \wedge \alpha)$) as the minimal-cardinality diagnoses cover a significant part of the *a posteriori* diagnosis probability space [de Kleer, 1990]. In what follows, we will see that SAFARI is very well suited for that task.

Theorem 3. *The probability of* SAFARI *configured with* $M = |\text{COMPS}|$ *computing a minimal diagnosis of cardinality* $|\omega|$ *in a system with* $|\text{COMPS}|$ *component variables approaches* $|\text{COMPS}|^{-|\omega|}$ *for* $|\text{COMPS}|/|\omega| \to \infty$.

Proof (Sketch). Assume a minimal diagnosis of cardinality $|\omega|$ exists. From Proposition 1 it follows that SAFARI configured with $M = |\text{COMPS}|$ is guaranteed to compute minimal diagnoses. Starting from the "all faulty" assignment, consider a step k in "improving" the diagnosis cardinality. If state k contains more than one diagnosis, then at state $k+1$, SAFARI will either (1) flip a literal belonging to this diagnosis (note that a literal may belong to more than one diagnosis) and subsequently prevent SAFARI of reaching this diagnosis or (2) flip a literal belonging to a diagnosis which has already been invalidated (i.e., one or more of its literals have been flipped at an earlier step).

The probability that a solution of cardinality $|\omega|$ "survives" a flip at iteration k (i.e., is not invalidated) is:

$$p\,(k) = 1 - \frac{|\omega|}{|\text{COMPS}| - k} = \frac{|\text{COMPS}| - |\omega| - k}{|\text{COMPS}| - k} \tag{3.10}$$

Similarly to our basic model (Sec. 3.5.1), the probability that a diagnosis ω "survives" until it is returned by the algorithm:

$$f(|\text{COMPS}| - |\omega| - 1) = \prod_{i=0}^{|\text{COMPS}|-|\omega|-1} p(i) = \tag{3.11}$$

$$= \prod_{i=0}^{|\text{COMPS}|-|\omega|-1} \frac{|\text{COMPS}| - |\omega| - i}{|\text{COMPS}| - i} \tag{3.12}$$

Rewriting the right hand side of Eq. (3.11) gives us:

$$f(|\text{COMPS}| - |\omega| - 1) = \frac{(|\text{COMPS}| - |\omega|)!}{(|\omega| + 1)(|\omega| + 2) \cdots |\text{COMPS}|} = \tag{3.13}$$

$$= \frac{|\omega|!(|\text{COMPS}| - |\omega|)!}{|\text{COMPS}|!} \tag{3.14}$$

Since

$$\frac{(|\text{COMPS}| - |\omega|)!}{|\text{COMPS}|!} = \tag{3.15}$$

$$= \frac{1}{(|\text{COMPS}| - |\omega| + 1)(|\text{COMPS}| - |\omega| + 2) \cdots |\text{COMPS}|} \tag{3.16}$$

it holds that

$$\lim_{|\text{COMPS}|/|\omega| \to \infty} \frac{(|\text{COMPS}| - |\omega|)!}{|\text{COMPS}|!} = |\text{COMPS}|^{-|\omega|} \tag{3.17}$$

As a result, for small $|\omega|$ relative to $|\text{COMPS}|$,

$$f(|\text{COMPS}| - |\omega| - 1) = |\omega|!|\text{COMPS}|^{-|\omega|} \tag{3.18}$$

which gives us the above theorem. □

The distribution $h_i(|\omega|)$ of the cardinalities of the minimal diagnoses in $\Omega^{\subseteq}(\text{SD} \wedge \alpha)$ depends on the topology of SD and on α; i.e., we can create SD and α having any $h_i(|\omega|)$. We denote the cardinality distribution of the minimal diagnoses computed by SAFARI as $h(|\omega|)$.

Theorem 3 gives us a termination criterion for SAFARI which can be used for enumerating and counting minimal-cardinality diagnoses. Instead of running SAFARI with a fixed N it is sufficient to compute the area under the output distribution function $\sum h$. This value will converge to a single value, hence we can terminate SAFARI after the change of $\sum h$ drops below a fixed threshold. Note that SAFARI is efficient in enumerating the minimal-cardinality diagnoses as they are computed with a probability that is exponentially higher than that of the probability of computing minimal diagnoses of higher-cardinality.

Corollary 1. SAFARI *computes diagnoses of equal cardinality with equal probability.*

Proof (Sketch). From Theorem 3 it follows that the probability of success f of SAFARI in computing a specific diagnosis ω depends only on $|\omega|$ and not on the actual composition of ω. □

The above corollary gives us a simple termination criterion for SAFARI in the cases when all minimal diagnoses are also minimal-cardinality diagnoses; it can

be proven that in this case all minimal-cardinality diagnoses are computed with the same probability.

We will see that, given an input cardinality distribution $h_i(|\omega|)$, SAFARI produces an output distribution $h(|\omega|)$ that is highly skewed to the right due to Theorem 3. To facilitate the study of how SAFARI transforms $h_i(|\omega|)$ into $h(|\omega|)$ we will use a Monte Carlo simulation of SAFARI. The advantage is that the Monte Carlo simulation is much simpler for analysing the run-time behavior of SAFARI than studying the algorithm itself.

Algorithm 2 Monte Carlo simulation of SAFARI.

```
1:  function SafariSimulate(Ω^⊆, N) returns a cardinality distribution
        inputs: Ω^⊆, a set of minimal diagnoses
                N, integer, number of tries
        local variables: h_i, vector, cardinality distributions
                         h, vector, cardinality distribution
                         b, vector, fault distribution,
                         n, integer, current run
                         i, integer
                         c, integer
2:      h_i ← CardinalityDistribution(Ω^⊆)
3:      for n ← 1, 2, ..., N do
4:          for c ← 1, 2, ..., |h_i| do
5:              b[c] ← c · h_i[c]
6:          end for
7:          for i ← 1, 2, ..., |Ω^⊆| do
8:              c ← DiscreteInverseRandomValue (b / Σb)
9:              b[c] ← b[c] − c
10:         end for
11:         h[c] ← h[c] + 1
12:     end for
13:     return h
14: end function
```

Algorithm 2 simulates which diagnoses from the input set of minimal diagnoses Ω are "reached" by SAFARI in N tries. The auxiliary subroutine CARDINALITYDISTRIBUTION computes the input distribution h_i by iterating over all diagnoses in $\Omega^{\subseteq}$. We store the input cardinality distribution h_i and the resulting cardinality distribution h in vectors (note the vector sums in lines 7 and 8 and the division of a vector by scalar in line 8).

The outermost loop of Alg. 2 (lines 3 – 12) simulates the N runs of SAFARI. This is done by computing and updating an auxiliary vector b, which contains the distribution of the component variables in $\Omega^{\subseteq}$ according to the cardinalities of

the diagnoses these variables belong to. Initially, b is initialized with the number of literals in single faults in position 1, the number of literals in double faults in position 2 (for example if there are three double faults in h_i, $b[2] = 6$), etc. This is done in lines 4 – 6 of Alg. 2. Note that we assume that diagnoses do not share literals. This restriction can be easily dropped by counting all the assumables in the input $\Omega^{\subseteq}$ (the latter assumption does not change the results of this section).

Lines 7 – 10 simulate the process of the actual bit flipping of SAFARI. At each step the simulation draws a random literal from the probability distribution function (*pdf*) $\frac{b}{\sum b}$; this is done by the DISCRETEINVERSERANDOMVALUE function in line 8. Each bit flip "invalidates" a diagnosis from the set $\Omega^{\subseteq}$, i.e., a diagnosis of cardinality c cannot be reached by SAFARI. After a diagnosis has been "invalidated", the vector b is updated, for example, if the simulation "invalidates" a quadruple fault, $b[4] = b[4] - 4$ (line 9). Note that the number of iterations in the loop in lines 7 – 10 equals the number of diagnoses in $\Omega^{\subseteq}$. As a result after terminating this loop, the value of the integer variable c is equal to the cardinality of the *last* "invalidated" diagnosis. The latter is the diagnosis which SAFARI computes in this run. What remains is to update the resulting pdf with the right cardinality (line 11).

The simulation in Alg. 2 links the distribution of the actual diagnoses in $\Omega^{\subseteq}$ to the distribution of the cardinalities of the diagnoses returned by SAFARI. As $\Omega^{\subseteq}$ can be arbitrarily set, we will apply Alg. 2 to a range of typical input distributions. The results of the simulation as well as the results of running SAFARI on synthetic problems with the same input distributions are shown in Fig. 3.11.

Fig. 3.11 shows (1) that Alg. 2 predicts the actual behavior of SAFARI (compare the second and third column of plots), and (2) that SAFARI computes diagnoses of small cardinality in agreement with Theorem 3. The only case when the output distribution is not a steep exponential is when the cardinalities in the set of the input minimal diagnoses grow exponentially. Table 3.2 summarizes the parameters of exponential fits for the input cardinality distributions shown in Fig. 3.11 (a is the initial (zero) cardinality, λ is the decay constant, and R^2 is the coefficient of determination).

Table 3.2: *Fit coefficients to exponential and goodness of fit for the cardinality distribution in Fig. 3.11*

Input Distribution	a	λ	R^2
Uniform	576	−0.44	1
Normal	423	−0.34	0.99
Exponential	69 470	−4.26	1
Reverse Exponential	385	−0.33	0.95

We have seen that SAFARI is suited for computing multiple diagnoses of small probability. In the next section we provide experimental results from the imple-

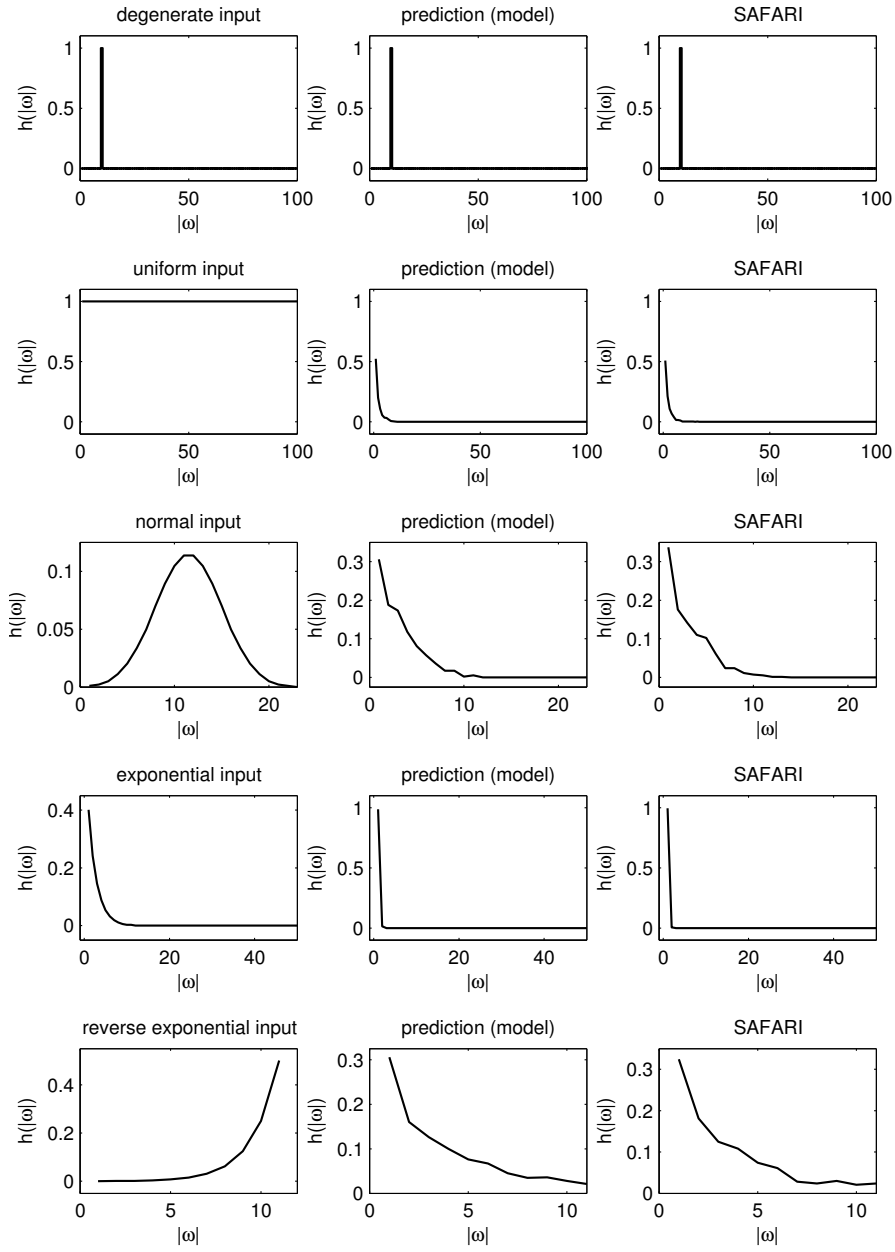


Figure 3.11: *Predicted and actual cardinality distributions*

mentation of SAFARI and related algorithms.

3.7 Experimental Results

This section discusses empirical results measured from an implementation of SAFARI. In order to compare the optimality and performance of SAFARI to various diagnostic algorithms, we have performed more than a million diagnosis computations on 64 dual-CPU nodes belonging to a cluster. Each node contains two 2.4 GHz AMD Opteron DP 250 processors and 4 Gb of RAM.

The default configuration of SAFARI (when not stated otherwise) was $M = 8$ and $N = 4$; that is, SAFARI is configured for a maximum number of 8 retries before giving up the climb, and a total of 4 attempts. To provide more precise average run-time optimality and performance data, all stochastic algorithms (e.g., ones based on SLS Max-SAT and SAFARI) have been repeatedly run 10 times on each model and observation vector.

3.7.1 Implementation Notes and Test Set Description

We have implemented SAFARI in approximately 1 000 lines of C code (excluding the LTMS, interface, and DPLL code) and it is a part of the LYDIA package.[7]

We have performed our experiments on the benchmark circuits described in Sec. 2.4. In order to provide both weak- and strong-fault cases, we have translated each circuit to a weak, stuck-at-0 (S-A-0), and stuck-at-1 (S-A-1) model. In the stuck-at models, the output of each faulty gate is assumed to be the same constant (cf. Def. 14).

The performance of diagnostic algorithms depends to various degrees on the observation vectors (algorithm designers strive to produce algorithms, the performance of which is not dependent on the observation vectors). Hence, we have performed our experimentation with a number of different observations for each model. We have implemented an algorithm (Alg. 3) that generates observations leading to diagnoses of different minimal-cardinality, varying from 1 to nearly the maximum for the respective circuits (for the 74XXX models it is the maximum). The experiments omit nominal scenarios as they are trivial from the viewpoint of MBD.

Algorithm 3 uses a number of auxiliary functions. RANDOMINPUTS (line 3) assigns uniformly distributed random values to each input in IN (note that for the generation of observation vectors we partition the observable variables OBS into inputs IN and outputs OUT and use the input/output information which comes with the original 74XXX/ISCAS85 circuits for simulation). Given the "all healthy" health assignment and the diagnostic system, COMPUTENOMINALOUTPUTS (line

[7] LYDIA, SAFARI, and the diagnostic benchmark are freely available for download at `http://fdir.org/lydia/`.

Algorithm 3 Algorithm for generation of observation vectors.

```
1:  function MakeAlphas(DS, N, K) returns a set of observations
        inputs: DS = ⟨SD, COMPS, OBS⟩, diagnostic system
                OBS = IN ∪ OUT, IN ∩ OUT = ∅
                N, integer, number of tries for Safari
                K, integer, maximal number of diagnoses per cardinality
        local variables: α, β, α_n, ω, terms
                         c, integer, best cardinality so far
                         A, set of terms (observation vectors), result
2:      for k ← 1, 2, ..., K do
3:          α ← RandomInputs(IN)
4:          β ← ComputeNominalOutputs(DS, α)
5:          c ← 0
6:          for all v ∈ OUT do
7:              α_n ← α ∧ Flip(β, v)
8:              ω ← MinCardinalityDiagnosis(Safari(SD, α_n, |COMPS|, N))
9:              if |ω| > c then
10:                 c ← |ω|
11:                 A ← A ∪ α_n
12:             end if
13:         end for
14:     end for
15:     return A
16: end function
```

4) performs simulation by propagating the input assignment α. The result is an assignment β which contains values for each output variable in OUT.

The loop in lines 7 – 14 increases the cardinality by greedily flipping the values of the output variables. For each new candidate observation α_n, Alg. 3 uses the diagnostic oracle Safari to compute a minimal diagnosis of cardinality c. As Safari returns more than one diagnosis (up to N), we use MinCardinality-Diagnosis to choose an arbitrary diagnosis of the smallest cardinality. If the cardinality c of this diagnosis increases in comparison to the previous iteration, the observation is added to the list.

By running Alg. 3 we get up to K observations leading to a single-fault, similar number of observations leading to a double-fault, etc., up to cardinality m, where m is the cardinality of the MFMC diagnosis [Feldman et al., 2008b] for the respective model. Algorithm 3 clearly shows a bootstrapping problem. In order to create potentially "difficult" observations for Safari we require Safari to solve those "difficult" observations. Although we have seen in Sec. 3.6 that Safari is heavily biased towards generating diagnoses of small cardinality, there is no guarantee. To alleviate this problem, for the generation of observation vectors, we have

configured SAFARI to compute subset-minimal diagnoses with $M = |\text{COMPS}|$ and N increased to 20.

Table 3.3: *Number of observation vectors for different model classes*

Name	Weak	stuck-at-0	stuck-at-1
74182	250	150	82
74L85	150	58	89
74283	202	202	202
74181	350	143	213
c432	301	301	301
c499	835	235	835
c880	1 182	217	335
c1355	836	836	836
c1908	846	846	846
c2670	1 162	134	123
c3540	756	625	743
c5315	2 038	158	228
c6288	404	274	366
c7552	1 557	255	233

Table 3.3 shows the number of observation vectors we have used for each model. The second, third, and fourth columns show the number of observation vectors with which we have tested the weak, S-A-0, and S-A-1 models, respectively. For the stuck-at models, we have simply chosen those weak-fault model observations that are consistent with their respective system descriptions (as in strong-fault models it is often the case that $\text{SD} \wedge \alpha \models \bot$, we have not considered such scenarios).

3.7.2 Comparison to Complete Algorithms

Table 3.4 shows the results from comparing SAFARI to implementations of two state-of-the-art complete and deterministic diagnostic algorithms: a modification for completeness of CDA* [Williams and Ragno, 2007] and HA* [Feldman and van Gemund, 2006]. Table 3.4 shows, for each model and for each algorithm, the percentage of all tests for which a diagnosis could be computed within 1 min cut-off time.

As it is visible from the three rightmost columns of Table 3.4, SAFARI could find diagnoses for all observation vectors, while the performance of the two deterministic algorithms (columns two to seven) degraded with the increase of the model size and the cardinality of the observation vector. Furthermore, we have observed a degradation of the performance of CDA* and HA* with increased cardinality of the minimal-cardinality diagnoses, while the performance of SAFARI

Table 3.4: *Comparison of CDA*, HA*, and* Safari *[% of tests solved]*

	CDA*			HA*			Safari		
Name	Weak	S-A-0	S-A-1	Weak	S-A-0	S-A-1	Weak	S-A-0	S-A-1
74182	100	100	100	100	100	100	100	100	100
74L85	100	100	100	100	100	100	100	100	100
74283	100	100	100	100	100	100	100	100	100
74181	79.1	98.6	97.7	100	100	100	100	100	100
c432	74.1	75.4	73.1	71.1	94.7	69.1	100	100	100
c499	29	45.5	27.7	24.1	77.9	25.9	100	100	100
c880	11.6	44.7	32.2	12.4	62.2	41.5	100	100	100
c1355	3.8	4.7	5.4	10.8	10.6	12.2	100	100	100
c1908	0	0	0	6.1	6	6.5	100	100	100
c2670	0	0	0	5	64.2	44.7	100	100	100
c3540	0	0	0	1.1	3.8	2.2	100	100	100
c5315	0	0	0	1.1	8.2	5.7	100	100	100
c6288	0	0	0	3.5	5.1	3.3	100	100	100
c7552	0	0	0	3.9	7.8	12	100	100	100

remained unaffected.

In what follows we compare Safari to two DXC'09 [Feldman et al., 2010] algorithms: NGDE [de Kleer, 2009] and RODON [Karin et al., 2006]. This comparison has been performed in the DXC'09 experimental framework. For more information on the conditions under which we have evaluated the performance of NGDE, RODON, and Safari, cf. [Feldman et al., 2010].

NGDE is an Allegro Common Lisp implementation of the classic GDE. NGDE uses a minimum-cardinality candidate generator to construct diagnoses from conflicts. For ADAPT-Lite it uses interval constraints.

RODON [Karin et al., 2006] is based on the principles of GDE as described by de Kleer and Williams [1987] and the G$^+$DE [Heller and Struss, 2001]. RODON uses contradictions (conflicts) between the simulated and the observed behavior to generate hypotheses about possible causes for the observed behavior. If the model contains failure modes in addition to the nominal behavior, these can be used to verify the hypotheses, which speeds up the diagnostic process and improve the results.

Table 3.5 shows the $\bar{M}_{\text{utl}}$, $\bar{M}_{\text{cpu}}$, and $\bar{M}_{\text{mem}}$ metrics (cf. Sec. 2.5). It can be seen that Safari has achieved significantly better $\bar{M}_{\text{cpu}}$ and $\bar{M}_{\text{mem}}$ than NGDE [de Kleer, 2009] and RODON [Karin et al., 2006]. M_{utl} of Safari is slightly worse due to smaller number of diagnostic candidates computed by this DA. Safari and RODON showed similar results in the utility metrics.

Table 3.6 shows the $\bar{M}_{\text{sru}}$, $\bar{M}_{\text{dru}}$, and $\bar{M}_{\text{err}}$ metrics. We can see that $\bar{M}_{\text{dru}}$ does

Table 3.5: ISCAS85/74XXX *metrics results*

	SAFARI			NGDE			RODON		
Name	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
74182	0.365	62	17	0.466	230	10 716	0.262	1 293	18 205
74L85	0.455	53	18	0.575	341	11 838	0.372	5 233	22 533
74283	0.419	57	17	0.479	206	10 654	0.353	4 863	20 714
74181	0.374	73	21	0.486	213	10 879	0.405	14 222	26 962
c432	0.529	91	24	0.664	319	12 058	0.492	19 129	36 772
c499	0.29	80	33	0.414	1 719	17 063	0.258	20 649	36 436
c880	0.262	1 842	37	0.296	1 516	21 437	0.275	18 404	34 843
c1355	0.335	387	34	0.37	4 734	23 967	0.373	22 133	33 653
c1908	0.208	745	29	0.232	8 994	33 995	0.19	24 361	36 102
c2670	0.603	327	119	0.921	571	14 828	0.886	17 178	34 069
c3540	0.355	833	33	0.374	9 223	31 954	0.307	49 397	48 162
c5315	0.243	811	94	0.531	6 477	22 406	0.238	87 720	50 526
c6288	0.316	2 162	32	0.32	11 784	65 086	0.316	89 130	51 268
c7552	0.3	2 001	97	0.436	8 638	39 592	0.364	172 558	65 846
Averaged	0.361	680	43	0.469	3 926	23 320	0.364	39 019	36 864

Table 3.6: ISCAS85/74XXX *secondary metrics results*

	SAFARI			NGDE			RODON		
	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{sru}}$	$\bar{M}_{\text{dru}}$	$\bar{M}_{\text{err}}$
74182	0.381	0.984	69	0.574	0.892	78	0.262	1	80
74L85	0.458	0.996	30	0.617	0.958	39	0.46	0.913	78
74283	0.437	0.982	46	0.523	0.957	51	0.423	0.93	82
74181	0.378	0.995	48	0.517	0.969	55	0.456	0.949	87
c432	0.53	0.999	29	0.671	0.993	35	0.505	0.987	64
c499	0.293	0.997	71	0.428	0.986	78	0.268	0.99	107
c880	0.263	0.999	89	0.306	0.99	127	0.281	0.994	113
c1355	0.336	0.999	73	0.375	0.995	94	0.375	0.999	71
c1908	0.208	0.999	69	0.239	0.993	113	0.191	1	70
c2670	0.603	1	24	0.921	1	6	0.886	1	10
c3540	0.355	1	58	0.376	0.999	88	0.308	0.999	82
c5315	0.243	1	73	0.532	0.999	58	0.239	0.999	114
c6288	0.317	1	16	0.32	1	15	0.317	0.999	18
c7552	0.3	1	60	0.437	0.999	69	0.364	0.999	70
Averaged	0.364	0.996	54.02	0.488	0.981	64.75	0.381	0.983	74.71

not contribute significantly to $\bar{M}_{\text{utl}}$. The reason for that is discussed in Sec. 2.5.

We have computed $\bar{M}_{\text{sat}}$ and the results are shown in Table 3.7. The SAT and UNSAT columns show the number of consistent and inconsistent candidates, respectively. NGDE has generated approximately two orders of magnitude more satisfiable candidates than SAFARI and RODON. The policy of SAFARI has been to compute a small number of candidates, minimizing $\bar{M}_{\text{mem}}$ and $\bar{M}_{\text{cpu}}$. In order to improve $\bar{M}_{\text{utl}}$, SAFARI has mapped multiple-cardinality candidates into single-component failure probabilities. Hence, only single-fault scenarios contribute to the $\bar{M}_{\text{sat}}$ score for SAFARI.

Table 3.7: *Satisfiability results*

	SAFARI			NGDE			RODON		
Name	SAT	UNSAT	$\bar{M}_{\text{sat}}$	SAT	UNSAT	$\bar{M}_{\text{sat}}$	SAT	UNSAT	$\bar{M}_{\text{sat}}$
74182	19	45	5.67	1240	0	28	0	0	0
74L85	1	27	1	178	0	20	13	7	13
74283	34	57	5.32	561	0	20	15	5	15
74181	12	43	2.4	691	0	20	4	16	4
c432	10	29	4.7	1 109	0	20	5	15	5
c499	2	118	1	707	0	20	2	12	2
c880	27	86	1.74	12 663	0	20	15	0	15
c1355	36	162	4.1	3 246	0	20	8	3	8
c1908	13	35	1	3 593	4	7	0	2	0
c2670	7	30	7	25	0	19	17	2	17
c3540	38	77	1.86	231	10	10	1	17	1
c5315	0	55	0	1 665	0	20	0	13	0
c6288	8	30	0.27	126	0	2	2	2	2
c7552	7	53	0.64	1 493	3	17	1	17	1
Averaged	15.29	60.50	2.62	1 966.29	1.21	17.36	5.93	7.93	5.93

3.7.3 Comparison to Algorithms Based on Max-SAT

Max-SAT is an extensively studied optimization problem and variations of Max-SAT (like Partial Max-SAT and Weighted Max-SAT) can be used for computing minimal diagnoses. In what follows we (1) describe a family of algorithms for computing minimal diagnoses based on Max-SAT and (2) experimentally evaluate the performance and optimality of these algorithms on a benchmark consisting of 74XXX/ISCAS85 combinational circuits. For the experiments we use both complete (partial and weighted) and SLS-based Max-SAT. We have established

that the performance of complete Max-SAT and the optimality of SLS-based Max-SAT degrade when increasing the circuit size or the cardinality of the faults.

Given a formula Φ in CNF, a Max-SAT [Hoos and Stützle, 2004] solution is a variable assignment that maximizes the number of satisfied clauses in Φ (in most cases of interest Φ is unsatisfiable, otherwise any variable assignment which satisfies Φ is also a Max-SAT solution). In partial Max-SAT some of the clauses in Φ are designated as hard, the others are "soft". A solution to the partial Max-SAT problem should satisfy all "hard" clauses and maximize the number of satisfied "soft" clauses. Similarly, in weighted Max-SAT a weight is assigned to each clause in Φ and a solution maximizes the sum of the weights of the satisfied clauses.

In this section we (1) show how to use Max-SAT for computing minimal diagnoses, (2) study the performance of a family of Max-SAT-based diagnostic algorithms, and (3) empirically compare the performance and optimality of the Max-SAT-based algorithms to SAFARI. The results show that, although diagnosis based on Max-SAT performs better than other algorithms for computing minimal diagnosis, specialized stochastic solvers like SAFARI outperform Max-SAT by at least an order-of-magnitude for the class of diagnostic problems we have considered. This is not surprising, as specialized MBD algorithms exploit specific properties of the search space, as we will describe.

A Max-SAT-Based MBD Algorithm

We have used the approach of Sang et al. [2007] for encoding Most Probable Explanation (MPE) as Max-SAT. Computing MPE is identical to computing a most-probable diagnosis in a more general framework. Algorithm 4 computes diagnoses by calling a Max-SAT oracle.

Note that the diagnostic problems we solve in this chapter can be translated to multiple optimization problems which can be solved with SAT-based methods [Giunchiglia and Maratea, 2006]. The Maximum Satisfiable Subset (MSS) problem, for example, is dual to the Minimal Unsatisfiable Subset problem [Bailey and Stuckey, 2005] and the two can be solved with Max-SAT and Min-UNSAT solvers, respectively [Liffiton and Sakallah, 2005]. From those, we have found preference in the research community towards Max-SAT, and for practical reasons we therefore compare SAFARI to Max-SAT.

Algorithm 4 adds a unit clause with weight 1 for each assumable (line 6). The weight of each input clause is set to a value greater than the number of all assumables (line 3). The loop in lines 8 – 11 computes a diagnosis with a call to Max-SAT (the auxiliary subroutine HEALTHLITERALS computes a diagnosis from a Max-SAT assignment by filtering the health literals) and if a diagnosis exists, it is added to the result (line 10). Finally, the negation of each diagnosis is added to the original set of clauses (line 9) to prevent subsequent computation of the same diagnosis. Note that the negation of a term is conveniently a clause.

Depending on the implementation of the MAX-SAT call in line 8 of Alg. 4 we have a family of MAX-SAT algorithms for diagnosis: (1) if MAX-SAT is a

Algorithm 4 Weighted Max-SAT based diagnosis algorithm.

```
1: function MaxSATDiagnose(DS, α) returns a set of diagnoses
       inputs: DS = ⟨SD, COMPS, OBS⟩, diagnostic system
               α, term, observation
       local variables: W, set of clauses; Ω, set of diagnoses
                        ω, diagnosis term; c, clause; h, variable
2:     for all c ∈ Clauses(SD) do
3:         W ← W ∪ ⟨∞, c⟩
4:     end for
5:     for all h ∈ COMPS do
6:         W ← W ∪ ⟨1, c⟩
7:     end for
8:     while ω ← HealthLiterals(Max-SAT(W)) do
9:         W ← W ∪ ⟨∞, ¬ω⟩
10:        Ω ← Ω ∪ ω
11:    end while
12:    return Ω
13: end function
```

partial Max-SAT solver, Alg. 4 computes diagnoses ordered by cardinality; (2) if Max-SAT is a weighted Max-SAT solver, Alg. 4 computes diagnoses ordered by probability; and (3) if Max-SAT is based on SLS, not every iteration of the main loop yields a diagnosis. We have run extensive experiments with all three Max-SAT variants, which we describe in the following sub-sections.

Experimental Results with Complete Max-SAT

We next show the performance of Alg. 4 with two partial Max-SAT algorithms (partial and weighted Max-SAT algorithms are always complete).
Table 3.8 shows performance results of the partial Max-SAT solver W-MaxSatz [Argelich et al., 2007] and the weighted MiniMaxSat [Heras et al., 2008]. Note that the performance of W-MaxSatz in weighted mode is slightly worse than when solving the respective partial formulae. Both solvers are state-of-the-art and have been submitted to the Second Max-SAT Evaluation 2007.[8] The performance of the complete Max-SAT solvers degrades with increasing circuit size and the cardinality of the injected faults. The results of those algorithms are comparable to, or slightly better, than the performance of CDA* and HA*. For comparison, Safari solves all test instances.

Our findings are consistent with the ones of Giunchiglia and Maratea [2006]. In their paper, Giunchiglia and Maratea [2006] report ISCAS85 times similar to the ones measured with state-of-the-art Max-SAT solvers where no solver com-

[8] `http://www.maxsat07.udl.es/`

Table 3.8: *Performance of* W-MaxSatz *and* MiniMaxSat *[% of tests solved]*

	W-MaxSatz			MiniMaxSat		
Name	Weak	S-A-0	S-A-1	Weak	S-A-0	S-A-1
74182	100	100	100	100	100	100
74L85	100	100	100	100	100	100
74283	100	100	100	100	100	100
74181	100	100	100	100	100	100
c432	99.7	100	100	100	100	100
c499	2	100	65.5	99.9	43.8	100
c880	0	9.2	34.9	95.5	100	99.7
c1355	0	0	0	62	17.6	50.1
c1908	0	0	0	36.5	0.5	0
c2670	0	3.7	0	74.1	0	0
c3540	0	0	0	0.1	0	0
c5315	0	0	0	0.2	0	0
c6288	0	0	0	0	0	0
c7552	0	0	0	0	0	0

putes a solution for a circuit larger than c3540. These experiments consider the plain ISCAS85 combinational circuits (i.e., their models do not have assumable variables) and it has been discussed by the same authors that adding assumable variables (or in their terminology increasing the number of preferences) increases the problem difficulty.

Experimental Results with SLS Max-SAT

A diagnostic algorithm based on SLS Max-SAT would be the best candidate for comparison to Safari due to the stochastic nature of the latter. Unfortunately, the following issues complicate the use of SLS Max-SAT in diagnostic algorithms:

- There is no simple termination criterion in diagnostic algorithms based on SLS Max-SAT, i.e., we keep the local diagnosis and restart Safari after a number of successive "unsuccessful" flips, while there is no notion of "unsuccessful" flip (from the viewpoint of diagnosis) in Max-SAT. As we will see from our experimentation, flipping a variable which decreases the weight (or number) of currently satisfied clauses may be necessary to escape plateaus and/or local optima, hence the accumulation of such flips cannot be used as a termination criterion;

- Diagnostic Max-SAT problems have two type of constraints: hard and soft. The hard constraints are the clauses of the original ("nominal") model, while

the soft constraints are the unit clauses received from the assumable variables. An SLS Max-SAT algorithm does not distinguish between those hard and soft clauses; if such an algorithm guaranteed the satisfaction of the hard-constraints it would be classified as hybrid and not stochastic.

The two reasons above prevent us from using a diagnostic reasoner based on SLS Max-SAT in practical applications. Despite that, we have conducted extensive experimentation with UBCSAT [Tompkins and Hoos, 2005] in order to evaluate the potential of SLS Max-SAT in MBD.

To overcome the termination problems with SLS Max-SAT, for the following experiments, we have chosen observations leading to known single faults. For each **WFM** we have chosen 50 observations. We have configured the SLS Max-SAT search to terminate after 100 000 variable flips and we have modified Alg. 4 to terminate after 10 calls to Max-SAT. The resulting optimality of algorithms based on SLS Max-SAT in computing single fault diagnoses is shown in Table 3.9. We have run experiments with all algorithms or algorithm variants implemented by the UBCSAT suite, covering the algorithms RGSAT, Schöening, CDRW, through GSAT.

The data in Table 3.9 show best cases. From each of the 500 Max-SAT invocations per algorithm/circuit (50 single faults, 10 runs per experiment) we have (1) ignored all results which do not satisfy all hard constraints, (2) recorded the best diagnostic cardinality achieved in the hill climbing (recall that these are single-faults hence the best result is 1) and (3) recorded the number of steps (bit flips) in which this best diagnostic cardinality was achieved (the number of bit-flips are given in parentheses below the optimality number in Table 3.9).

Table 3.9 shows the generally poor performance of SLS Max-SAT algorithms. In most of the cases the algorithm could either never satisfy all hard-constraints or achieved increasingly worse cardinality with the growth of the circuit. Exceptions are the two variants of SAPS [Hutter et al., 2002] and we attribute this relatively good optimality of SAPS to its mechanism for assigning and updating weights to clauses based on the clause length. Recall that in our diagnostic problems clauses of assumable literals have unit weights while hard-constraints have weights greater than the number of assumable literals. Despite that, in the best case for c7552, SAPS needed 77 264 bit flips to find the optimal single-fault diagnosis. In comparison SAFARI performed 11 bit flips, and although an LTMS/SAT consistency check of SAFARI is strictly more expensive than the consistency checking of SLS Max-SAT (the former is worst-case NP-hard while the latter is in P), SAFARI is computationally more efficient on average.

Figure 3.12 illustrates the progress of two SLS Max-SAT invocations. The Conflict-Directed Random Walk (CDRW) [Papadimitriou, 1991] starts with a random variable assignment and flips the most profitable (for increasing the satisfied weight) variable. This often leads to violated hard-constraints (due to flipping of non-assumable variables), and the restarts which are needed for escaping those situations lead to the relatively noisy ascent of CDRW. Other SLS Max-SAT al-

Table 3.9: *Diagnostic cardinality and number of bit-flips (in parentheses) of SLS-based Max-SAT algorithms and* SAFARI *(***WFM***)*

Name	RGSAT	Schöening	CDRW	URW	IRoTS	RoTS	G2WSAT	SAPS[a]	SAPS[b]	Adaptive Novelty$^+$	Novelty$^+$	Novelty	WalkSAT/TABU	WalkSAT	HSAT	GWSAT	GSAT	SAFARI
74182	1	1	1	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	(4331)	(163)	(501)	(29896)	(1265)	(19)	(40)	(40)	(76)	(42260)	(28)	(56)	(21)	(37)	(26)	(186)	(38)	(2)
74L85	1	1	1	–	1	3	1	1	1	1	1	1	5	1	1	1	1	1
	(65623)	(276)	(353)	(–)	(7236)	(27)	(39)	(199)	(143)	(80782)	(9815)	(1267)	(47)	(116)	(34)	(629)	(34)	(4)
74283	1	1	1	–	1	1	1	1	1	1	1	1	4	1	1	1	1	1
	(63429)	(1134)	(573)	(–)	(3644)	(26)	(21806)	(166)	(112)	(25003)	(31900)	(435)	(35)	(1034)	(29)	(187)	(32)	(2)
74181	4	1	1	–	1	13	1	1	1	1	1	1	10	1	3	1	3	1
	(48395)	(4525)	(6542)	(–)	(49967)	(1498)	(1814)	(146)	(453)	(1045)	(81883)	(2060)	(75)	(2123)	(67)	(1139)	(54)	(3)
c432	28	1	1	–	4	36	4	1	1	7	1	8	–	1	13	1	16	1
	(43176)	(1866)	(7452)	(–)	(82496)	(101)	(10262)	(806)	(564)	(20211)	(75488)	(16393)	(–)	(905)	(133)	(24164)	(167)	(2)
c499	–	1	1	–	29	–	6	1	1	7	5	13	–	1	–	1	–	1
	(–)	(24203)	(95242)	(–)	(99999)	(–)	(98683)	(42534)	(66519)	(2112)	(87589)	(49516)	(–)	(47102)	(–)	(45465)	(–)	(1)
c880	–	1	1	–	–	–	26	1	1	35	23	30	–	1	–	1	46	1
	(–)	(36269)	(8388)	(–)	(–)	(–)	(9561)	(4244)	(1551)	(5680)	(80926)	(34989)	(–)	(9825)	(–)	(27026)	(304)	(5)
c1355	–	2	1	–	–	–	44	1	1	52	51	53	–	1	–	1	–	1
	(–)	(93549)	(34235)	(–)	(–)	(–)	(83944)	(9670)	(4975)	(84777)	(67454)	(85365)	(–)	(34786)	(–)	(93656)	(–)	(5)
c1908	240	–	–	–	–	–	91	1	1	94	98	101	–	1	125	1	139	1
	(68649)	(–)	(–)	(–)	(–)	(–)	(97567)	(95942)	(19664)	(5142)	(79442)	(43541)	(–)	(50660)	(654)	(94695)	(643)	(6)
c2670	–	–	–	–	–	–	122	1	1	126	135	144	–	1	173	1	177	1
	(–)	(–)	(–)	(–)	(–)	(–)	(93799)	(9547)	(6635)	(19000)	(67042)	(6532)	(–)	(83292)	(1008)	(85581)	(976)	(5)
c3540	–	–	–	–	–	–	167	1	1	170	177	203	–	1	–	16	–	1
	(–)	(–)	(–)	(–)	(–)	(–)	(79188)	(16376)	(17776)	(88052)	(68342)	(68663)	(–)	(99209)	(–)	(93639)	(–)	(9)
c5315	–	–	–	–	–	–	261	1	1	270	275	329	–	14	–	70	–	1
	(–)	(–)	(–)	(–)	(–)	(–)	(78093)	(79438)	(48388)	(64079)	(96973)	(48497)	(–)	(98557)	(–)	(89462)	(–)	(9)
c6288	–	–	–	–	–	–	329	4	8	360	372	–	–	58	–	–	–	1
	(–)	(–)	(–)	(–)	(–)	(–)	(23191)	(79807)	(73752)	(36373)	(91636)	(–)	(–)	(94673)	(–)	(–)	(–)	(3)
c7552	–	–	–	–	–	–	403	1	1	412	440	449	–	115	492	181	507	1
	(–)	(–)	(–)	(–)	(–)	(–)	(12905)	(77264)	(42284)	(39046)	(90485)	(32732)	(–)	(90449)	(2543)	(99340)	(2552)	(11)

[a] Clause penalties are initialized to the clause weights and smoothed back to their initial values.
[b] Clause penalties are initialized to the clause weights.

gorithms like HSAT [Gent and Walsh, 1993] avoid downward flips (flips which decrease the currently satisfied weight), quickly increasing the satisfied weight but ultimately get stuck in local optima. A close inspection of Fig. 3.12 reveals that HSAT oscillates forever short of satisfying all hard constraints.

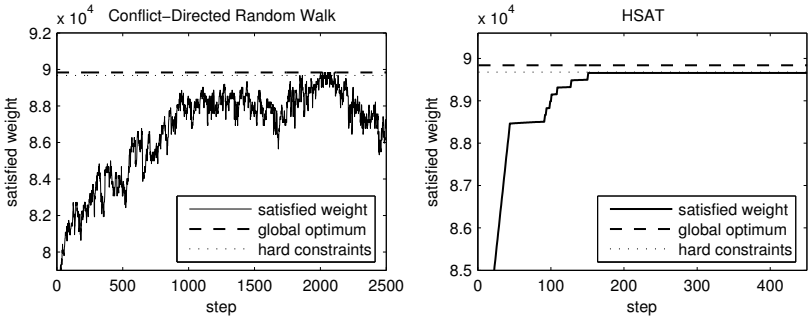


Figure 3.12: *Progress of two SLS Max-SAT algorithms in a weak-fault model of c432, single fault observation*

We have shown a family of Max-SAT-based algorithms for computing minimal diagnoses. Depending on the type of the Max-SAT oracle, these algorithms compute minimal-cardinality diagnoses (by using partial Max-SAT) or probability-minimal diagnoses (by using weighted Max-SAT). We have also discussed the implications of using an SLS-based Max-SAT algorithm and studied the optimality of the computed diagnoses with a number of SLS-based Max-SAT algorithms.

We have experimented with the 74XXX/ISCAS85 combinational circuits and a number of observation vectors. The results showed that although Max-SAT could compute diagnoses in many of the cases, the performance of Max-SAT degraded when increasing the circuit size or the cardinality of the injected faults.

3.7.4 Comparison to Algorithms Based on ALLSAT and Model Counting

We have compared the performance of SAFARI to that of a pure SAT-based approach, which uses blocking clauses for avoiding duplicate diagnoses [Jin et al., 2005]. Although SAT encodings have worked efficiently on a variety of other domains, such as planning, the weak health modeling makes the diagnostic problem so underconstrained that an uninformed ALLSAT strategy (i.e., a search not exploiting the continuity imposed by the weak-fault modeling) is quite inefficient, even for small models.

To substantiate our claim, we have experimented with the state-of-the-art satisfiability solver RELSAT, version 2.02 [Bayardo and Pehoushek, 2000]. Instead of enumerating all solutions and filtering the minimal diagnoses only, we have performed model-counting, whose relation to MBD has been extensively studied [Kumar, 2002]. While it was possible to solve the two smallest circuits, the solver

did not terminate for any of the larger models within the predetermined time of 1 h. The results are shown in Table 3.10.

Table 3.10: *Model count and time for counting*

	RelSat		SampleCount	
Name	Models	Time [s]	Models	Time [s]
74182	3.9896×10^{7}	1	$\geq 3.526359 \times 10^{6}$	0.2
74L85	8.3861×10^{14}	340	$\geq 7.412344 \times 10^{13}$	0.3
74283	$\geq 1.0326 \times 10^{15}$	$> 3\,600$	$\geq 3.050026 \times 10^{14}$	0.3
74181	$\geq 5.6283 \times 10^{15}$	$> 3\,600$	$\geq 1.538589 \times 10^{27}$	1.1
c432	$\geq 7.2045 \times 10^{18}$	$> 3\,600$	$\geq 1.496602 \times 10^{67}$	9.9
c499	$\geq 3.6731 \times 10^{20}$	$> 3\,600$	$\geq 7.549183 \times 10^{83}$	13.1
c880	$\geq 9.4737 \times 10^{39}$	$> 3\,600$	$\geq 8.332702 \times 10^{166}$	42.7
c1355	$\geq 1.4668 \times 10^{28}$	$> 3\,600$	$\geq 7.488300 \times 10^{233}$	99.8
c1908	$\geq 2.1704 \times 10^{31}$	$> 3\,600$	–	–
c2670	$\geq 9.0845 \times 10^{15}$	$> 3\,600$	–	–
c3540	$\geq 4.8611 \times 10^{19}$	$> 3\,600$	–	–
c5315	$\geq 9.3551 \times 10^{16}$	$> 3\,600$	–	–
c6288	$\geq 1.0300 \times 10^{18}$	$> 3\,600$	–	–
c7552	$\geq 1.0049 \times 10^{16}$	$> 3\,600$	–	–

The second column of Table 3.10 shows the model count returned by RelSat, with sample single-fault observations from our benchmark. The third column reports the time for model counting. This slow performance on relatively small diagnostic instances leads us to the conclusion that specialized solvers like Safari are better suited for finding minimal diagnoses than off-the-shelf ALLSAT (model counting) implementations that do not encode inference properties similar to those encoded in Safari.

We have used the state-of-the-art, non-exact model counting method SampleCount [Gomes et al., 2007] to compute lower bounds of the model counts. The results are shown in the third and fourth columns of Table 3.10. Configured with the default settings ($\alpha = 3.5$, $t = 2$, $z = 20$, cutoff 10 000), SampleCount could not find lower bounds for circuits larger than c1355. Although the performance of SampleCount is significantly better than RelSAT, the fact that SampleCount computes lower bounds and does not scale to large circuits prevent us from building a diagnosis algorithm based on approximate model counting.

A satisfiability-based method for diagnosing an optimized version of ISCAS85 has been used by [Smith et al., 2004]. In a recent paper [Smith et al., 2005], the SAT-based approach has been replaced by a Quantified Boolean Formula (QBF) solver for computing multiple-fault diagnoses. These methods report good absolute execution time for single and double-faults (and we believe that they scale

well for higher cardinalities), but require modifications of the initial circuits (i.e., introduce cardinality and test constraints) and suggest specialized heuristics for the SAT solvers in order to improve the search performance. Comparison of the performance of SAFARI to the timings reported by these papers would be difficult due to a number of reasons like the use of different and optimized benchmark sets, trading-off memory for speed, rewriting the original circuits, etc.

3.7.5 Performance of the Greedy Stochastic Search

Table 3.11 shows the absolute performance of SAFARI ($M = |\text{COMPS}|, N = 4$). This varies from under a millisecond for the small models, to approx. 30 s for the largest strong-fault model. These fast absolute times show that SAFARI is suitable for on-line reasoning tasks, where autonomy depends on speedy computation of diagnosis.

Table 3.11: *Performance of* SAFARI *[ms]*

	Weak		S-A-0		S-A-1	
Name	$t_{\min}$	$t_{\max}$	$t_{\min}$	$t_{\max}$	$t_{\min}$	$t_{\max}$
74182	0.41	1.25	0.39	4.41	0.40	0.98
74L85	0.78	7.47	0.72	1.89	0.69	4.77
74283	0.92	4.84	0.88	3.65	0.92	5.2
74181	2.04	6.94	2.13	22.4	2.07	7.19
c432	8.65	38.94	7.58	30.59	7.96	38.27
c499	14.19	31.78	11.03	30.32	10.79	31.11
c880	48.08	88.87	37.08	80.74	38.47	81.34
c1355	95.03	141.59	76.57	150.29	83.14	135.29
c1908	237.77	349.96	196.13	300.11	217.32	442.91
c2670	500.54	801.12	646.95	1 776.72	463.24	931.8
c3540	984.31	1 300.98	1 248.5	2 516.46	976.56	2 565.18
c5315	1 950.12	2 635.71	3 346.49	7 845.41	2 034.5	4 671.17
c6288	2 105.28	2 688.34	2 246.84	3 554.4	1 799.18	2 469.48
c7552	4 557.4	6 545.21	9 975.04	32 210.71	5 338.97	12 101.61

For each model, the minimum and maximum time for computing a diagnosis has been computed. These values are shown under columns t_{min} and t_{max}, respectively. The small range of $t_{max} - t_{min}$ confirms our theoretical results that SAFARI is insensitive to the fault cardinalities of the diagnoses it computes. The performance of CDA* and HA*, on the other hand, is dependent on the fault cardinality and quickly degrades.

3.7.6 Optimality of the Greedy Stochastic Search

From the result produced by the complete diagnostic methods (CDA* and HA*) we know the exact cardinalities of the minimal-cardinality diagnoses for some of the observations. By considering these observations which lead to single and double faults we have evaluated the average optimality of SAFARI. Table 3.12 shows these optimality results for the greedy search. The second column of Table 3.12 shows the number of observation vectors leading to single faults for each weak-fault model. The third column shows the average cardinality of SAFARI. The second and third column are repeated for the S-A-0 and S-A-1 models, and then, all the six columns are repeated for double faults.

Table 3.12: *Optimality of* SAFARI *[average cardinality]*

	Single Faults						Double Faults					
	Weak		S-A-0		S-A-1		Weak		S-A-0		S-A-1	
Name	#	Card.	#	Card.	#	Card.	#	Card.	#	Card.	#	Card.
74182	50	1	37	1	40	1	50	2	38	2	18	2
74L85	50	1.04	18	1.02	40	1.03	50	2.12	17	2.06	35	2.07
74283	50	1.08	34	1.59	46	1.88	50	2.2	45	2.41	42	2.6
74181	50	1.19	36	2.81	46	2.6	50	2.25	36	3.61	43	3.16
c432	58	1.19	52	1.06	37	1.04	82	2.46	80	2.25	48	2.15
c499	84	1.49	53	1.49	84	1.01	115	3.27	34	3.01	115	2.03
c880	50	1	39	1.1	40	1.05	50	2.01	34	2.14	35	2.07
c1355	84	1.66	82	1	84	1.02	6	2.15	7	2	18	2.07
c1908	52	1.05	49	2.91	52	4.79	–	–	2	3	3	3.17
c2670	29	1.03	39	1.77	28	2.06	13	2.12	24	2.78	15	3.27
c3540	8	1.01	23	2.5	16	3.74	–	–	1	4.9	–	–
c5315	14	1	9	3.54	12	5.4	7	2	3	3.7	1	3.8
c6288	13	1	13	28.83	12	28.68	1	2	1	27	–	–
c7552	27	1.01	11	17.37	18	23.38	16	2	4	18.5	6	27.53

Table 3.12 shows that, for weak fault models, the average cardinality returned by SAFARI is very close to the optimal values for both single and double faults. The c1355 model shows the worst-case results for the single-fault observations, while c499 is the most-difficult weak-fault model for computing a double-fault diagnosis. These results can be easily improved by increasing M and N as discussed in Sec. 3.5.

With strong-fault models results are close to optimal for the small models and the quality of diagnosis deteriorates for c3540 and bigger. This is not surprising considering the modest number of retries and number of "flips" with which SAFARI was configured.

3.7.7 Computing Multiple Minimal-Cardinality Diagnoses

We next show the results of experiments supporting the claims made in Sec. 3.6. For that, we have first chosen these observations α for which we could compute $|\Omega^{\leq}(\mathrm{SD} \wedge \alpha)|$ with a deterministic algorithm like CDA* or HA* (mostly observations leading to single or double faults). We have then configured SAFARI with $M = |\mathrm{COMPS}|$ and $N = 10|\Omega^{\leq}(\mathrm{SD} \wedge \alpha)|$. Finally, from the diagnoses computed by SAFARI we have filtered the minimal-cardinality ones. The results are summarized in Table 3.13.

Table 3.13: *% of all minimal-cardinality diagnoses computed by* SAFARI

	Weak			S-A-0			S-A-1		
Name	$\lvert\Omega^{\leq}\rvert$	M_c	M_f	$\lvert\Omega^{\leq}\rvert$	M_c	M_f	$\lvert\Omega^{\leq}\rvert$	M_c	M_f
74182	$1-25$	100	0	$1-2$	100	0	$1-20$	100	0
74L85	$1-78$	99.2	2	$1-4$	100	0	$1-49$	99.7	0
74283	$1-48$	97.9	3	$1-16$	93.8	0	$1-29$	84.9	4
74181	$1-133$	97.4	1	$1-16$	88.6	4.07	$1-57$	96.7	6.36
c432	$1-99$	94.2	7.14	$1-40$	89.7	0	$1-18$	97	0
c499	$1-22$	78.5	1.51	$1-15$	96.3	6	$1-16$	94.8	0
c880	$2-646$	99.9	0	$1-160$	96.9	0	$1-210$	97.5	0
c1355	$5-2\,770$	79.4	1.02	$2-648$	95.7	0	$2-347$	95.2	0.52
c1908	$2-1\,447$	96.6	2.61	$2-579$	85.2	1.85	$2-374$	82.3	1.24
c2670	$1-76$	100	2.34	$1-20$	97.1	0	$1-181$	89.7	0
c3540	$1-384$	81.5	8.52	$1-153$	88.8	7.98	$1-171$	78.2	7.27
c5315	$1-235$	97.7	1.74	$1-24$	81.7	7.04	$1-30$	93.4	8.24
c6288	$1-154$	100	13.1	$1-73$	78.1	5.1	$1-101$	82.1	1.22
c7552	$1-490$	93.1	2.17	$4-236$	90.8	13.55	$1-168$	78	12.1

The data in Table 3.13 are to be interpreted as follows. The columns marked with $|\Omega^{\leq}|$ show the minimal and maximal number of minimal-cardinality diagnoses per model as computed by a deterministic algorithm. The columns M_c show the percentage of minimal-cardinality diagnoses returned by SAFARI (from all minimal-cardinality diagnoses) for those α for which $|\Omega^{\leq}(\mathrm{SD} \wedge \alpha)| > 1$. The columns M_f show the percentage of observations for which SAFARI could not compute any minimal-cardinality diagnosis.

The results shown in Table 3.13 show that even for moderate values of N ($N \leq 27\,770$), SAFARI was capable of computing a significant portion of all minimal-cardinality diagnoses. This portion varies from 78.5% to 100% for weak-fault models and from 78% to 100% for strong-fault models. The percentage of cases in which SAFARI could not reach a minimal-cardinality diagnosis is limited (smaller than 13.55%) and is mainly in the cases in which there exists only one single-fault diagnosis. Note that even in the cases in which SAFARI cannot compute any

minimal-cardinality diagnoses, the result of SAFARI can be useful. For example, a subset-minimal diagnosis of small cardinality differing in one or two literals only, still brings useful diagnostic information (a discussion on diagnostic metrics is beyond the scope of this chapter).

3.7.8 Experimentation Summary

We have applied SAFARI to a suite of benchmark combinatorial circuits encoded using weak fault models and stuck-at strong fault models, and shown significant performance improvements for multiple-fault diagnoses, compared to two state-of-the-art deterministic algorithms, CDA* and HA*. Our results indicate that SAFARI shows at least an order-of-magnitude speedup over CDA* and HA* for multiple-fault diagnoses. Moreover, whereas the search complexity for the deterministic algorithms tested increases exponentially with fault cardinality, the search complexity for this stochastic algorithm appears to be independent of fault cardinality.

We have compared the performance of SAFARI to that of an algorithm based on Max-SAT and SAFARI shows at least an order-of-magnitude speedup in computing diagnoses. We have compared the optimality of SAFARI to that of an algorithm based on SLS Max-SAT and SAFARI consistently computes diagnoses of smaller cardinality whereas the SLS Max-SAT diagnostic algorithm often fails to compute any diagnoses.

3.8 Summary

We have described a greedy stochastic algorithm for computing diagnoses within a model-based diagnosis framework. We have shown that subset-minimal diagnoses can be computed optimally in weak fault models and in an important subset of strong fault models, and that almost all minimal-cardinality diagnoses can be computed for more general fault models.

We argue that SAFARI can be of broad practical significance, as it can compute a significant fraction of minimal-cardinality diagnoses for systems too large or complex to be diagnosed by existing deterministic algorithms.

In future work, we plan to experiment on models with a combination of weak and strong failure-mode descriptions. We also plan on experimenting with a wider variety of stochastic methods, such as simulated annealing and genetic search, using a larger set of benchmark models. Last, we plan to apply our algorithms to a wider class of abduction and constraint optimization problems.

Chapter 4

Computing Worst-Case Diagnostic Scenarios

Model-Based Diagnosis (MBD) typically focuses on diagnoses, minimal under some minimality criterion, e.g., the minimal-cardinality set of faulty components that explain an observation α. However, for different α there may be minimal-cardinality diagnoses of differing cardinalities, and several applications (such as test pattern generation and benchmark model analysis) need to identify the α leading to the max-cardinality diagnosis amongst them. We denote this problem as a Max-Fault Min-Cardinality (MFMC) problem. This chapter considers the generation of observations that lead to MFMC diagnoses. We present a near-optimal, stochastic algorithm, called MIRANDA (Max-fault mIn-caRdinAlity observatioN Deduction Algorithm), that computes MFMC observations. Compared to optimal, deterministic approaches such as ATPG, the algorithm has very low cost, allowing us to generate observations corresponding to high-cardinality faults.

4.1 Introduction

The problem of computing minimal-cardinality diagnoses, given an observation and a system description, is central to Model-Based Diagnosis [de Kleer and Williams, 1987]. In this chapter we consider the "inverse" problem of computing an observation that simultaneously isolates k faulty components. These observations are useful in system testing and benchmarking of multiple-fault diagnostic techniques. Computing observations (in particular inputs) that distinguish a single failing component ($k = 1$) is studied by Automatic Test Pattern Generation (ATPG) and dates back to the D-algorithm [Roth, 1966]. The goal of ATPG is to compute a *sequence* of test vectors that can detect every possible single fault in a device. Single-fault ATPG has been extended to finding observation vec-

tors leading to double faults [Hughes, 1988] and to multiple faults [Kubiak and Fuchs, 1991]. These approaches have several drawbacks, including: (1) they do not determine the maximum possible value of k (2) they suffer from very high computational complexity, and (3) they severely limit the class of system abstractions by imposing various model restrictions.

Few papers have proposed algorithms computing observation vectors that distinguish the *maximum* number of failing components in a system [Abramovici, 1981]. The GuidedProbe algorithm in the latter paper relies on probing to achieve the maximal fault resolution for a fixed test T. The author of this algorithm has a different goal, i.e., achieving maximal resolution by minimizing the number of probes, and proposes essentially a sequential algorithm. This is very different from MBD approaches, which try to solve the multiple-fault problem with only one observation.

To the best of our knowledge, we are the first to formally state the problem and significance of finding MFMC observations, and then to define an algorithm that is able to approximate such a computationally difficult problem. Our method is based on a greedy stochastic search algorithm, called Miranda (Max-fault mIn-caRdinAlity observatioN Deduction Algorithm), and uses an MBD oracle for computing minimal-cardinality diagnoses. The algorithm is greedy in that it monotonically exploits part of the problem search space. The performance is determined by the efficiency of the underlying MBD engine; i.e., it is efficient with a fast (usually incomplete) procedure for computing minimal-diagnoses.

One advantage of Miranda over related k-fault ATPG algorithms is that it uncovers the maximum value of k. Furthermore, it does not impose any limitations on the model (e.g., it neither requires no stuck-at modes nor assumes unlimited observability). This makes our approach applicable not only to system testing but also to MBD benchmarking and to a wider range of Model-Based Reasoning (MBR) problems, such as optimal sensor placement [Console et al., 2000], active testing, etc. In this chapter the MFMC algorithm is applied to MBD benchmarking [Provan and Wang, 2007], but it can be applied to compute a set of MFMC test vectors covering all components in a system.

We have evaluated the performance of Miranda using the ISCAS85 benchmark extended with 4 smaller circuits from the 74XXX family. For the 74XXX circuits we have been able to exactly compute all MFMC observation vectors. Since deterministic MBD algorithms cannot compute the high fault-cardinalities associated with MFMC vectors for the ISCAS85 circuits, we have used a stochastic MBD oracle [Feldman et al., 2008c].

4.2 A Range of MBD Problems

We continue with presenting a number of problems for computing observation vectors.

4.2.1 Observation Vector Optimization Problems

Consider the set of diagnoses in a subset-minimal ambiguity group $\Omega^{\subseteq}(\mathrm{SD} \wedge \alpha)$. Let us denote the distribution of the diagnosis cardinalities in $\Omega^{\subseteq}(\mathrm{SD} \wedge \alpha)$ as $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$. Note that $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$ can be arbitrary, i.e., we can construct a system description SD and an observation α resulting in any $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$. In this chapter, SD is fixed, and the main focus of our work is how $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$ changes for various instantiations of the observation α. In particular we are interested in computing observations α that optimize certain parameters defined on the distribution $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$.

Figure 4.1 shows $\widetilde{\Omega}^{\subseteq}(\mathrm{SD} \wedge \alpha)$ for a weak-fault model of the 74182 combinatorial circuit (part of the 74XXX/ISCAS85 benchmark, cf. Sec. 4.6) and an arbitrary observation α. In addition to that, Fig. 4.1 illustrates a number of observation vector optimization problems.

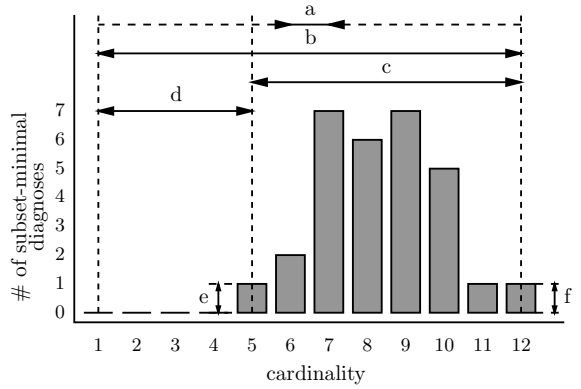


Figure 4.1: *An example cardinality distribution in a subset-minimal ambiguity group and a number of observation vector optimization problems*

From the seven observation vector optimization problems shown in Fig. 4.1, two are of practical significance to MBD: MFMC and MSMC. We next formally define those.

Problem 1 (MFMC Observation)**.** *Given a system* $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$*, compute an observation* α *(defined as Max-Fault Min-Cardinality (MFMC) observation) such that* ω *is a minimal-cardinality diagnosis of* $\mathrm{SD} \wedge \alpha$ *and* $|\omega|$ *is maximized.*

In addition to an MFMC observation, we also refer to an MFMC diagnosis ω of a model SD, which refers to any of the diagnoses entailed by an MFMC observation α. The cardinality of this diagnosis is denoted as $MFMC(\mathrm{SD})$ and, next to the associated MFMC observations, this is a key model property we seek to compute.

Problem 2 (MSMC Observation)**.** *Given a system* DS = ⟨SD, COMPS, OBS⟩*, compute an observation* α *(defined as Max-Size Min-Cardinality ambiguity group (MSMC) observation) such that* $|\Omega^{\leq}(\text{SD} \wedge \alpha)|$ *is maximized.*

MFMC observation vectors are relevant because many diagnostic algorithms (e.g. CDA* [Williams and Ragno, 2007]) search for diagnoses by checking diagnostic hypothesis in order of their cardinality. MFMC observation vectors result in difficult problems for such algorithms. MSMC observation vectors lead to difficult diagnostic problems due to the large uncertainty associated with the ambiguity group. MSMC observation vectors expose boundary conditions for a number of probing [de Kleer and Williams, 1987] and active testing [Feldman et al., 2009b] algorithms. In addition to that MSMC observation vectors minimize the upper bound of some isolation accuracy metrics [Feldman et al., 2010] and hence represent interesting worst-case scenarios.

The rest of the observation vector optimization problems are of less practical significance. The max-cardinality-range problem, for example, is of interest when we use algorithms that efficiently compute subset-minimal diagnoses [Feldman et al., 2008c] for the purpose of computing cardinality-minimal diagnoses. Clearly, such algorithms perform better with observation vectors leading to a smaller cardinality-range. Max-size subset-minimal group observation vectors pose challenges to algorithms that compute diagnostic entropy [de Kleer and Williams, 1987] or to diagnosis counting algorithms.

In this chapter we analyze mainly MFMC and we provide efficient algorithms for computing near optimal MFMC observation vectors. For the rest of the observation optimization problems we provide only short complexity analysis and some insights on building efficient algorithms.

4.2.2 MFMC Properties

When analyzing the properties of MBD problems, it is often convenient to abstract from the actual propositional **Wff** and to reason about the topology of the component interconnections. We next define structural abstraction in the spirit of Mozetič and Holzbaur [1994].

Definition 18 (Structural Abstraction)**.** Given a diagnostic system DS = ⟨SD, COMPS, OBS⟩, SD = $\{C_1, C_2, \ldots, C_n\}$, a structural abstraction $\text{SA}_{\text{DS}} = \langle V, E\rangle$ is a network constructed as follows. The set of nodes is $V = \text{COMPS} \cup \{\mathcal{I}, \mathcal{O}\}$, where node $\mathcal{I}$ is a source and node $\mathcal{O}$ is a sink. For each pair of components $\langle C_i, C_j\rangle$ that share variables, there is an edge in E. For each C_k that uses input variables, there is an input edge $\langle \mathcal{I}, C_k\rangle$ in E. For each C_l that uses output variables, there is an output edge $\langle C_l, \mathcal{O}\rangle$ in E.

Note that it is not always straightforward to obtain a structural abstraction network as defined above. The benchmark circuits from Sec. 2.4, however, have

input/output information from which we obtain the direction of the edges. Structural abstractions of the benchmark models can be obtained in polynomial time (we will not discuss this simple algorithm for brevity). Throughout this chapter, we assume a unit capacity for each edge in $\mathrm{SA_{DS}}$.

We continue the running example from Sec. 2.1.1. Figure 4.2 shows the structural abstraction (cf. Def 18) of the Boolean subtractor.

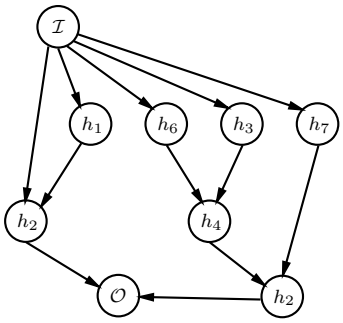


Figure 4.2: *A structural abstraction of the Boolean subtractor circuit shown in Fig. 2.1*

Hypothesis 2. Consider a diagnostic system $\mathrm{DS} = \langle \mathrm{SD}, \mathrm{COMPS}, \mathrm{OBS} \rangle$, $\mathrm{SD} \in$ **WFM**, and its structural abstraction graph $\mathrm{SA_{DS}}$. Let F be the min-cut/max-flow capacity of $\mathrm{SA_{DS}}$. It holds that $MFMC(\mathrm{SD}) \leq F$.

We denote the upper bound obtained from Hypothesis 2 as $MFMC^{\leq}(\mathrm{SD})$. This bound can be computed in polynomial time (recall that the min-cut/max-flow problem [Ford and Fulkerson, 1956] is in P).

MFMC and MSMC are related. There is a trivial MSMC upper bound:

$$MSMC(\mathrm{SD}) \leq \binom{|\mathrm{COMPS}|}{MFMC(\mathrm{SD})} \tag{4.1}$$

In a system with $|\mathrm{COMPS}|$ components, there are $\binom{|\mathrm{COMPS}|}{q}$ q-component combinations, hence the above upper bound.

We expect most MSMC observation vectors to be also MFMC observations. The reason for that is that, in a **WFM** and without applying an observation, the number of faults of cardinality k grows geometrically when increasing k.

4.3 Worst-Case Complexity

We will show that a restricted, decision version of the MFMC problem, MFMC_d, is Π_2^P-complete, by showing a reduction from Minmax-Clique (problem **GT2** in Schaefer and Umans [2002]) to MFMC_d. MFMC_d frames the problem of finding

an observation α that isolates a cardinality-minimal diagnosis ω of cardinality at least k.

Problem 3 (MFMC$_d$)**.** *Given a system* DS $= \langle$SD, COMPS, OBS$\rangle$ *and a constant* $k > 0$*, does there exist an observation* α *and a health assignment* ω *such that* $\min_{\omega \models \text{SD} \wedge \alpha} |\omega| \geq k$.

We next show the complexity of the MFMC$_d$ problem.

Theorem 4. *MFMC$_d$ is* Π_2^P*-complete.*[1]

Proof.

1. It is easy to see that MFMC$_d$ is in Π_2^P, since it is solvable by a polynomial-time nondeterministic machine with the use of an *NP*-complete oracle. The machine chooses nondeterministically an observation α and checks with an *NP*-hard algorithm whether $\exists \omega : \text{SD} \wedge \alpha \wedge \omega \not\models \bot, |\omega| < k$.

2. We now show a reduction from the Π_2^P-complete problem MINMAX-CLIQUE (problem **GT2** in Schaefer and Umans [2002]).

   MINMAX-CLIQUE is denoted by a graph $G = (V, E)$, a partition $(V_{i,j})_{i \in I, j \in J}$ of V, an integer κ, a function $t : I \to J$, and χ_t is the size of the largest clique in G restricted to $\bigcup_{i \in I} (V_{i,t(i)})$. In other words, we can denote $\chi_t(G) = \min_t \max_\chi \{|\chi| : \chi \subseteq V \text{ is a clique in } G_t\}$. G_t denotes the induced subgraph of G on the vertex set $V_i = \bigcup_{i=1}^{I} V_{i,t(i)}$.

   Given the partition, $|V| = I \cdot J = n$; hence $V = \{v_{1,1}, v_{1,2}, \cdots, v_{1,J}, v_{2,1}, \cdots, v_{2,J}, \cdots, v_{I,J}\}$. Partition i is given by $\{v_{i,1}, v_{i,2}, \cdots, v_{i,J}\}$ for $i = 1, 2, \cdots, I$.

   We assume that in MFMC$_d$, there are I health variables, each of which has J possible (abnormal) health values, i.e., health variable k has values given by $\{h_{k,1}, h_{k,2}, \cdots, h_{k,J}\}$, $k = 1, 2, \cdots, I$. Hence our space of possible health values is given by $\mathcal{H} = \{h_{1,1}, h_{1,2}, \cdots, h_{1,J}, h_{2,1}, \cdots, h_{2,J}, \cdots, h_{I,J}\}$.

   We construct a CNF formula f from G as follows.

   Variables The variables in f are as follows:

   - For each vertex $v_{i,j}$, introduce a variable $h_{v_{i,j}}$ (the semantics is that $h_{v_{i,j}}$ is true if $v_{i,j}$ is in the clique, and the only literals occurring in a clique χ are health literals).
   - For each $i = 1, 2, \cdots, I$, $j = 1, 2, \cdots, J$, and $k = 0, 1, 2, \cdots, \kappa$, introduce a variable $Y_{i,j,k}$.

   The second set of variables ensures that $Y_{i,j,k}$ is true if there are at least k true variables in the set $\{h_{v_{1,1}}, \ldots, h_{v_{1,J}}, h_{v_{2,1}}, \ldots, h_{v_{I,J}}\}$.

[1] The proof of this theorem first appeared in Provan [2008]. The material in this chapter is based on the joint paper submitted to Journal of Artificial Intelligence.

Clauses The clauses in f are as follows:

- For each pair of vertices $u, v \in G$ having no edge (u, v), add a clause $\neg h_v \vee \neg h_u$ to f.[2]
- For each $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$ and $k = 1, 2, \ldots, \kappa$, add clause $Y_{i-1,j,k} \vee (Y_{i-1,j,k-1} \wedge h_{v_{i,j}})$.
- For each $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$, add clauses enforcing $Y_{i,j,0} = \text{true}$.
- For each $k = 1, 2, \ldots, \kappa$, add clauses enforcing $Y_{0,0,k} = \text{false}$.
- Finally, add a clause containing just the variable $Y_{I,J,\kappa}$.

The above reduction clearly can be computed in polynomial time. Next we verify that the reduction is correct.

$\Rightarrow$ Suppose G has a clique χ_t of size κ. If this is true, then in f we must assign true to the variables h_v such that $v \in \chi_t$, and false to the other variables. We also assign true to those $Y_{i,j,k}$'s such that there are at least k true variables in $\{h_{v_1}, h_{v_2}, \ldots, h_{v_i}\}$, and we assign false to the other $Y_{i,j,k}$'s. This gives a satisfying assignment for f.

Further, under the partition function $t : I \to J$, $\chi_t(G) = \min_t \max_\chi\{|\chi| : \chi \subseteq V\}$. This corresponds in our satisfying assignment in f to there existing $\min_{\omega \models f \wedge \alpha} |\omega| = \kappa$, since every node in $\chi_t(G)$ consists of a health literal from a different partition, i.e., it consists of a minimal diagnosis ω such that $|\omega| = \kappa$.

$\Leftarrow$ Suppose f has a satisfying assignment. Let $\chi = \{v : h_v \text{ is assigned true}\}$. Then χ is a clique because for any pair of vertices $u, v \in \chi$, there cannot be a clause $\neg h_v \vee \neg h_u$ in f (because such a clause would not be satisfied), so there must be an edge $(u, v) \in G$. χ has size κ or more because $Y_{I,J,\kappa}$ must be true.

Further, a satisfying assignment in f must correspond to there existing $\chi_t(G) = \min_t \max_\chi\{|\chi| : \chi \subseteq V\}$, since under the satisfying assignment we have $\min_{\omega \models f \wedge \alpha} |\omega| = \kappa$, and every node in $\chi_t(G)$ consists of a health literal from a different partition, i.e., it consists of a minimal diagnosis ω such that $|\omega| = \kappa$.

□

4.4 System Description Simplifications

The complexity of computing MFMC is exponential in the number of components. Therefore, any reduction in number of components which does not affect *MFMC*(SD) provides significant computational advantage. Consider again the

[2] At this point, the satisfying assignments to f correspond to cliques in G.

circuit of Figure 2.1. Gates with assumables h_1 and h_2 can be combined into one gate h_1 - h_2 with one output d and three inputs x, y and p. This combination corresponds to the top dashed region of Fig. 4.3. The only other combinable components is the bottom region. No other combination is possible.

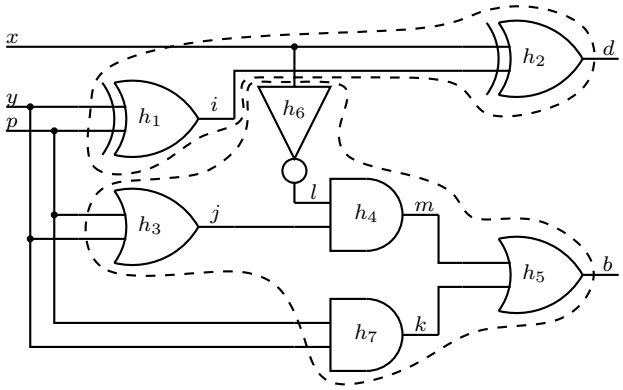


Figure 4.3: *Subtractor gates partitioned into two cones*

The intuition why h_1 and h_2 can be combined is as follows. The internal variable i is not observable. Therefore, any conflict involving h_1 will necessarily involve h_2 as well. If $x = 1$, $y = 1$, and $p = 1$, then if h_1 and h_2 were operating correctly $d = 1$. Observation $d = 0$ yields the conflict $\{h_1, h_2\}$. Therefore, for the purpose of calculating *MFMC*(SD), one can replace h_1 and h_2 with one composite component thereby reducing the complexity of searching for *MFMC*(SD).

In the case of stuck-at faults, there are many algorithms to collapse faults [Bushnell and Agrawal, 2000]. In general:

Proposition 3. *Any connected subset of components of* SD *having only one output can be replaced by a single composite component with equivalent behavior without changing MFMC*(SD).

Identifying all such subsets is typically too expensive. Fortunately, many such equivalent subsets can be identified very quickly:

Theorem 5. *If an output of component A is connected to an internal variable n and that variable is an input into only one component B, then A, B and n can be replaced with a single composite component with logically equivalent behavior without changing MFMC*(SD).

Proof (Sketch). By construction, the input-output behavior of the circuit remains unchanged with the replacement. As no other component is connected to n every conflict involving A must also include B. Therefore, B would appear in any diagnosis in which A appears. Thus replacing A and B with a single composite component does not change *MFMC*(SD). □

For example component h_1, variable i and component h_2 of Fig. 4.3 can be replaced with a single component with inputs x, y and p and output d. By iteratively applying the preceding theorem the number of components can be significantly reduced without changing $MFMC(\mathrm{SD})$. The combined $h_1 - h_2(h_{1,2})$ component is described by the following propositional **Wff**:

$$h_{1,2} \Rightarrow [i \Leftrightarrow \neg\,(y \Leftrightarrow p)] \wedge [d \Leftrightarrow \neg\,(x \Leftrightarrow i)]$$

Algorithm 5 demonstrates how to quickly identify cones. The approach is similar to that of [Siddiqi and Huang, 2007] and yields equivalent results. Algorithm 5 uses the notion of dominated sets. A dominated set of a component g ($\mathcal{G}[g]$) is defined as the set of components X such that any path from a component in X to any system output in OUT goes through g. A dominated set is also defined as a cone. We have seen in Theorem 5 that combining all the assumable variables in each cone preserves the MFMC value.

Algorithm 5 An algorithm for computing cones

```
1:  function ComputeCones(DS, IN, OUT) returns a set of sets
        inputs: DS, a diagnostic system (DS = ⟨SD, COMPS, OBS⟩)
                IN, OUT, variable sets (IN ∪ OUT = OBS, IN ∩ OUT = ∅)
        local variables: terminate, Boolean flag
                         S, stack
                         g, d_1, d_2, ..., d_n, component variables (n = |COMPS|)
                         G, set of sets, (G is indexed by component variable,
                         |G| = |COMPS|)
2:      repeat
3:          terminate ← T
4:          Push(S, OUT)
5:          while g ← Pop(S) do
6:              {d_1, d_2, ..., d_m} ← GetDrivenComponents(g)
7:              if G[d_1] ∩ G[d_2] ∩ ··· ∩ G[d_m] ≠ ∅ then
8:                  terminate ← F
9:              end if
10:             G[g] ← g ∪ (G[d_1] ∩ G[d_2] ∩ ··· ∩ G[d_m])
11:             Push(S, GetDrivingComponents(g))
12:         end while
13:     until terminate
14:     return{RemoveSubsumptions(G)}
15: end function
```

The idea of Alg. 5 is to start from the system outputs OUT and greedily identify cones. The cone identification is repeated until no further change in the dominated sets is achieved (lines 7 – 9). A dominated set is computed as the intersection of

the dominated sets of all driving components (a component x drives a component y is an output of x is connected to an input of y). A gate always dominates itself.

Algorithm 5 uses a stack S (and the auxiliary subroutines PUSH and POP) to store the gates whose dominated sets are scheduled for update. Note that Algorithm 5 does not satisfy any optimality criteria for the computed cones, i.e., the order in which output gates in line 4 and driving components in line 11 are pushed to S is not specified making Alg 5 non-deterministic.

As the properties of cones and Alg. 5 are well-studied we will skip the issues of termination and complexity. Extensive experimentation (cf. Sec. 4.6) shows that Alg. 5 is very efficient even for large circuits and produces results which preserve the MFMC values.

Important properties which improve running time dramatically on many circuits are:

- Suppose SD can be divided into two SD_1 and SD_2 such they share no internal variables. In this case, MFMC calculation is greatly simplified:

$$\mathit{MFMC}(\mathrm{SD}) = \mathit{MFMC}(\mathrm{SD}_1) + \mathit{MFMC}(\mathrm{SD}_2)$$

  Similarly,

$$\mathit{MSMC}(\mathrm{SD}) = \mathit{MSMC}(\mathrm{SD}_1) \times \mathit{MSMC}(\mathrm{SD}_2)$$

  The two formulae above follow from the fact that the global set of diagnoses of two independent systems is the Cartesian product of the diagnoses of the individual systems.

- Let $\mathrm{SD}|_{i=x}$ represent the system SD with input i set to x and the resulting circuit simplified by removing irrelevant components.

$$\mathit{MFMC}(\mathrm{SD}) = \max(\mathit{MFMC}(\mathrm{SD}|_{i=0}), \mathit{MFMC}(\mathrm{SD}|_{i=1}))$$

- If all of the inputs of a components are fed directly from the inputs (and no other component is driven by those inputs), then this set of inputs can be reduced to one and the component replaced by a buffer.

4.5 MFMC Algorithm

A naïve approach to compute $\mathit{MFMC}(\mathrm{SD})$ is to consider an exhaustive algorithm. Such an algorithm would enumerate all the $2^{|\mathrm{OBS}|}$ instantiations of the variables in OBS; one can easily show that only an assignments to *all* variables in OBS can be an MFMC observation vector as the MFMC problem is monotonic in respect to partial observations. For each full instantiation α, an MBD oracle computes the associated minimal fault cardinality.

Taking this exhaustive approach in our running example, we compute that $\mathit{MFMC}(\mathrm{SD}) = 2$ and that there is a total of 9 observation vectors discerning a

minimal-cardinality diagnosis of 2 faults (α_2 from Sec. 2.1 is an example of such an observation vector). From all the 32 possible observation vectors, there are 7, 16, and 9 observation vectors leading to a nominal, single-fault, and double-fault minimal-cardinality diagnosis, respectively.

Of course, such an exhaustive algorithm is computationally infeasible. We propose a stochastic method that trades optimality for a huge speedup, allowing very-high-k observations to be computed for very large circuits. Despite the inherent suboptimality of the stochastic approach, we will see in the experimental section of this chapter that, for smaller circuits from the 74XXX family, using MIRANDA results in optimal observation vectors. The success of our stochastic approach is that, as we will see in the experimental section, landscapes of typical MFMC search problems have many optima which are close or equal to the global optimum.

Alg. 6 assumes that an "all-healthy" mode of all assumable variables allows an input assignment to be propagated to all outputs. This is typical for health-models of digital circuits and for diagnosis problems.
Alg. 6 performs L independent attempts (restarts), each one starting from a random observation vector that corresponds to nominal health. This random starting point is computed as follows. First, the RANDOMINPUTS function assigns to each variable in IN a random value, the resulting term is then assigned to β. These random inputs are then fed to the COMPUTEOUTPUTS subroutine which assigns healthy values to the assumable variables, and computes the values of the variables in the output set OUT. This can be done by using a suitable propagation method like Binary Constraint Propagation [Zabih and McAllester, 1988]. The result of COMPUTEOUTPUTS is then assigned to γ.

Starting from this initial candidate observation $\beta \wedge \gamma$, Alg. 6 attempts to reduce the cardinality of a minimal-diagnosis consistent with an observation vector by "flipping" the values of the output variables. This is achieved by the auxiliary function FLIPLITERAL. At each step, the cardinality of the minimal-cardinality diagnosis is computed by a call to the MBD oracle FINDMCDIAGNOSIS. The observation leading to the highest-cardinality fault is stored and returned as a result of the MFMC computation.

Our MBD oracle must be carefully designed, since computing minimal cardinality diagnoses has a very high worst-case complexity: given arbitrary propositional theories in SD, the complexity of finding the cardinality of a minimal-cardinality diagnosis is $\Sigma_2^{\mathbf{P}}$-hard [Eiter and Gottlob, 1995]. The complexity decreases by imposing restrictions on the class of admissible system models, e.g., models with ignorance of abnormal behavior [de Kleer et al., 1992a], Horn theories, etc. For improving the speed of MBD in the average case, the literature has discussed a number of learning [Williams and Ragno, 2007] or approximation [Feldman et al., 2008c] techniques. Although our MFMC algorithm is transparent to the choice of the minimal-diagnosis oracle, the choice can be optimized when additional information on the specific properties of the system descriptions is available. In our implementation we use SAFARI (cf. Chapter 3) as a function for

Algorithm 6 A greedy stochastic MFMC algorithm

```
1: function MFMC(DS, IN, OUT, L) returns a term
       inputs: DS, a diagnostic system
                   DS = ⟨SD, COMPS, OBS⟩
                   IN, OUT, variable sets, inputs and outputs
                   IN ∪ OUT = OBS, IN ∩ OUT = ∅
                   L, an integer, number of runs
       local variables: β, γ, γ′, ω, R, terms
                        n, q, integers
                        l, a literal
2:     n ← 0
3:     q ← 0
4:     repeat
5:         β ← RandomInputs(IN)
6:         γ ← ComputeOutputs(DS, β, OUT)
7:         for all l ∈ γ do
8:             γ′ ← FlipLiteral(γ, l)
9:             ω ← FindMCDiagnosis(DS, β ∧ γ′)
10:            if |ω| > q then
11:                q ← |ω|
12:                γ ← γ′
13:                R ← β ∧ γ′
14:            end if
15:        end for
16:        n ← n + 1
17:    until n < L
18:    return R
19: end function
```

computing the minimal-diagnosis. Safari is a stochastic diagnostic solver which returns minimal diagnoses as an approximation to minimal-cardinality diagnoses but, as we will see later on, the incompleteness is compensated by the superior performance of this method.

We now illustrate the workings of the greedy algorithm on the Boolean subtractor circuit from our running example. We will consider only one run ($L = 1$). The RandomInputs function can return, for example, an input vector $\beta = \neg x \wedge y \wedge \neg p$. After assuming the "all-healthy" assignment $\omega_6 = h_1 \wedge h_2 \wedge \cdots \wedge h_7$, the subroutine ComputeOutputs computes the values of the output variables as $\gamma = d \wedge b$. Our greedy MFMC algorithm first changes the literal b in γ to $\neg b$. The inputs β and the modified γ makes an observation $\alpha_4 = \neg x \wedge y \wedge \neg p \wedge \neg b \wedge d$. The FindMCDiagnosis function, then, computes that $\mathit{MinCard}(\mathrm{SD} \wedge \alpha_4) = 1$. "Flipping" the sign of the second output variable d in γ leads to an observation $\alpha_5 = \neg x \wedge y \wedge \neg p \wedge \neg b \wedge \neg d$.

Diagnosing SD$\wedge\alpha_5$ results in $\mathit{MinCard}(\mathrm{SD}\wedge\alpha_5) = 2$. In bigger circuits, of course, "flipping" the second variable does not necessarily increase the cardinality of the minimal-cardinality diagnosis. Hence we need multiple attempts, caching the best observation computed so far. At this point there are no more output variables to "flip", hence the run returns α_5 leading to $\mathit{MFMC}(\mathrm{SD}) = 2$.

The number of minimal-cardinality diagnoses MIRANDA computes is determined by the number of restarts L and the number of output variables |OUT| in a system DS (recall that MIRANDA "flips" only output variables). The outermost loop of Alg. 6 performs L iterations, where in each iteration exactly $|\mathit{OUT}|$ literals are "flipped"; hence, the worst-case complexity is $O(L\,|\mathrm{OUT}|\,\Theta)$, where Θ is the computational complexity of a single minimal-cardinality diagnosis. Every time the sign of a literal is changed, MIRANDA computes a minimal-cardinality diagnosis, which gives us the stated complexity. In particular, with an incomplete diagnostic oracle like SAFARI [Feldman et al., 2008c] and an incomplete BCP method for consistency checking in the diagnostic procedure, the complexity of MIRANDA becomes $O(L\,|\mathrm{OUT}|\,|\mathrm{COMPS}|\,C)$, where C is the number of clauses in the CNF representation of SD [Zhang and Stickel, 2000]. This makes our algorithm applicable to larger models. Component-abstraction approaches, e.g., [Siddiqi and Huang, 2007], can also further increase the size of models that can be tackled.

4.6 Experimental Results

This section discusses some results from an implementation of the algorithms described in the previous sections.

4.6.1 Experimental Setup, Simplification Results and Bounds

We have used the benchmark circuits from Sec. 2.4 for experimenting with MIRANDA. The cone reductions performed by Alg. 5 yield significantly smaller number of components, as shown in the fifth column of Table 4.1. The cones formulations of Siddiqi and Huang [2007] and de Kleer [2008] yield an analogous reduction in components, which is an indication for the correctness of the algorithms.

The circuit with the largest number of cones (as computed by Alg. 5) is c6288 (1 456 cones). The biggest reduction in the number of components is for c1355 (the number of cones is $\approx 11\%$ of the number of original components). Note that c1355 and c499 implement the same Boolean function, where in c1355 each c499 logic gate is implemented from nand-gates.

Algorithm 5 is very efficient. All cone reductions were performed in less than 1 s.

The rightmost column of Table 4.1 ($\mathit{MFMC}^{\leq}$) shows the min-cut/max-flow capacity of the structural abstraction network of each model. We can see that

Table 4.1: *Reduction in* |COMPS| *using the prime implicate reduction rule*

Name	|IN|	|OUT|	|COMPS|	reduced	$MFMC^{\leq}$
74182	9	5	19	6	5
74L85	11	3	33	15	3
74283	9	5	36	14	5
74181	14	8	65	21	8
c432	36	7	160	59	7
c499	41	32	202	58	32
c880	60	26	383	77	26
c1355	41	32	546	58	32
c1908	33	25	880	160	25
c2670	233	140	1 193	167	64
c3540	50	22	1 669	353	22
c5315	178	123	2 307	385	116
c6288	32	32	2 416	1 456	32
c7552	207	108	3 512	545	105

in most of the cases $MFMC^{\leq}(\mathrm{SD}) = |\mathrm{OUT}|$, i.e., the min-cut is at the primary outputs. Exceptions are c2670, c5315, and c7552.

4.6.2 Computing MFMC Numbers and Vectors

Even after supplying MIRANDA with a state-of-the-art complete diagnostic solver [Feldman and van Gemund, 2006], the only circuits amenable to exhaustively enumerating all possible observation vectors were the ones from the 74XXX family. The exact cardinalities of the minimal-cardinality diagnoses of 74182, 74L85, 74283, and 74181 are 5, 3, 5, and 7, respectively.

Instead of configuring MIRANDA with a fixed number of restarts L, in our first experiment with Alg. 6, we show the number of restarts necessary for computing optimal MFMC values for the small 74XXX circuits. For this experiment MIRANDA was configured with the same complete diagnostic procedure which was used for the earlier, exhaustive experiment. Our implementation of MIRANDA reached the optimal MFMC values after performing 1.1, 3.4, 207.7, and 174.3 restarts for the 74182, 74L85, 74283, and 74181 circuits, respectively (the numbers are averages over 10 runs). The large value of L for the 74283 circuit arises because it has 2 MFMC observation vectors only. Similarly, the 74181 circuit has 456 observations, leading to an MFMC diagnosis of cardinality 7 from a total of 2^{22} observations.

The running time for finding the optimal 74XXX MFMC values (averaged over 10 runs) varied from 0.01 s for the 74182 circuit to 34.2 min for 74181. The

long running time for reaching the MFMC of 74181 model comes from the poor performance of the complete diagnostic procedure we have used (despite the fact that we have employed a state-of-the-art solver). This is not surprising, considering that, with traditional MBD algorithms for computing minimal-cardinality diagnoses, the computational cost of finding a k-minimal-cardinality diagnosis increases with k.

Consider an MFMC algorithm that uses the GDE [de Kleer and Williams, 1987] algorithm as a diagnostic oracle. GDE first constructs all minimal conflicts and then constructs the minimal diagnoses from the minimal conflicts. This approach constructs far more conflicts than are necessary to find one minimal-cardinality diagnosis, and hence its cardinality. If there are a large number of minimal diagnoses, most of this is useless work. The c6288 circuit can never be analyzed by this approach as it yields an exponential number of minimal conflicts. These result from its inherently parallel structure illustrated in Fig. 4.4. Each box in Fig. 4.4 represents a 2-bit adder (there are 240 of them). In addition to that, c6288 uses an and-gate that connects the two least significant inputs to the output o_0 (this and-gate is not shown in Fig. 4.4).

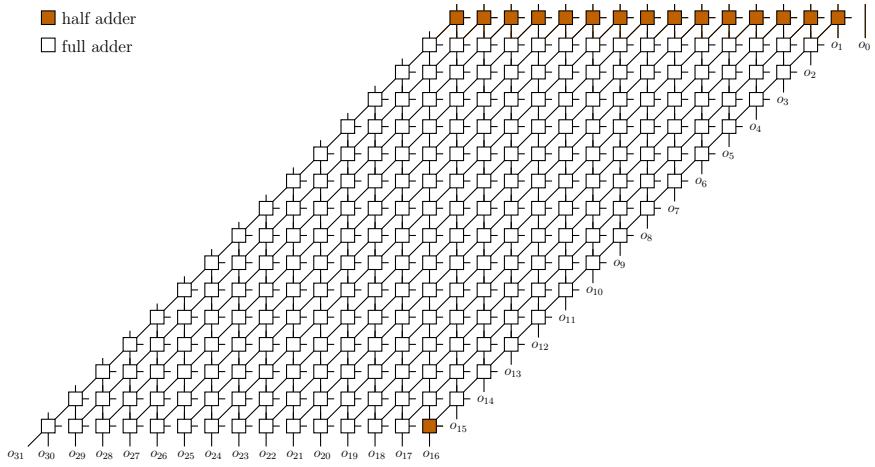


Figure 4.4: *The most difficult to analyze circuit from the* ISCAS85 *test suite. It is a* 16×16 *bit parallel multiplier built out of half and full adders.*

The MFMC of the c6288 circuit can be determined by visually inspecting the circuit diagram (cf. Fig. 4.4). We have brute-forced a 3-bit multiplier, and its MFMC value is 3. The circuit in Fig. 4.4 can be partitioned by cutting 16 connections. Hence, we hypothesize, that $MFMC(\text{c6288}) = 16$. Furthermore, we hypothesize that the MFMC value of a regular, n-bit multiplier is n.

Given a system DS, we denote as $g(\text{DS})$ the pdf of the minimal-cardinalities of the diagnoses of all observations in DS. Figure 4.5 shows a histogram of the true minimal-diagnosis cardinalities for the four 74XXX circuits for which we have

exhaustively determined $g(\mathrm{DS})$, fitted by a normal distribution.

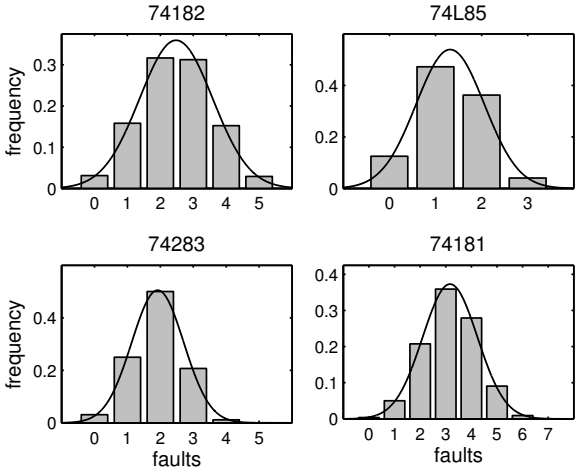


Figure 4.5: 74XXX *minimal-cardinalities pdf*

To describe our error bounds, we focus on the partitioned observation vector $\alpha = \mathrm{IN} \cup \mathrm{OUT}$. Given an observation α leading to a k-fault minimal diagnosis, we associate a nominal-diagnosis observation α_n, which may differ from α only in the OUT sub-vector. The number of OUT-values in which α and α_n differ is called the *distance* of α, $D(\mathrm{SD}, \alpha)$. If $n = |\mathrm{OUT}|$ is the number of output variables in SD, then starting from any nominal observation α_n, there are ${}_nC_k$ ways to select a distance-k vector α, each of which corresponds to a diagnosis. In the case where each such diagnosis is a minimum cardinality diagnosis, $g(\mathrm{SD})$ is binomially-distributed. This is true given some assumptions on the model SD (e.g., SD is a weak-fault model of a deterministic Boolean circuit).

Although the above model is an approximation, it provides useful bounds on MFMC errors. For the 74XXX and ISCAS85 benchmarks, the fraction of "m-flips" resulting in minimal-cardinality diagnoses of cardinality smaller than m is relatively small and does not vary significantly for different m.
To overcome the complexity of using a complete diagnostic procedure, in the rest of our experiments, we have used the incomplete SAFARI algorithm, which is virtually insensitive to k (cf. Chapter 3). The stochastic MBD oracle dramatically increases the performance of MIRANDA, at the price of approximating the cardinality of a minimal-cardinality diagnosis. While SAFARI returns minimal diagnoses, they are not necessarily minimal-cardinality diagnoses. To some extent the optimistic MFMC values from SAFARI compensate for the pessimistic effect of the limited number of MIRANDA retries, but still cause MIRANDA to produce optimistic MFMC values for the ISCAS85 circuits.

Table 4.2 shows the MFMC data and run times using MIRANDA and SAFARI.

Table 4.2: MIRANDA/SAFARI *MFMC approximations*

			$L = 1$		$L = 100$	
	Exhaustive	$N = 1$		$N = 100$	$N = 1$	$N = 100$
Name	MFMC	Time [s]	MFMC	MFMC	MFMC	MFMC
74182	5	0.001	5	5	5	5
74L85	3	0.003	3	3	6	3
74283	5	0.004	4	2	7	4
74181	7	0.013	8	5	9	6
c432	–	0.07	4	3	12	5
c499	–	0.343	19	17	25	20
c880	–	0.595	17	20	21	26
c1355	–	0.838	17	15	22	19
c1908	–	2.801	15	14	27	20
c2670	–	21.208	21	33	26	41
c3540	–	11.019	10	11	27	21
c5315	–	91.084	27	36	53	55
c6288	–	252.985	22	16	48	18
c7552	–	195.130	29	29	44	42

The third column measures the time for executing one run of MIRANDA for SAFARI configured with $N = 1$ (hence the total number of runs of SAFARI is $|\text{OUT}|$). The fourth column of Table 4.2 shows the value of the MFMC approximation reached during this one run. Recall, that SAFARI overestimates the cardinality of the minimal cardinality diagnosis (cf. Chapter 3). As a result of the single run ($L = 1$), MIRANDA underestimates the cardinality of the MFMC diagnosis. The error factors are with different signs, and combining them leads to a smaller error.

Increasing the number of SAFARI restarts N decreases the overestimation of the cardinality of the minimal-cardinality diagnosis by SAFARI. The MFMC results for $N = 100$ (number of restarts of SAFARI) and $L = 1$ (number of restarts of MIRANDA) are shown in the fifth column of Table 4.2. We can see that the values in the fifth column are always smaller than the values in the fourth columns.

The MIRANDA values for $N = 1$ (number of restarts of SAFARI) and $L = 100$ (number of restarts of MIRANDA) are shown in the sixth column of Table 4.2. In this case MIRANDA overestimates the MFMC values. It can be seen that the values in this configuration are always greater than the MFMC approximations in the fourth column of Table 4.2 ($N = 1$ (SAFARI), $L = 1$ (MIRANDA)).

The rightmost column of Table 4.2 shows the MIRANDA values for $N = 100$ (number of restarts of SAFARI) and $L = 100$ (number of restarts of MIRANDA). This configuration is computationally very expensive. For the large ISCAS85

circuits we have used a cluster as the CPU time that is needed is approximately 10 000 times the time shown in the third column of Table 4.2. The MFMC values for MIRANDA, $L = 100$, SAFARI, $N = 100$ are close to these for MIRANDA, $L = 1$, SAFARI, $N = 1$. For 74XXX, the approximated values are, in the worst case, 80% of the respective global optima.

The only benchmark model for which we could compute an MSMC value with an exhaustive search was 74182, $MSMC(74182) = 400$. Figure 4.6 shows a bivariate histogram of the number of observation vectors per cardinality and per number of minimal-cardinality diagnoses. There are 36 MSMC observation vectors. Of those, 18 observations lead to a minimal-cardinality diagnosis of cardinality 4 and 18 observations lead to a minimal-cardinality diagnosis of cardinality 5. All MSMC observations lead to nearly MFMC diagnoses. As it is visible from Fig. 4.6, MFMC observations lead to all sizes of minimal-cardinality ambiguity groups. For example, there are 7 MFMC observations that lead to a unique minimal-cardinality diagnosis.

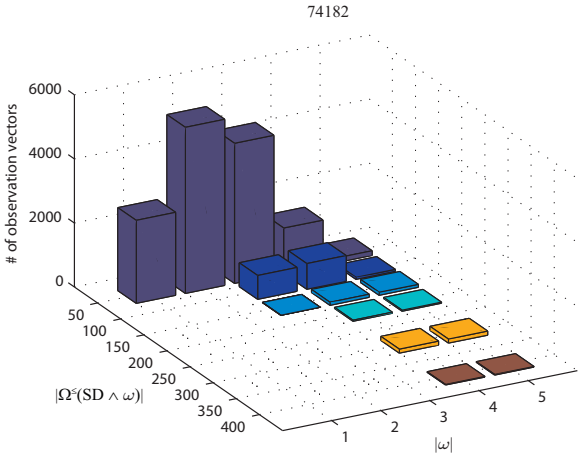


Figure 4.6: *Observation vectors bivariate histogram for* 74182

4.7 Summary

This chapter introduced the problem of computing MFMC observation vectors and suggested a greedy stochastic algorithm for computing such vectors. Our algorithm is very efficient, given a fast subroutine for computing the cardinality of a minimal-cardinality diagnosis. The MFMC of real-world systems is an important property quantifying the diagnosability of a model, as it shows the maximum number of malfunctioning components that can be distinguished observing

a set of variables. We have empirically demonstrated MIRANDA on a number of 74XXX/ISCAS85 combinatorial circuits, computing the MFMC of these circuits.

Chapter 5

A Model-Based Active Testing Approach to Sequential Diagnosis

Model-based diagnostic reasoning often leads to a large number of diagnostic hypotheses. The set of diagnoses can be reduced by taking into account extra observations (passive monitoring), measuring additional variables (probing) or executing additional tests (sequential diagnosis/test sequencing). In this chapter we combine the above approaches, as well as from Automated Test Pattern Generation (ATPG) and Model-Based Diagnosis (MBD) into a framework called FRACTAL (FRamework for ACtive Testing ALgorithms). Apart from the inputs and outputs that connect a system to its environment, in active testing we consider additional input variables to which a sequence of test vectors can be supplied. We address the computationally hard problem of computing optimal control assignments (as defined in FRACTAL) in terms of a greedy approximation algorithm called FRACTAL$^{\mathrm{G}}$. We model the decrease in the number of remaining minimal cardinality diagnoses produced by an active testing algorithm as a geometric decay process and measure the algorithm's performance in terms of the decay rate. FRACTAL$^{\mathrm{G}}$ employs approximation approaches, such as stochastic sampling for computing the expected number of remaining minimal cardinality diagnoses and greedy search for computing the optimal control assignment, which results in a computationally efficient method. We compare the optimality of FRACTAL$^{\mathrm{G}}$ to that of two more FRACTAL algorithms: FRACTAL$^{\mathrm{A}}$ and FRACTAL$^{\mathrm{P}}$. FRACTAL$^{\mathrm{A}}$ is based on ATPG and sequential diagnosis while FRACTAL$^{\mathrm{P}}$, although not an active testing algorithm, provides a baseline for comparing the lower bound on the number of reachable diagnoses for the FRACTAL algorithms. We empirically evaluate the trade-offs of the three FRACTAL algorithms by performing extensive experimentation on the ISCAS85/74XXX benchmark of combinational circuits.

5.1 Introduction

Combinational Model-Based Diagnosis (MBD) approaches [de Kleer and Williams, 1987] often lead to a large number of diagnoses, in the worst-case, exponential in the number of components. Combining multiple sensor readings (observation vectors) [Pietersma and van Gemund, 2006] helps in a limited number of cases because the approach is inherently passive, i.e., there are situations in which the observations repeat themselves (for example, in systems that are stationary, pending a reconfiguration).

Sequential diagnosis algorithms [Shakeri, 1996] can be used as an alternative to the above passive approach with better decay of the number of diagnostic hypotheses. The decay rate depends on the tests and test dictionary matrix, and is bounded from below for tests with binary outcomes. Sequential diagnosis algorithms suffer from a number of other limitations. Early approaches assume single-faults while multiple-fault sequential diagnosis is super-exponential (Σ_2^p or harder) [Shakeri et al., 2000].

As observations (test outcomes) are not known in advance, the goal of a diagnostician is to create a policy that minimizes the diagnostic uncertainty on average, i.e., one aims at minimizing the average depth of a test tree. Pattipati and Alexandridis [1990] have shown that under certain conditions (e.g., unit test costs and equal prior fault probabilities) one-step look-ahead policy leads to an optimal average depth of the test tree while de Kleer et al. [1992b] have shown that one-step look-ahead delivers good practical result for a range of combinational circuits.

As reliable component failure rates may be problematic to obtain, we assume equally likely and small prior probabilities of failure and measure the diagnostic uncertainty as the number of Minimal Cardinality (MC) diagnoses. The latter approach also has computational advantages. It can be shown that with equal and small prior probabilities of failure, the diagnostic entropy, e.g., as used by de Kleer and Williams [1987], can be computed directly from the number of MC diagnoses.

The computational complexity of deterministic algorithms for sequential diagnosis increases with respect to both the fault-cardinality and the number of tests (the size of the test dictionary). To enable performance to scale up to real-world problems, which may have high fault-cardinality and a large number of tests, we propose FRACTAL$^{\mathrm{G}}$–a low-cost greedy stochastic approach that maintains exponential decay of the number of MC diagnoses. Instead of assuming single faults or timing out, FRACTAL$^{\mathrm{G}}$ may result in suboptimal but still exponential decay.

FRACTAL$^{\mathrm{G}}$ is specified within a formally-defined framework called FRACTAL (FRamework for ACtive Testing ALgorithms). In FRACTAL, in addition to the input (IN) and output (OUT) variables, we also discern control (CTL) variables. Controls are similar to inputs, except they can be modified by users while the system is connected to its environment. The use of models from first principles and controls in FRACTAL allows us to eliminate the need of designing explicit tests

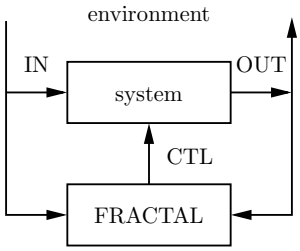


Figure 5.1: *Active testing dataflow for* FRACTAL

and test dictionaries. Our algorithms implicitly create test matrices leading to optimal decay based on the built-in testing capabilities of the system. The above approach we call active testing; i.e., a technique using models from first principle and controls for creating test sequences which reduce the diagnostic uncertainty of a system. The architecture in which we use FRACTAL and active testing is shown in Fig. 5.1.

We study the performance of FRACTAL^{G} compared to two alternatives: ATPG-based sequential diagnosis (FRACTAL^{A}), and probing [de Kleer and Williams, 1987] (FRACTAL^{P}). FRACTAL^{A} is derived from sequential testing, it is deterministic, myopic and allows us to evaluate how well a single-step lookahead approach works on the given model. Although probing is not classified as a technique for sequential diagnosis, it can be viewed as a process generating tests using additional control circuitry (machine or human) to execute the probe whereby some output reveals the internal variable. Its significance is that it shows the lower bound on the number of diagnoses achievable for a model extended with unlimited CTL circuitry.

Table 5.1: *Properties of techniques for sequential diagnosis*

Technique	Model	User Actions	Automatic Tests	Performance	Cost
(Passive) Monitoring	first principles	-	-	variable[1]	low
Sequential Diagnosis	test dictionary	apply test	no	good	variable[2]
FRACTAL^{A}	first principles	apply controls	yes	variable[3]	medium
FRACTAL^{G}	first principles	apply controls	yes	good	high
Probing (FRACTAL^{P})	first principles	measure internals	-	binary search	medium

[1] Depends on the environment (IN/OUT data).
[2] Speed deteriorates rapidly with multiple-faults.
[3] Depends on the model topology.

Table 5.1 summarizes the properties of the various techniques for sequential diagnosis discussed in this chapter. FRACTAL eliminates the need for using tools (e.g., [Deb et al., 2000]) for building tests and test dictionaries, as they can be automatically constructed from design specifications and models. At the same time FRACTAL delivers comparable or better diagnostic convergence at reasonable computational price.

Our contributions are as follows:

- We devise an approach for reducing the diagnostic uncertainty, called active testing in a framework that generalizes over sequential diagnosis and MBD, allows combination of multiple passive sensor readings, which does not require explicit tests and test dictionaries.
- We design FRACTAL^{A}—a single-step look-ahead algorithm based on ATPG—for solving the active testing problem.
- We design and implement FRACTAL^{G}–a greedy approximation algorithm for active testing that overcomes the limitations of FRACTAL^{A} and offers a trade-off in computational complexity vs. optimality for reducing the diagnostic uncertainty. We compare FRACTAL^{G} and FRACTAL^{A}.
- We implement FRACTAL^{P} and use it as a computationally efficient, myopic (one-step lookahead), easy-to-analyze baseline technique for reducing diagnostic uncertainty. Although FRACTAL^{P} is technically not an active testing algorithm, the implementation of probing and active testing in a common framework and the unified experimentation help understanding the cost vs. performance trade-offs in (active and passive) testing vs. probing strategies.
- We present extensive empirical data on 74XXX/ISCAS85 circuits, which evaluate FRACTAL^{A}, FRACTAL^{G}, and FRACTAL^{P} in terms of their ability to reduce the number of remaining diagnoses according to a geometric decay function.

This chapter is organized as follows. Section 5.2 introduces related work. Section 5.3 presents basic MBD notions, the concept of remaining number of diagnoses and a framework for sequential diagnosis. Section 5.4 introduces a stochastic sampling-based algorithm for computing the expected number of cardinality-minimal diagnoses. Section 5.5 describes the FRACTAL^{A}, FRACTAL^{G}, and FRACTAL^{P} algorithms. Section 5.6 shows experimental results. Finally, in Sec. 5.7, we summarize this chapter and discuss future work.

5.2 Related Work

Early work aimed at diagnostic convergence is the approach by de Kleer and Williams [1987] which computes the probe sequence that reduces diagnostic entropy

using a myopic search strategy. Unlike their work, in active testing we assume that probes are not available, other than indirectly exposed through diagnosis based on test vectors, which offers an automated solution.

Generating test vectors to deduce faults has received considerable attention. Automatic test pattern generation (ATPG) aims at verifying particular, single-faults [Stephan et al., 1996]. ATPG differs from active testing in that the vectors are specific for particular single-faults, whereas active testing generates a sequence of vectors to isolate *unknown, multiple*-faults, a much harder problem.

Active testing bears some resemblance with sequential diagnosis, which also generates a sequence of test vectors [Kundakcioglu and Ünlüyurt, 2007; Pattipati and Alexandridis, 1990; Raghavan et al., 1999; Tu and Pattipati, 2003]. The principle difference is that in sequential diagnosis a fault dictionary is used ("fault matrix"). This pre-compiled dictionary has the following drawback: in order to limit the (exponential) size of the dictionary, the number of stored test vectors is extremely small compared to the test vector space. This severely constrains the optimality of the vector sequence that can be generated, compared to active testing, where test vectors are computed on the fly using a model-based approach. Furthermore, the matrix specifies tests that only have a binary (pass/fail) outcome, whereas active testing exploits all the system's outputs, leading to faster diagnostic convergence. In addition, we allow the inputs to be dynamic, which makes our framework suitable for online fault isolation.

The sequential diagnosis problem studies optimal trees when there is a cost associated with each test [Tu and Pattipati, 2003]. When costs are equal, it can be shown that the optimization problem reduces to a next best control problem (assuming one uses information entropy). In this thesis, a diagnostician who is given a sequence S and who tries to compute the *next* optimal control assignment would try to minimize the expected number of remaining diagnoses $|\Omega(S)|$.

Our task is harder than that of Raghavan et al. [1999], since despite the diagnosis lookup using a fault dictionary, the diagnosis task is NP-hard; in our case we compute a new diagnosis after every test. Hence we have an NP-hard sequential problem interleaved with the complexity of diagnostic inference at each step (in our case the complexity of diagnosis is Σ_2^p-hard). Apart from the above-mentioned differences, we note that optimal test sequencing is infeasible for the size of problems in which we are interested.

Model-Based Testing (MBT) [Struss, 1994] is a generalization of sequential diagnosis. The purpose of MBT is to compute inputs manifesting a certain (faulty) behavior. The main differences from our active testing approach are that MBT (1) assumes that all inputs are controllable and (2) MBT aims at *confirming* single faulty behavior as opposed to maximally decreasing the diagnostic uncertainty.

Brodie et al. [2003] cast their models in terms of Bayesian networks. Our notion of entropy is the size of the diagnosis space, whereas Brodie et al. [2003] use decision-theoretic notions of entropy to guide test selection.

We solve a different problem than that of Alur et al. [1995]; Heinz and Sachenbacher [2008]. Alur et al. [1995]; Heinz and Sachenbacher [2008] assume a non-de-

terministic system defined as an automaton. In contrast, our framework assumes a static system (plant model) for which we must compute a temporal sequence of tests to best isolate the diagnosis.

Esser and Struss [2007] also adopt an automaton framework for test generation, except that, unlike Heinz and Sachenbacher [2008] or Alur et al. [1995], they transform this automaton to a relational specification, and apply their framework to software diagnosis. This automaton-based framework accommodates more general situations than ours does, such as the possibility that the system's state after a transition may not be uniquely determined by the state before the transition and the input, and/or the system's state may be associated with several possible observations. In our MBD framework, a test consists of an instantiation of several variables, which corresponds to the notion of test sequence within the automaton framework of Heinz and Sachenbacher [2008].

A recent approach to active diagnosis is described by Kuhn et al. [2008], where additional test vectors are computed to optimize the diagnosis while the system (a copier) remains operational. Their work differs from ours in that plans (roughly analogous to test sequences) with a probability of failure T are computed statically, and a plan remains unmodified even if it fails to achieve its desired goal (a manifestation of a failure with probability close to T). Conversely, FRACTAL dynamically computes next-best control settings in a game-like manner.

Feldman et al. [2009b] discusses an early version of FRACTAL$^{\mathrm{G}}$. This chapter (1) generalizes Feldman et al. [2009b], (2) introduces FRACTAL$^{\mathrm{A}}$ and FRACTAL$^{\mathrm{P}}$, (3) extends the experimental results, and (4) provides a comparison of the different FRACTAL approaches.

5.3 Concepts and Definitions

Our discussion starts by adopting relevant MBD notions. Central to MBD, a *model* of an artifact is represented as a propositional **Wff** over a set of variables V. We will discern four subsets of these variables: *assumable*, *observable*[1], *control*, and *internal* variables. This gives us our initial definition:

Definition 19 (Active Testing System)**.** An active testing system ATS is defined as ATS = ⟨SD, COMPS, CTL, OBS⟩, where SD is a propositional **Wff** over a variables set V, COMPS ∪ OBS ∪ CTL ⊆ V, and COMPS, OBS, and CTL are sets containing assumable, observable, and control variables, respectively.

The set of internal variables is denoted as INT, INT = $V \setminus \{\mathrm{COMPS} \cup \mathrm{OBS} \cup \mathrm{CTL}\}$. Throughout this chapter we assume that OBS, COMPS, and CTL are disjoint, and SD $\not\models \bot$. Sometimes it is convenient (but not necessary) to split OBS into non-controllable inputs IN and outputs OUT (OBS = IN ∪ OUT, IN ∩ OUT = ∅).

[1] In the MBD literature the assumable variables are also referred to as "component", "failure-mode", or "health" variables. Observable variables are also called "measurable" variables.

5.3.1 Running Example

We will use the Boolean circuit shown in Fig. 5.2 as a running example for illustrating all notions and the algorithm shown in this chapter. The 2-to-4 line demultiplexer consists of four Boolean inverters and four and-gates.

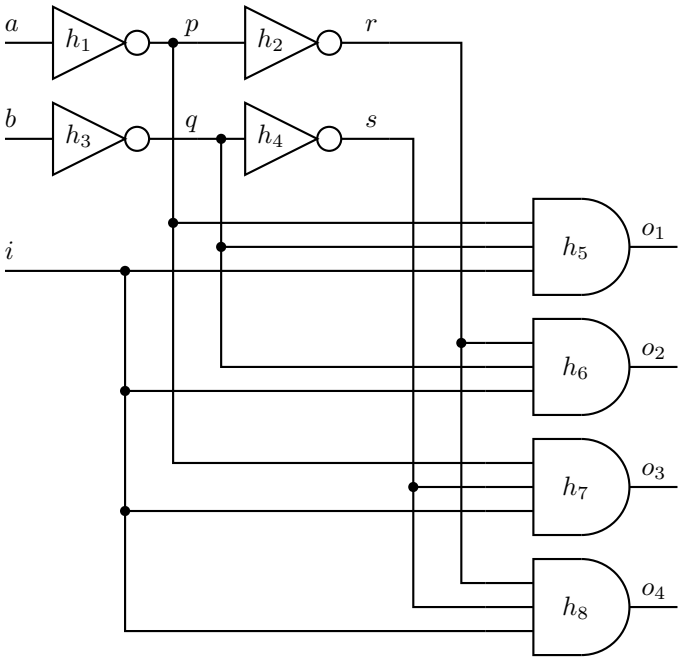


Figure 5.2: *A demultiplexer circuit*

The expression $h \Rightarrow (o \Leftrightarrow \neg i)$ models an inverter, where the variables i, o, and h represent input, output, and health respectively. Similarly, an and-gate is modeled as $h \Rightarrow (o \Leftrightarrow i_1 \wedge i_2 \wedge i_3)$. The above propositional formulae are copied for each gate in Fig. 5.2 and their variables subscripted and renamed in such a way as to ensure a proper disambiguation and to connect the circuit. The result is the following propositional model:

$$\mathrm{SD} = \begin{cases} [h_1 \Rightarrow (a \Leftrightarrow \neg p)] \wedge [h_2 \Rightarrow (p \Leftrightarrow \neg r)] \\ [h_3 \Rightarrow (b \Leftrightarrow \neg q)] \wedge [h_4 \Rightarrow (q \Leftrightarrow \neg s)] \\ h_5 \Rightarrow (o_1 \Leftrightarrow i \wedge p \wedge q) \\ h_6 \Rightarrow (o_2 \Leftrightarrow i \wedge r \wedge q) \\ h_7 \Rightarrow (o_3 \Leftrightarrow i \wedge p \wedge s) \\ h_8 \Rightarrow (o_4 \Leftrightarrow i \wedge r \wedge s) \end{cases} \tag{5.1}$$

The set of assumable variables is $\mathrm{COMPS} = \{h_1, h_2, \ldots, h_8\}$, the observable variables are $\mathrm{OBS} = \{a, b, o_1, o_2, o_3, o_4\}$, and the set of control variables is the sin-

gleton CTL = $\{i\}$. Note the conventional selection of the sign of the "health" variables $h_1, h_2, \ldots, h_n$. Other authors use "ab" for abnormal.

5.3.2 Sequential Diagnosis

Throughout this chapter, we assume that the health of the system under test does not change during the test (i.e., the same inputs and a fault produce the same outputs) and call this assumption stationary health.

Lemma 8. *Given a system* ATS*, a stationary health state for its components* ω*, and an observation sequence* S*, it follows that* $\omega \in \Omega(\mathrm{SD}, \alpha_1) \cap \Omega(\mathrm{SD}, \alpha_2) \cap \ldots \cap \Omega(\mathrm{SD}, \alpha_k)$.

Proof. The above statement follows immediately from the stationary health assumption and Def. 5. □

Lemma 8 can be applied only in the cases in which *all* diagnoses are considered. If we compute subset-minimal diagnoses in a weak-fault model, for example, the intersection operator has to be redefined to handle subsumptions. To handle non-characterizing sets of diagnoses (e.g., MC or first m diagnoses), we provide the following definition.

Definition 20 (Consistency-Based Intersection)**.** Given a set of diagnoses D of $\mathrm{SD} \wedge \alpha$, and a posteriori observation α', the intersection of D with the diagnoses of $\mathrm{SD} \wedge \alpha'$, denoted as $\Omega^{\cap}(D, \alpha')$, is defined as the set D' ($D' \subseteq D$) such that for each $\omega \in D'$ it holds that $\mathrm{SD} \wedge \alpha' \wedge \omega \not\models \bot$.

It is straightforward to generalize the above definition to an observation sequence S.

Definition 21 (Remaining Minimal-Cardinality Diagnoses)**.** Given a diagnostic system ATS and an observation sequence S, the set of remaining diagnoses $\Omega(S)$ is defined as $\Omega(S) = \Omega^{\cap}(\Omega^{\cap}(\cdots \Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha_1), \alpha_2), \cdots), \alpha_k)$.

We use $|\Omega(S)|$ instead of the more precise diagnostic entropy as defined by de Kleer et al. [de Kleer and Williams, 1987] and subsequent works, as this allows low-complexity estimations (discussed in Sec. 5.4). In particular, if all diagnoses are of minimal-cardinality and the failure probability of each component is the same, then the gain in the diagnostic entropy can be directly computed from $|\Omega(S)|$.

5.4 Computing the Expected Number of MC Diagnoses

Active testing aims to minimize the expected number of diagnoses that result from the possible set of outputs that may occur from a give control vector. In this section we present an algorithm to approximate this expectation.

We will compute the expected number of diagnoses for a set of observable variables M ($M \subseteq \text{OBS}$). The initial observation α and the set of MC diagnoses $D = \Omega^{\leq}(\text{SD}, \alpha)$ modify the probability density function (pdf) of subsequent outputs (observations), i.e., a subsequent observation α' changes its likelihood. The (non-normalized) a posteriori probability of an observation α', given a function $\Omega^{\leq}$ that computes the set of MC diagnoses and an initial observation α is:

$$\Pr(\alpha'|\text{SD}, \alpha) = \frac{|\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|}{|\Omega^{\leq}(\text{SD}, \alpha)|} \tag{5.2}$$

The above formula comes by quantifying how a given a priori set of diagnoses restricts the possible outputs (i.e., we take as probability the ratio of the number of remaining diagnoses to the number of initial diagnoses). In practice, there are many α for which $\Pr(\alpha'|\text{SD}, \alpha) = 0$, because a certain fault heavily restricts the possible outputs of a system (i.e., the set of the remaining diagnoses in the nominator is empty).

The expected number of remaining MC diagnoses for a variable set M, given an initial observation α, is then the weighted average of the intersection sizes of all possible instantiations over the variables in M (the weight is the probability of an output):

$$E^{\leq}(\text{SD}, M|\alpha) = \frac{\sum_{\alpha' \in M^*} |\Omega^{\cap}(D, \alpha')| \cdot \Pr(\alpha'|\text{SD}, \alpha)}{\sum_{\alpha' \in M^*} \Pr(\alpha'|\text{SD}, \alpha)} \tag{5.3}$$

where $D = \Omega^{\leq}(\text{SD}, \alpha)$ and M^* is the set of all possible assignment to the variables in M. Replacing (5.2) in (5.3) and simplifying gives us the following definition:

Definition 22 (Expected Intersection Size)**.** Given a system ATS and an initial observation α, the expected remaining number of MC diagnoses $E^{\leq}(\text{SD}, \text{OBS}|\alpha)$ is defined as:

$$E^{\leq}(\text{SD}, \text{OBS}|\alpha) = \frac{\sum_{\alpha' \in \text{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|^2}{\sum_{\alpha' \in \text{OBS}^*} |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \alpha')|} \tag{5.4}$$

where OBS^* is the set of all possible assignments to all variables in OBS.

Two of the algorithms presented in this chapter compute the expected number of remaining MC diagnoses for one variable. As a result the expectation expression in (5.4) simplifies to:

$$E^{\leq}(\text{SD}, v|\alpha) = \frac{|\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), v)|^2 + |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \neg v)|^2}{|\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), v)| + |\Omega^{\cap}(\Omega^{\leq}(\text{SD}, \alpha), \neg v)|} \tag{5.5}$$

The complexity of computing (5.5) depends only on the length of the sequence S, the complexity of the MC oracle computing $\Omega^{\leq}(\mathrm{SD}, \alpha)$ and the complexity of the intersection algorithm.

5.4.1 Computing the Expectation Using Importance Sampling

The key insight which allows us to build a fast method for computing the expected number of remaining diagnoses is that the prior observation (and respectively the set of MC diagnoses) shifts the probability of the outputs. Hence, an algorithm which samples the possible input assignments (it is safe to assume that inputs are equally likely) and counts the number of *different* observations, given the set of prior diagnoses, would produce a good approximation.

Algorithm 7 Approximate expectation of $\Omega(S)$

```
1:  function EXPECTATION(ATS, γ, D) returns a real
        inputs: ATS, active testing system
                γ, term, control vector
                D, set of diagnoses, prior diagnoses
        local variables: α, β, ω, terms
                         s, q, integers, initially 0
                         Z, set of terms, samples
                         Ê, real, expectation
2:      Z ← ∅
3:      repeat
4:          α ← RANDOMINPUTS(SD, IN)
5:          for all ω ∈ D do
6:              β ← INFEROUTPUTS(SD, OUT, α ∧ γ, ω)
7:              if α ∧ β ∉ Z then
8:                  Z ← Z ∪ {α ∧ β}
9:                  q ← q + |Ω∩(D, α ∧ β ∧ γ)|²
10:                 s ← s + |Ω∩(D, α ∧ β ∧ γ)|
11:                 Ê ← q/s
12:             end if
13:         end for
14:     until TERMINATE(Ê)
15:     return Ê
16: end function
```

Algorithm 7 uses a couple of auxiliary functions: RANDOMINPUTS assigns random values to all inputs and INFEROUTPUTS computes all outputs from the system model, all inputs and a diagnosis.[2] The computation of the intersection

[2]This is not always possible in the general case. In our framework, we have a number of

size $|\Omega^{\cap}(D, \alpha \wedge \beta \wedge \gamma)|$ can be implemented by counting those $\omega \in D$ for which $\text{SD} \wedge \alpha \wedge \beta \wedge \gamma \wedge \omega \not\models \bot$.

The algorithm terminates when a termination criterion (checked by TERMINATE) is satisfied. In our implementation TERMINATE returns success when the last n iterations (where n is a small constant) leave the expected number of diagnoses, $\hat{E}$, unchanged, in terms of its integer representation. Our experiments show that for all problems considered, $n = 100$ yields a negligible error.

The complexity of Alg. 7 is determined by the complexity of consistency checking (line 9 – 10) and the size of D. If we denote the complexity of a single consistency check with Υ, then the complexity of Alg. 7 becomes $O(|D|\Upsilon)$. Although, consistency checking for diagnostic problems is *NP*-hard in the worst case, for average-case problems, it is easy. In our implementation of EXPECTATION we overcome the complexity of consistency checking by using an incomplete Logic-Based Truth Maintenance System (LTMS) [Forbus and de Kleer, 1993].

5.5 Algorithms for Reducing the Diagnostic Uncertainty

In this section we introduce three algorithms: FRACTAL$^{\text{A}}$, FRACTAL$^{\text{G}}$, and FRACTAL$^{\text{P}}$. FRACTAL$^{\text{A}}$ and FRACTAL$^{\text{G}}$ differ in the way they compute subsequent control assignments. FRACTAL$^{\text{A}}$ computes an optimal "pivoting" component and uses ATPG for constructing a control assignment that tests this component and FRACTAL$^{\text{G}}$ is a greedy algorithm. FRACTAL$^{\text{P}}$ is an adaptation for our framework of the well-known probing algorithm in GDE [de Kleer and Williams, 1987] and it differs from FRACTAL$^{\text{A}}$ and FRACTAL$^{\text{G}}$ in that it measures unobserved internal variables as opposed to modifying a control assignment.

5.5.1 Problem Definition and Exhaustive Search

Our AT problem is defined as follows:

Problem 4 (Optimal Control Sequence)**.** *Given a system* ATS*, a sequence (of past observations and controls)* $S = \langle \alpha_1 \wedge \gamma_1,\ \alpha_2 \wedge \gamma_2,\ \cdots,\ \alpha_k \wedge \gamma_k \rangle$*, where* α_i *(*$1 \leq i \leq k$*) are* OBS *assignments and* γ_j *(*$1 \leq j \leq k$*) are* CTL *assignments, compute a new* CTL *assignment* γ_{k+1}*, such that:*

$$\gamma_{k+1} = \underset{\gamma \in \text{CTL}^{\star}}{\operatorname{argmin}} E^{\leq}(\Omega^{\cap}(\text{SD}, S), \{\text{IN} \cup \text{OUT}\}|\gamma) \tag{5.6}$$

where CTL* *is the space of all possible control assignments.*

Problem 4, as formulated above, implies a myopic, one-step look-ahead strategy for constructing a test tree, similar to the probing approach of de Kleer and Williams [1987] (cf. Sec. 5.6.3). Using a multi-step look-ahead (equivalent to

assumptions, i.e., a weak-fault model, well-formed circuit, etc. The complexity of INFEROUTPUTS varies on the framework and the assumptions.

k-step probing) is computationally less efficient and does not lead to significantly faster diagnostic convergence [de Kleer et al., 1992b].

Problem 4 is different from the sequential testing problem in the general case, as formulated by Shakeri [1996]. In the latter formulation, there are different test costs and different prior failure probabilities where Problem 4 implies equal costs and equal small prior probabilities of failure. Pattipati and Alexandridis [1990] show that under those assumptions, minimizing the test cost at each step is an optimal policy for minimizing the expected test cost. Hence, solving Problem 4 is solving the lesser problem of generating an optimal test strategy given unit costs and equal prior failure probability. Note that we can use an algorithm that optimizes Problem 4 as a heuristic algorithm for solving the sequential testing problem. In this case the expected cost would be arbitrarily far from the optimum one, depending on the cost distribution and the tests.

Consider our running example with an initial observation vector (and control assignment) $\alpha_3 \wedge \gamma_3 = a \wedge b \wedge i \wedge o_1 \wedge \neg o_2 \wedge \neg o_3 \wedge \neg o_4$, where $\gamma_3 = i$ is chosen as the initial control input. The four MC diagnoses of SD$\wedge\alpha_3\wedge\gamma_3$ are $\Omega^{\leq} = \{\{\neg h_1, \neg h_3\}, \{\neg h_2, \neg h_5\}, \{\neg h_4, \neg h_5\}, \{\neg h_5, \neg h_8\}\}$.

An exhaustive algorithm would compute the expected number of diagnoses for each of the $2^{|\mathrm{CTL}|}$ next possible control assignments. In our running example we have one control variable i and two possible control assignments ($\gamma_5 = i$ and $\gamma_6 = \neg i$). To compute the expected number of diagnoses, for each possible control assignment γ and for each possible observation vector α, we have to count the number of initial diagnoses which are consistent with $\alpha \wedge \gamma$.

Computing the intersection sizes for our running example gives us Table 5.2. Note that, in order to save space, Table 5.2 contains rows for those $\alpha \wedge \gamma$ only, for which $\Pr(\alpha \wedge \gamma) \neq 0$, given the initial diagnoses $\Omega^{\leq}$ (and, as a result, $\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha_3 \wedge \gamma_3), \alpha \wedge \gamma) \neq 0$). It is straightforward to compute the expected number of diagnoses for any control assignment with the help of this marginalization table. In order to do this we have to (1) filter out those lines which are consistent with the control assignment γ and (2) compute the sum and the sum of the squares of the intersection sizes (the rightmost column of Table 5.2).

To compute $E(\mathrm{SD}, \mathrm{OBS}|\alpha_3 \wedge \neg i)$, for example, we have to find the sum and the sum of the squares of the intersection sizes of all rows in Table 5.2 for which column i is **F**. It can be checked that $E(\mathrm{SD}, \mathrm{OBS}|\alpha_3, \neg i) = 24/16 = 1.5$. Similarly, $E(\mathrm{SD}, \mathrm{OBS}|\alpha_3 \wedge i) = 34/16 = 2.125$. Hence an optimal diagnostician would consider a second measurement with control setting $\gamma = i$.

The obvious problem with the above brute-force approach is that the size of the marginalization table is, in the worst-case, exponential in |OBS|. Although many of the rows in the marginalization table can be skipped as the intersections are empty (there are no consistent prior diagnoses with the respective observation vector and control assignment), the construction of this table is computationally so demanding that we will consider an approximation algorithm (to construct Table 7 for our tiny example, the exhaustive approach had to perform a total of 512 consistency checks).

Table 5.2: *Marginalization table for* SD *and* α_3

i	a	b	o_1	o_2	o_3	o_4	Pr	$\lvert\Omega^{\cap}\rvert$	i	a	b	o_1	o_2	o_3	o_4	Pr	$\lvert\Omega^{\cap}\rvert$
F	**F**	**F**	**F**	**F**	**F**	**F**	0.03125	1	**F**	**T**	**T**	**T**	**F**	**F**	**T**	0.03125	1
F	**F**	**F**	**T**	**F**	**F**	**F**	0.0625	2	**T**	**F**	**F**	**F**	**F**	**F**	**T**	0.0625	2
F	**F**	**F**	**T**	**F**	**F**	**T**	0.03125	1	**T**	**F**	**F**	**F**	**F**	**T**	**F**	0.03125	1
F	**F**	**T**	**F**	**F**	**F**	**F**	0.03125	1	**T**	**F**	**F**	**F**	**T**	**F**	**F**	0.03125	1
F	**F**	**T**	**T**	**F**	**F**	**F**	0.0625	2	**T**	**F**	**T**	**F**	**T**	**F**	**F**	0.03125	1
F	**F**	**T**	**T**	**F**	**F**	**T**	0.03125	1	**T**	**F**	**T**	**T**	**F**	**F**	**F**	0.03125	1
F	**T**	**F**	**F**	**F**	**F**	**F**	0.03125	1	**T**	**F**	**T**	**T**	**F**	**T**	**T**	0.0625	2
F	**T**	**F**	**T**	**F**	**F**	**F**	0.0625	2	**T**	**T**	**F**	**F**	**F**	**T**	**F**	0.03125	1
F	**T**	**F**	**T**	**F**	**F**	**T**	0.03125	1	**T**	**T**	**F**	**T**	**F**	**F**	**F**	0.03125	1
F	**T**	**T**	**F**	**F**	**F**	**F**	0.03125	1	**T**	**T**	**F**	**T**	**T**	**F**	**T**	0.0625	2
F	**T**	**T**	**T**	**F**	**F**	**F**	0.0625	2	**T**	**T**	**T**	**T**	**F**	**F**	**F**	0.125	4

5.5.2 FRACTAL^{A}

Consider the running example from Sec. 5.3 and an observation $\alpha_4 = a \wedge b \wedge i \wedge o_1 \wedge \neg o_4$. This leads to the 6 double-fault MC diagnoses, shown in Fig. 5.3.

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
ω_1	✓	✗	✓	✗	✗	✗	✗	✗
ω_2	✓	✗	✗	✗	✓	✗	✗	✗
ω_3	✗	✓	✗	✗	✓	✗	✗	✗
ω_4	✗	✗	✓	✗	✓	✗	✗	✗
ω_5	✗	✗	✗	✓	✓	✗	✗	✗
ω_6	✗	✗	✗	✗	✓	✗	✗	✓
$E^{\leq}$	$\frac{10}{3}$	$\frac{13}{3}$	$\frac{10}{3}$	$\frac{13}{3}$	$\frac{13}{3}$	6	6	$\frac{13}{3}$

Figure 5.3: *ATPG-Based active testing example*

Instead of searching through the space of all possible control assignments, we directly compute a control assignment that tests a specific component c by using an approach from ATPG. We choose this component c to be the one that most decreases the expected number of remaining MC diagnoses by minimizing $E^{\leq}(\mathrm{SD}, c|\alpha \wedge \gamma)$. If we look at Fig. 5.3 we can see that knowing the health of h_1 and h_3 leads to $E^{\leq} \approx 3.33$, for h_2, h_4, h_5, and h_7, we have $E^{\leq} \approx 4.33$, and for h_6 and h_7 we have $E^{\leq} = 6$. Choosing a control setting that computes the state of h_1 or h_3 is intuitive as the state of this component makes the most balanced partition of the prior diagnoses.

We next present the FRACTAL$^{\text{A}}$ algorithm that uses the approach illustrated above.

Algorithm 8 ATPG-Based active testing algorithm

```
1: function FRACTAL^ATPG(ATS, α, γ) returns a control term
       inputs: ATS, active testing system
               α, term, initial (non-modifiable) observation
               γ, term, initial control
       local variables: c, component variable
                        f, integer
                        d, term, diagnosis
                        γ, term, control setting
                        H, set of pairs, component/expectation pairs
                        D, set of terms, diagnoses
2:     D ← Ω^≤(SD, α ∧ γ)
3:     for all c ∈ COMPS do
4:         f ← 0
5:         for all d ∈ D do
6:             if c ∈ d then
7:                 f ← f + 1
8:             end if
9:         end for
10:        H ← ⟨c, f^2 + (|D| − f)^2⟩
11:    end for
12:    H ← SORTBYEXPECTATION(H)
13:    for i = 1 ... |H| do
14:        if γ ← ATPG(ATS, α, H_i⟨c⟩) then
15:            return γ
16:        end if
17:    end for
18:    return RANDOMCONTROLS()
19: end function
```

Algorithm 8 counts in how many prior diagnoses each component appears (lines 4 - 8) and the result is saved in the variable f. This number is then used to compute the expected number of remaining MC diagnoses given the component health (line 10). For each component the expected number of diagnoses is stored in the set H (line 10). The set H is then sorted in increasing order of expectation (line 12). We then iterate over the set of components in order of expectation (lines 13 – 17). For each component we try to compute an ATPG vector that tests it. In some cases such vector may not exist. In the worst case there is no ATPG vector that can test any component and Alg. 8 has no better strategy but to return a random control assignment (line 18).

The time complexity of Alg. 8 is determined by the complexity of the diagnostic search (line 2) and the complexity of ATPG (line 14). If we denote the former with Ψ and the latter with Φ then the complexity of FRACTAL$^{\text{A}}$ becomes $O(\Phi\Psi|\text{COMPS}|)$. As the complexity of ATPG is usually lower than that of diagnosis (abductive reasoning) ($\Phi < \Psi$), then the complexity of FRACTAL$^{\text{A}}$ is determined by the time for computing MC diagnoses.

Computing ATPG vectors has been extensively studied [Bushnell and Agrawal, 2000] and although it is know to be an *NP*-hard problem [Ibarra and Sahni, 1975], there exists evidence that ATPG is easy for practical problems [Prasad et al., 1999]. Some efficient ATPG algorithms integrate randomized approach and Boolean difference [Bushnell and Agrawal, 2000]. The former approach efficiently computes test vectors for the majority of components while the latter computes test vectors for the remaining ones by using a DPLL-solver.

We implement ATPG as follows. First we duplicate the system description SD by renaming each variable $v : v \notin \{\text{IN} \cup \text{CTL}\}$ to v', thus receiving SD$'$ (SD and SD$'$ share the same input and control variables). Then we create the all healthy HA ω^0 and the single fault assignment ω^I such that ω^0 and ω^I differ only in the sign of the literal whose component we want to test. Finally, we construct the following propositional expression:

$$\Delta \equiv \alpha \wedge \text{SD} \wedge \text{SD}' \wedge \omega^0 \wedge \omega^I \wedge \left[\bigvee_{o \in \text{OUT}} o \oplus o'\right] \tag{5.7}$$

To compute an ATPG control vector we need one satisfiable solution of Δ. Note that an ATPG control vector may not exist ($\Delta \models \perp$), i.e., a component may not be testable given CTL and SD $\wedge\, \alpha$. Often there are multiple satisfying control assignments. In this case FRACTAL$^{\text{A}}$ chooses an arbitrary one. The latter does not mean that all satisfiable ATPG control vectors achieve the same uncertainty reduction. FRACTAL$^{\text{A}}$ becomes suboptimal when there is no control testing a given component, or when there are multiple controls. FRACTAL$^{\text{A}}$ becomes completely random when there are no components that can be tested with the given choice of controls.

There are two problems with FRACTAL$^{\text{A}}$. First, FRACTAL$^{\text{A}}$ assumes stationary inputs, i.e., FRACTAL$^{\text{A}}$ ignores a source of uncertainty. The non-modifiable inputs, however, can only help in the decay process, hence FRACTAL$^{\text{A}}$ is "conservative" in choosing the control assignments–a feature that leads to suboptimality. A bigger problem is that FRACTAL$^{\text{A}}$ decreases the expected number of remaining MC diagnoses by computing the exact health of one component. Here, the problem is not that FRACTAL$^{\text{A}}$ tests *one* component per step but that it tries to compute a control assignment that computes the exact state of this component. An active testing algorithm can decrease the diagnostic uncertainty by computing a probability distribution function for the state of each component.

A natural extension of FRACTAL$^{\text{A}}$ is a an algorithm that computes the state of k components simultaneously. The latter approach assumes that the system is

k-component testable–an unrealistic assumption. In our experiments we have seen that systems are often even not single-component testable. Note that computing the exact states of components is not a requirement for decreasing the diagnostic uncertainty. Instead of computing the exact state of one or more components, the algorithm shown in the next section implicitly builds a pdf for the health state of each component, and does not suffer from the problems of FRACTAL$^{\mathrm{A}}$.

5.5.3 FRACTAL$^{\mathrm{G}}$

Consider SD from the example started in Sec. 5.3, input variables IN $= \{i\}$, control variables CTL $= \{a, b\}$, initial input values $\beta = i$, and an initial observation $\alpha_3 = \beta \wedge (\neg a \wedge \neg b) \wedge (\neg o_1 \wedge o_2 \wedge o_3 \wedge o_4)$. The initial observation α_3 leads to 5 triple-fault MC diagnoses: $\Omega^{\leq}(\mathrm{SD}, \alpha_3) = \{\{\neg h_1, \neg h_4, \neg h_7\}, \{\neg h_1, \neg h_7, \neg h_8\}, \{\neg h_2, \neg h_3, \neg h_6\}, \{\neg h_2, \neg h_4, \neg h_5\}, \{\neg h_3, \neg h_6, \neg h_8\}\}$. We also write $D = \Omega^{\leq}(\mathrm{SD}, \alpha_3)$ and choose one of the faults in D to be the truly injected fault ω^* (let $\omega^* = \{\neg h_1, \neg h_7, \neg h_8\}$).

Exhaustive

k	γ_1	$E^{\leq}(\mathrm{SD}, \mathrm{IN}\|\gamma_1)$
1	$\neg a \wedge \neg b$	4.33
2	$a \wedge \neg b$	1.57
3	$\neg a \wedge b$	1.57
4	$a \wedge b$	1.33
$\lvert\Omega^{\cap}(D, \beta \wedge \gamma_1)\rvert = 2$		
k	γ_2	$E^{\leq}(\mathrm{SD}, \mathrm{IN}\|\gamma_2)$
1	$\neg a \wedge \neg b$	1.67
2	$a \wedge \neg b$	1
3	$\neg a \wedge b$	1
4	$a \wedge b$	1.67
$\lvert\Omega^{\cap}(\Omega^{\cap}(D, \beta \wedge \gamma_1), \beta \wedge \gamma_2)\rvert = 1$		

Greedy

k	γ_1	$E^{\leq}(\mathrm{SD}, \mathrm{IN}\|\gamma_1)$
1	$\neg a \wedge \neg b$	4.33
2	$\mathbf{a} \wedge \neg b$	1.57
3	$a \wedge \mathbf{b}$	1.33
$\lvert\Omega^{\cap}(D, \beta \wedge \gamma_1)\rvert = 2$		
k	γ_2	$E^{\leq}(\mathrm{SD}, \mathrm{IN}\|\gamma_2)$
1	$\neg a \wedge \neg b$	1.67
2	$\neg a \wedge \mathbf{b}$	1
3	$a \wedge b$	1.67
$\lvert\Omega^{\cap}(\Omega^{\cap}(D, \beta \wedge \gamma_1), \beta \wedge \gamma_2)\rvert = 1$		

Figure 5.4: *Exhaustive and greedy search for an optimal control assignment*

The left and right parts of Fig. 5.4 show two possible scenarios for locating ω^*. On the left we have an exhaustive approach which considers all the $2^{|\mathrm{CTL}|}$ control assignments, hence it cannot be used to solve practical problems. The greedy scenario on the right side of Fig. 5.4 decreases the number of computations of expected number of remaining MC diagnoses from $2^{|\mathrm{CTL}|}$ to $|\mathrm{CTL}|$. The idea is to flip one control variable at a time, to compute the expected number of remaining MC diagnoses and to keep the flip (shown in bold in Fig. 5.4) if $E^{\leq}$ decreases. Given an initial control assignment γ we consider the space of possible

control flips. This space can be visualized as a lattice (Fig. 5.5 shows a small example). Figure 5.5 shows the expected number of MC diagnoses for each control assignment. Note that probing can be visualized in a similar way.

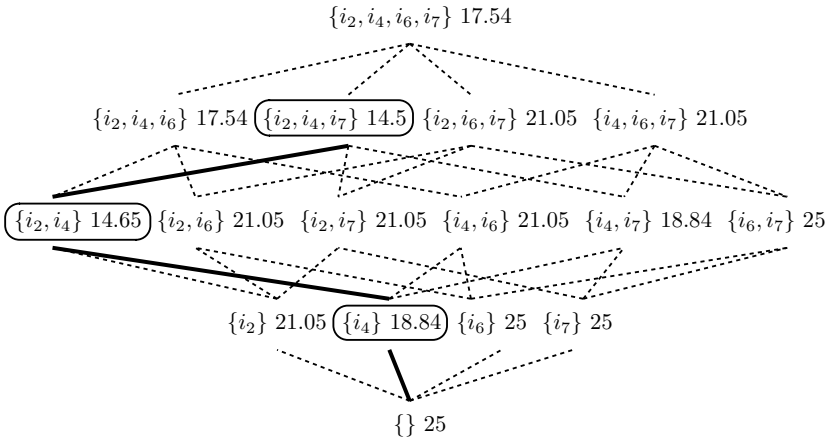


Figure 5.5: *Example of an expectation optimization lattice (*74182, $|\text{CTL}| = 4, |\text{IN}| = 5$*)*

In practice, control literals are mostly independent and even though the space of control assignments is not continuous in general, it has large continuous subspaces. The greedy approach is shown in Alg. 9, which computes a control assignment for a given active testing system and a prior observation.
The set of initial diagnoses is computed from the initial observation in line 2. In line 5, Alg. 9 "flips" the next literal in the current control assignment. The auxiliary FLIPLITERAL subroutine simply changes the sign of a specified literal in a term. After each "flip" the expected intersection size is computed with a call to EXPECTATION (cf. Alg. 7). If the new expected intersection size is smaller than the current one, then the proposed control assignment is accepted as the current control assignment, and the search continues from there.

The complexity of FRACTAL$^{\text{G}}$ is determined by the complexity of the diagnostic search (line 2) and the complexity of EXPECTATION (line 3 and line 6). If we denote the former with Ψ and the latter with Ξ then the complexity of FRACTAL$^{\text{G}}$ becomes $O(\Phi\Xi|\text{CTL}|)$. In practice FRACTAL$^{\text{G}}$ requires more computation than FRACTAL$^{\text{A}}$ to compute a sufficient decay. This is due to the design of EXPECTATION (Alg. 7).

While the active-testing problem is worst-case *NP*-hard (it can be reduced to computing a diagnosis), as we will see in the experimentation section, it is possible to achieve very good average-case performance by choosing an appropriate MBD oracle. The advantage of the greedy approach, in particular, is that the number of computations of the expected number of diagnoses is linear in the number of literals in the control assignment. This is done at the price of some optimality

Algorithm 9 Greedy active testing algorithm

```
1: function FRACTAL(ATS, α) returns a control term
       inputs: ATS, active testing system
               α, term, initial observation
       local variables: γ, γ′, terms, control configurations
                        E, E′, reals, expectations
                        D, set of terms, diagnoses
                        l, literal, control literal
2:     D ← Ω^≤(SD, α)
3:     E ← EXPECTATION(ATS, γ, D)
4:     for all l ∈ γ do
5:         γ′ ← FLIPLITERAL(γ, l)
6:         E′ ← EXPECTATION(ATS, γ′, D)
7:         if E′ < E then
8:             γ ← γ′
9:             E ← E′
10:        end if
11:    end for
12:    return γ
13: end function
```

(i.e., the effect of combinations of controls is neglected).

5.5.4 FRACTAL^{P}

Probing is related to active testing as measuring internal variables can be thought of as revealing internal control circuits. Alternatively, one can add control circuitry to a model that reveals the values of internal variables. To reveal this hidden control potential we implement GDE probing [de Kleer and Williams, 1987] in FRACTAL^{P}. Our approach is different from GDE in two ways. First, we compute the expected number of remaining MC diagnoses instead of expected diagnostic entropy. Second, FRACTAL does not use an Assumption-Based Truth Maintenance System (ATMS) [de Kleer, 1986b].

Consider the running example from Sec. 5.3 and an observation $\alpha_5 = \neg a \wedge \neg b \wedge i \wedge \neg o_1 \wedge o_2 \wedge o_3 \wedge o_4$. This leads to 5 triple-fault MC diagnoses: $\Omega^{\leq}(\mathrm{SD}, \alpha_3) = \{\{\neg h_1, \neg h_4, \neg h_7\}, \{\neg h_1, \neg h_7, \neg h_8\}, \{\neg h_2, \neg h_3, \neg h_6\}, \{\neg h_2, \neg h_4, \neg h_5\}, \{\neg h_3, \neg h_6, \neg h_8\}\}$. Subsequent measurement of p gives us $|\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha_3), p)| = 3$ if p is positive and $|\Omega^{\cap}(\Omega^{\leq}(\mathrm{SD}, \alpha_5), \neg p)| = 2$ otherwise. The expected number of MC diagnoses is $E^{\leq}(\mathrm{SD}, \{p\}|\alpha_3) = 2.6$. Repeating this for the remaining internal variables results in $E^{\leq}(\mathrm{SD}, \{q\}|\alpha_3) = 2.6$, $E^{\leq}(\mathrm{SD}, \{r\}|\alpha_3) = 3.4$, and $E^{\leq}(\mathrm{SD}, \{s\}|\alpha_3) = 3.4$. As a result we can see that measuring p and q is less informative than measuring r and s which is intuitive as r and s give a more balanced

partitioning of the circuit.

Problem 5 (Probe Sequencing)**.** *Given a system* ATS*, an observation* α *and a partial assignment to the internal variables* ψ*, choose a variable* p^* *from the set* U *of unassigned in* ψ *internal variables, such that:*

$$p^* = \underset{p \in U}{\operatorname{argmin}} E^{\leq}(\mathrm{SD}, p|\alpha \wedge \psi) \tag{5.8}$$

Algorithm 10 solves Problem 5. Algorithm 10 computes the expected number of diagnoses for each unobserved variable (lines 3 - 11). Starting from the set D of initial diagnoses (computed in line 2), Alg. 10 perform a total of $2|D||V \setminus \{\mathrm{OBS} \cup \mathrm{COMPS}\}|$ consistency checks (lines 4 and 5) to determine the expected number of MC diagnoses for each unobserved variable.

Algorithm 10 Probing algorithm

```
1:  function FRACTAL^P(ATS, α) returns a variable
        inputs: ATS, active testing system
                α, term, observation
        local variables: v, R, variables
                         E, E', reals, expectations
                         p, q, reals
                         D, set of terms, diagnoses
2:      D ← Ω^≤(SD, α)
3:      E ← ∞
4:      for all v ∈ V \ {COMPS ∪ OBS} do
5:          p ← |Ω^∩(D, α ∧ v)|
6:          q ← |Ω^∩(D, α ∧ ¬v)|
7:          E' ← (p^2 + q^2)/(p + q)
8:          if E' < E then
9:              R ← v
10:             E ← E'
11:         end if
12:     end for
13:     return R
14: end function
```

Instead of computing the expected number of remaining MC diagnoses for a single variable p, it is possible to consider measuring all pairs of variables $\langle p_1, p_2 \rangle$, or in general, all k-tuples of internal variables $\langle p_1, p_2, \ldots, p_m \rangle$ for $m \leq |V \setminus \{\mathrm{OBS} \cup \mathrm{COMPS}\}|$. We will refer to probing involving more than 1 variable as k-probing. Although it has been shown that users do not significantly benefit in terms of diagnostic uncertainty by performing k-probing [de Kleer et al., 1992b], we can easily modify FRACTAL$^{\mathrm{P}}$ to consider multiple probes. Note that for $m = |V \setminus$

$\{\text{OBS} \cup \text{COMPS}\}|$ there is no probing problem as there is only one way to pick all internal variables.

The most complex operation in FRACTAL$^{\text{P}}$ is again computing the initial set of MC diagnoses. In addition to that we have $|V \setminus \{\text{COMPS} \cup \text{OBS}\}|$ consistency checks. Consistency checking is, in general, easier than diagnosis. Note that all FRACTAL algorithms (FRACTAL$^{\text{A}}$, FRACTAL$^{\text{G}}$, and FRACTAL$^{\text{P}}$) start with computing the set of initial MC diagnoses. Hence, the difference in their performance is determined by the complexity of reducing the initial set $\Omega^{\leq}(\text{SD}, \alpha)$. According to this criterion, the fastest algorithm is FRACTAL$^{\text{P}}$ as it only performs a small number of consistency checks, followed closely by FRACTAL$^{\text{A}}$ (computing ATPG vectors). The slowest algorithm is FRACTAL$^{\text{G}}$ as it computes the expected number of MC diagnoses given multiple variables.

5.6 Experimental Results

We have implemented FRACTAL in approximately 3 000 lines of C code (excluding the diagnostic engine and the Logic Based Truth Maintenance System). All experiments were run on a 64-node dual-CPU cluster (each node configured with two 2.4 GHz AMD Opteron DP 250 processors and 4 Gb of RAM).

5.6.1 Experimental Setup

In this chapter, we have used "stuck-at-opposite" models of the benchmark circuits introduced in Sec. 2.4. In order to use the same model for *both* MC diagnosis counting[3] and simulation, the fault mode of each logic gate is "stuck-at-opposite", i.e., when faulty the output of a gate assumes the opposite value from the nominal.

In addition to the original 74XXX/ISCAS85 models (cf. Sec. 2.4), we have performed cone reductions as described in Sec. 4.4. We call models with a single health variable per cone "reduced".

Both initial observation vectors and control settings are used in the first step of the FRACTAL inference. To illustrate the significant diagnostic convergence that is possible, we use initial observations leading to high numbers of initial MC diagnoses. To average over the diagnostic outcomes of the observations, we repeat each experiment with a range of initial observation vectors. We have limited our experiments to observation vectors leading to double faults only (the cardinality of the MC diagnosis is of no significance to FRACTAL but it produces a significant burden on the diagnostic oracle [Feldman et al., 2008b]).

For each circuit we have generated 1 000 non-masking double-faults and for each observation we have computed the number of initial MC diagnoses. From the 1 000 observation vectors we have taken the 100 with the largest number of MC diagnoses. The resulting observations are summarized in Table 5.3. For

[3] Recall that counting the number of MC diagnoses of $\text{SD} \wedge \alpha$ is equivalent to computing $|\Omega^{\leq}(\text{SD}, \alpha)|$.

example, we can see a staggering number of 46 003 double faults for the most under-constrained c7552 observation.

Table 5.3: *Number of MC diagnoses per observation vector*

	Original			Reduced		
Name	Min	Max	Mean	Min	Max	Mean
74182	25	25	25	2	2	2
74L85	32	88	50.2	5	13	7
74283	48	60	51.8	6	9	7
74181	93	175	113.5	10	19	13.3
c432	88	370	165.8	21	127	47.5
c499	168	292	214.1	4	31	15.8
c880	783	1 944	1 032.5	20	190	39.7
c1355	1 200	1 996	1 623.3	4	31	15
c1908	1 782	5 614	2 321.9	40	341	84.8
c2670	2 747	7 275	3 621.7	10	90	21.3
c3540	1 364	2 650	1 642.2	158	576	226
c5315	3 312	17 423	6 202.1	15	192	34.5
c6288	6 246	15 795	8 526.1	2 928	6 811	3 853
c7552	16 617	46 003	23 641.2	45	624	121.6

Since the 74XXX/ISCAS85 circuits have no control variables we "abuse" the benchmark by designating a fraction of the input variables as controls.

We define two policies for generating next inputs: *random* and *stationary*. The latter input policy (where the input values do not change in time) is a typical diagnostic worst-case for system environments which are, for example, paused pending diagnostic investigation, and it provides us with useful bounds for analyzing FRACTAL performance.

5.6.2 Expected Number of MC Diagnoses

We have observed that the error of Alg. 7 is not sensitive to the number or the composition of the input variables. It can be seen that the value of the expected number of diagnoses $\hat{E}$ approaches the exact value E when increasing the number of samples n. In particular, $\hat{E}$ is equal to the exact value of the expected number of MC diagnoses E, when all possible input values are considered. Figure 5.6 shows examples of $\hat{E}$ approaching E for three of our benchmark models.

TERMINATE approximates the intermediate value of $\hat{E}$ by computing the sequence $\mathbf{E} = \langle \hat{E}_1, \hat{E}_2, \ldots, \hat{E}_n \rangle$. The standard error of the mean of $\mathbf{E}$ is defined

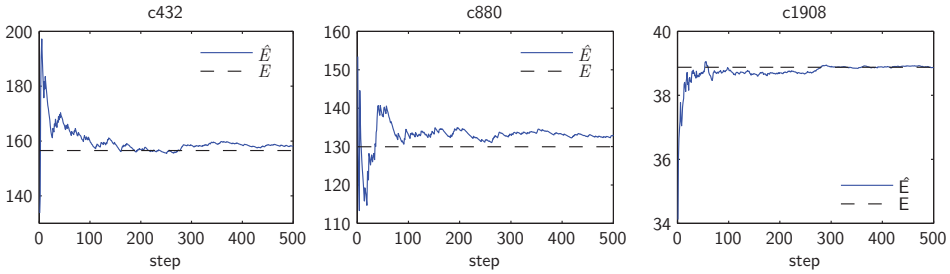


Figure 5.6: *Convergence of expected number of MC diagnoses with increasing sample size*

as:

$$\mathrm{SEM}_{\mathbf{E}} = \frac{s}{\sqrt{n}}, \tag{5.9}$$

where s is the standard deviation of $\mathbf{E}$. We have set TERMINATE to terminate Alg. 7 when $n > 15$ and $\mathrm{SEM}_{\mathbf{E}} < \theta$, where θ is a circuit-dependent threshold constant. Table 5.4 shows θ for the various circuits we have experimented on.

Table 5.4: *Termination parameters for Alg. 7*

	Original			Reduced		
Name	θ	Mean n	Max n	θ	Mean n	Max n
74182	0.1	52.7	110	0.01	151.6	223
74L85	0.11	176.2	246	0.03	170.3	212
74283	0.2	139.9	225	0.01	211.3	243
74181	0.4	169.1	203	0.07	143.2	181
c432	0.72	48.2	99	0.18	108.6	158
c499	0.77	36.4	61	0.02	55.7	92
c880	3.57	93.9	163	0.1	156.3	204
c1355	4.3	51.3	93	0.01	121.4	148
c1908	14.01	19.5	35	0.62	18.8	25
c2670	12.77	40.5	78	0.1	65.2	102
c3540	13.6	78.9	196	0.66	89.6	132
c5315	23.35	34.2	39	0.09	36.0	48
c6288	33.18	37.2	144	19.1	39.4	74
c7552	68.11	68.7	91	3.73	73	122

We have determined θ using the following procedure. First, for each circuit, we choose an arbitrary initial observation and a small set IN of input variables

(|IN| = 8). The small cardinality of IN allows us to compute true values of E. Next, for each circuit we run 10 pseudo-random experiments. For θ we choose the smallest value of $\text{SEM}_{\mathbf{E}}$ such that its corresponding $\hat{E}$ is within 95% of E. Table 5.4 shows the average and maximum number of steps in which Alg. 7 reaches this value. In all cases $n = O(250)$ is a safe termination criterion.

5.6.3 Comparison of Algorithms

Consider a weak-fault model of a chain of n inverters and a set of MC diagnoses D (initially, $|D| = n$). At each step single-variable probing can eliminate at most $0.5|D|$ diagnoses. It can be also shown that halving the expected number of remaining MC diagnoses is a theoretical bound of any one-step lookahead strategy. As a result we use the geometric decay curve

$$N(k) = N_0 p^k + N_\infty \tag{5.10}$$

as a model of the diagnosis decay. In this case, N_0 is the initial number of diagnoses, N_∞ is the value to which $|\Omega(S)|$ converges, and p is the decay rate constant. For probing, it can be shown that $N_\infty = 1$. In all our experiments we will fit both the expected number of remaining MC diagnoses E and the actual number or remaining MC diagnoses $\Omega(S)$ to (5.10).

FRACTAL^A

Figure 5.7 shows the reduction of the expected number of MC diagnoses as a function of (1) the number of control variables |CTL| and (2) the time k. One can easily see that a global optimum is reached quickly on both independent axes. This decay is shown for both c432 (Fig. 5.7, left) and the reduced c880 (Fig. 5.7, right). The number of control variables |CTL| varies from 0 to 36 for c432 (|IN| = 36) and from 0 to 60 for c880 (|IN| = 60).

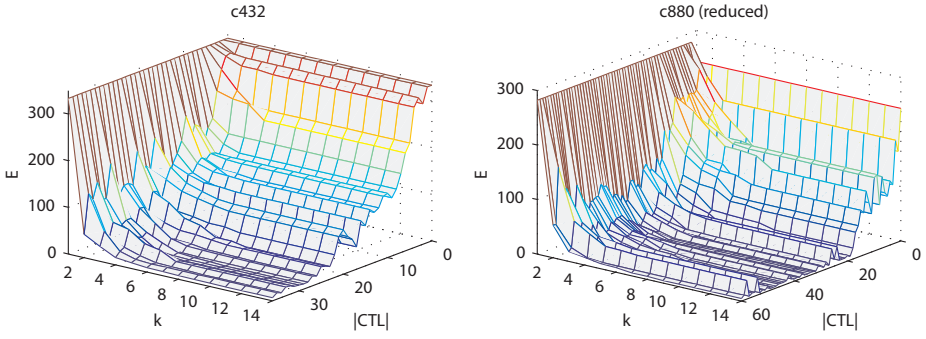


Figure 5.7: *Decay of E, stationary inputs,* FRACTAL^A

Using $|\Omega(S)|$ instead of E results in similar plots (there is high correlation between E and $|\Omega(S)|$), hence we have omitted the $|\Omega(S)|$ plots. The minimum, maximum and mean Pearson's linear correlation coefficient between E from Fig. 5.7 and the respective $|\Omega(S)|$ for each number of control variables in c432 is $\rho_{\min} = 0.713$, $\rho_{\max} = 0.999$, and $\rho_{\text{avg}} = 0.951$, respectively. The corresponding correlation coefficients for the reduced c880 are $\rho_{\min} = 0.834$, $\rho_{\max} = 1$, and $\rho_{\text{avg}} = 0.972$.

It can be seen that E quickly reaches a global optimum when increasing $|\text{CTL}|$ which means that turning even a small number of input variables into controls allows for a geometric decay of the diagnostic entropy. The results for the reduced c880 are similar to the non-reduced c432. Hence, identification of cones helps the performance of the diagnostic oracle, but does not change the convergence behavior or the effect of the control variables.

Fitting geometric decay curves (5.10) on the $|\text{CTL}|$ axes of Fig. 5.7 produces better fits for c880 than for c432. Similarly, the values of N_∞ for fits alongside the k-axis are larger for c432 than for c880. The reason for that is the small number of outputs in c432 (cf. Table 2.2). In circuits with small number of outputs, randomly turning a limited number of inputs into controls may not lead to a fast decay or a small N_∞ as the control-output connectivity of a model is essential for decreasing the diagnostic uncertainty.

Table 5.5 and Table 5.6 summarize a total of 14 000 FRACTAL$^{\text{A}}$ experiments over the whole 74XXX/ISCAS85 benchmark. Table 5.5 shows the correlation between the expected number of remaining MC diagnoses and the actual number of remaining MC diagnoses. In the second and third columns of Table 5.5 we can see the minimum and average correlations between E and $\Omega^{\leq}(S)$. The third and fourth cases specify the fraction of observations for which we have $\rho > 0.95$ and $\rho > 0.975$, respectively. Columns 6 – 9 repeat this data for the reduced 74XXX/ISCAS85 circuits.

Table 5.6 summarizes the parameters of the geometric decay curves fitted to $\Omega(S)$. We can see that although $\Omega(S)$ is well approximated by a geometric decay curve (the average goodness-of-fit criterion R^2 is 0.84) the average decay constant p is low (0.13 for the non-reduced and 0.22 for the reduced 74XXX/ISCAS85 circuits).

The decay rate p depends mostly on the circuit topology, hence the large variance in Table 5.6. Consider, for example, an artificial topology, where there are n components, and n output variables that produce the health-state of each component for a specific control assignment (e.g., a self-test). In this topology p would be very small as a diagnostician needs at most one test (control assignment) to decrease the number of MC diagnoses to one.

The performance of FRACTAL$^{\text{A}}$ is determined by the size of the model and the diagnostic oracle. In the above experiments the overall time for executing a single scenario varied from 3.4 s for 74182 to 1 015 s for c6288. The satisfiability problems in the ATPG part were always easy and the DPLL solver spent millisecond in computing control assignments.

The decay rate of FRACTAL^{A} depends on the number and composition of controls. In what follows we will see that FRACTAL^{G} can achieve similar decay rate with a smaller number of control variables.

FRACTAL^{G}

Figure 5.8 shows the decay in the expected number of remaining MC diagnoses for FRACTAL^{G}. While the reduction is similar for c432, we can see a steeper reduction in the number of remaining MC diagnoses on both independent axes. Hence, the greedy algorithm is better than FRACTAL^{A} in identifying control combinations of small size that lead to better decay rate.

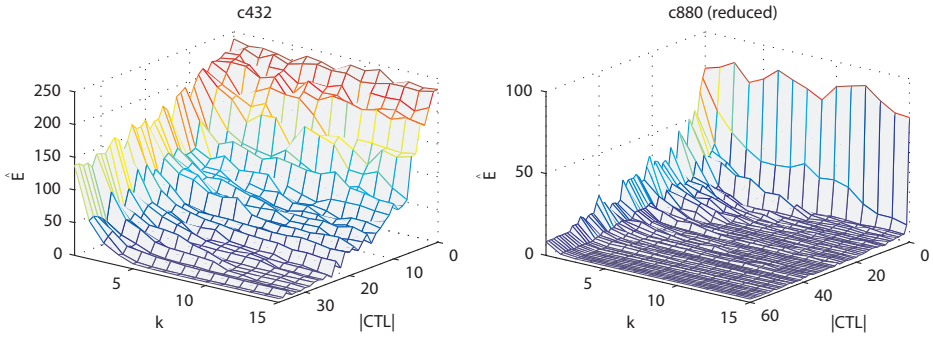


Figure 5.8: *Decay of E (left) and $\Omega(S)$ (right), stationary inputs,* FRACTAL^{G}

Table 5.7 and Table 5.8 summarize the whole 74XXX/ISCAS85 benchmark. Table 5.7 shows that FRACTAL^{G}, similar to FRACTAL^{A}, results in high average correlation between $\Omega(S)$ and $\hat{E}$ ($\rho_{\text{avg}} > 0.79$ for all circuits).

The decay rates of FRACTAL^{A} and FRACTAL^{G} are similar (cf. Table 5.6 and Table 5.8), but as it is visible from Fig. 5.8, FRACTAL^{G} reduces faster the number of remaining MC diagnoses with smaller number of control variables. The c432 combinational circuit is difficult for active testing because it has a small number of outputs compared to the number of inputs (cf. Table 2.2), hence reducing the diagnostic utility.

To summarize the effect of the number of controls on the diagnostic convergence, we again fit the geometric decay curve (5.10) to $\Omega(S)$ for each of the 100 initial observation vectors and various |CTL|. In this case, N_0 is the initial number of diagnoses, N_∞ is the value to which $|\Omega(S)|$ converges, and p is the decay constant (the most important parameter of our fits). For an "easy" circuit with chain topology, for $p = \frac{1}{2}$, N_0 halves every k steps, like in binary search, hence p corresponds to one bit. For $p = \frac{1}{4}$, p corresponds to two bits, etc. Table 5.9 shows the average p over all initial observations and for various numbers of control bits $b = \lg |\text{CTL}|$.

Table 5.5: *Correlation coefficient ρ of the expected number E and the actual number $\Omega^{\leq}(S)$ of remaining MC diagnoses, $|\mathrm{CTL}| = \frac{1}{4}|\mathrm{IN}|$, stationary inputs,* FRACTAL$^{\mathrm{A}}$

	Original				Reduced			
Name	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$
74182	0.55	0.98	0.82	0.79	1	1	1	1
74L85	0.46	0.91	0.52	0.44	0.45	0.81	0.4	0.39
74283	0.46	0.91	0.69	0.61	0.45	0.84	0.38	0.31
74181	0.46	0.88	0.48	0.39	0.45	0.86	0.44	0.35
c432	0	0.83	0.24	0.16	0	0.81	0.29	0.23
c499	0.5	0.87	0.32	0.15	0	0.86	0.42	0.33
c880	0.51	0.86	0.28	0.18	0.51	0.85	0.3	0.19
c1355	0	0.88	0.32	0.18	0	0.87	0.47	0.34
c1908	0.38	0.87	0.31	0.18	0	0.79	0.22	0.15
c2670	0	0.89	0.31	0.16	0	0.79	0.19	0.12
c3540	0.48	0.86	0.29	0.19	0.06	0.82	0.3	0.23
c5315	0.54	0.93	0.48	0.39	0.47	0.88	0.44	0.32
c6288	0.42	0.9	0.41	0.24	0.4	0.9	0.36	0.21
c7552	0.62	0.88	0.44	0.13	0.45	0.89	0.43	0.23

Table 5.6: *Decay rate p (minimal, maximal, and average) and average goodness-of-fit R^2 of geometric decay best-fit to $\Omega(S)$, $|\mathrm{CTL}| = \frac{1}{4}|\mathrm{IN}|$, stationary inputs,* FRACTAL$^{\mathrm{A}}$

	Original				Reduced			
Name	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}
74182	0.3	0.52	0.43	0.95	0.5	0.5	0.5	1
74L85	0.06	0.75	0.48	0.88	0.35	0.64	0.5	0.92
74283	0.18	0.68	0.57	0.78	0.31	0.6	0.48	0.94
74181	0.11	0.71	0.5	0.86	0.18	0.64	0.5	0.9
c432	0.03	0.8	0.56	0.81	0.03	0.79	0.52	0.82
c499	0.1	0.79	0.64	0.84	0.48	0.76	0.65	0.84
c880	0.06	0.84	0.54	0.83	0.06	0.8	0.53	0.87
c1355	0.03	0.9	0.62	0.81	0.39	0.76	0.63	0.85
c1908	0.02	0.93	0.65	0.68	0.51	0.74	0.64	0.77
c2670	0.05	0.91	0.63	0.77	0.14	0.75	0.6	0.8
c3540	0.02	0.87	0.52	0.85	0.04	0.76	0.45	0.89
c5315	0.22	0.91	0.65	0.8	0.06	0.79	0.54	0.84
c6288	0.02	0.91	0.52	0.88	0.02	0.9	0.51	0.89
c7552	0.56	0.95	0.76	0.61	0.01	0.88	0.58	0.85
Average	0.13	0.82	0.58	0.81	0.22	0.74	0.55	0.87

Table 5.7: *Correlation coefficient ρ of the expected number E and the actual number $\Omega^{\leq}(S)$ of remaining MC diagnoses, $|\mathrm{CTL}| = \frac{1}{4}|\mathrm{IN}|$, stationary inputs,* FRACTAL$^{\mathrm{G}}$

	Original				Reduced			
Name	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$
74182	0.03	0.88	0.39	0.18	1	1	1	1
74L85	0	0.72	0.12	0.06	0.01	0.66	0.16	0.14
74283	0	0.5	0.08	0.03	0	0.48	0.12	0.11
74181	0	0.56	0.05	0.02	0	0.55	0.09	0.07
c432	0.01	0.75	0.07	0.02	0.01	0.68	0.07	0.05
c499	0.01	0.88	0.29	0.08	0	0.85	0.33	0.2
c880	0.05	0.77	0.09	0.06	0	0.73	0.08	0.04
c1355	0.08	0.86	0.36	0.21	0.42	0.9	0.39	0.16
c1908	0.05	0.81	0.25	0.14	0	0.8	0.4	0.3
c2670	0.01	0.83	0.38	0.22	0.01	0.76	0.37	0.26
c3540	0.34	0.73	0.09	0.05	0	0.7	0.04	0.01
c5315	0.27	0.78	0.05	0	0	0.6	0.1	0.07
c6288	0.09	0.81	0.11	0.05	0.1	0.78	0.09	0.04
c7552	0.78	0.86	0.06	0.06	0.21	0.83	0.13	0.01

Table 5.8: *Decay rate p (minimal, maximal, and average) and average goodness-of-fit R^2 of geometric decay best-fit to $\Omega(S)$, $|\mathrm{CTL}| = \frac{1}{4}|\mathrm{IN}|$, stationary inputs,* FRACTAL$^{\mathrm{G}}$

	Original				Reduced			
Name	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}
74182	0.24	0.53	0.43	0.95	0.5	0.5	0.5	1
74L85	0.05	0.74	0.47	0.9	0.25	0.65	0.49	0.93
74283	0.12	0.67	0.42	0.9	0.35	0.58	0.44	0.96
74181	0.15	0.75	0.48	0.9	0.15	0.69	0.44	0.93
c432	0.04	0.88	0.56	0.83	0.03	0.86	0.59	0.8
c499	0.09	0.88	0.71	0.81	0.34	0.85	0.68	0.85
c880	0.12	0.67	0.42	0.9	0.07	0.83	0.52	0.88
c1355	0.19	0.87	0.63	0.87	0.11	0.82	0.68	0.85
c1908	0.32	0.73	0.53	0.87	0.05	0.84	0.59	0.83
c2670	0.21	0.74	0.53	0.87	0.15	0.81	0.6	0.8
c3540	0.34	0.63	0.53	0.91	0.01	0.8	0.44	0.9
c5315	0.3	0.83	0.61	0.83	0.06	0.86	0.58	0.82
c6288	0.04	0.81	0.5	0.9	0.08	0.77	0.47	0.89
c7552	0.08	0.54	0.34	0.92	0.16	0.83	0.59	0.84
Average	0.16	0.73	0.51	0.88	0.17	0.76	0.54	0.88

Table 5.9: *Mean p (over all initial observations) for various numbers of control bits and stationary input policies,* FRACTAL^{G}

	Original			Reduced		
Name	3 bits	4 bits	5 bits	3 bits	4 bits	5 bits
c432	0.61	0.69	0.42	0.7	0.71	0.57
c499	0.79	0.83	0.77	0.58	0.62	0.52
c880	0.5	0.55	0.62	0.49	0.47	0.44
c1355	0.71	0.72	0.59	0.8	0.82	0.75
c1908	0.68	0.7	0.41	0.54	0.52	0.3
c2670	0.45	0.49	0.39	0.39	0.44	0.42
c3540	0.39	0.38	0.43	0.79	0.8	0.61
c5315	0.52	0.62	0.67	0.81	0.72	0.79
c6288	0.31	0.41	0.23	0.64	0.7	0.59
c7552	0.62	0.77	0.3	0.59	0.34	0.38

Table 5.9 does not include data for the 74XXX circuits as they do not have enough inputs (we need circuits with at least 32 inputs). From Table 5.9 it is visible that an exponential increase in the number of control variables does not lead to a significant decrease in p. Hence, for ISCAS85, even turning a small number of the input variables into controls, leads to a near-optimal decrease in the number of remaining MC diagnoses.

The performance of FRACTAL^{G} was worse than that of FRACTAL^{A} due to the multi-variable expectation. The running time varied between 7.1 s for 74182 and 2 382 s for c6288. Most of the CPU time was spent in the EXPECTATION subroutine (cf. Alg. 7). Each consistency check was computationally easy, but for each circuit there were thousands of them. Hence, improving the performance of LTMS would lead to an increase of the performance of FRACTAL^{G}.

FRACTAL^{P}

We next discuss FRACTAL^{P}. As mentioned earlier, probing is different from active testing as it assumes full observability of the model, i.e., all internal variables can be measured (cf. Sec. 5.5). Furthermore, probing considers one internal variable per step, while active testing assigns value to all control variables.[4]

The value of the decay rate p depends on (1) the topology of the circuit, (2) the initial observation and (3) the values of the subsequent probes. For probing in ISCAS85 we see that the values of the decay rate p are close to 0.5 for both $\Omega(S)$ and E. Figure 5.9 shows the actual and expected number of remaining MC diagnoses ($\Omega^{\leq}(S)$ and E, respectively) and a geometric fit to E for three probing scenarios.

[4]There exist multi-probe generalizations of probing [de Kleer et al., 1992b].

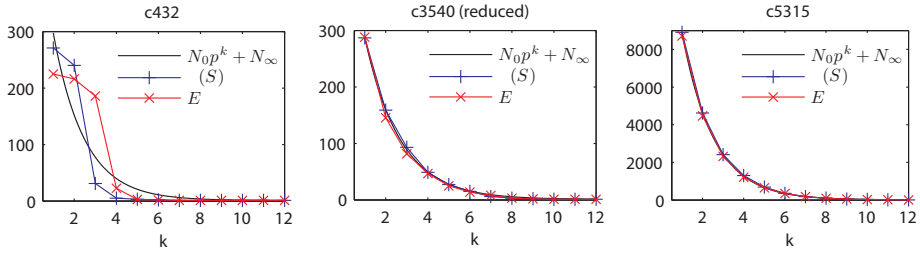


Figure 5.9: *Actual number of remaining MC diagnoses $\Omega(S)$, expected number of remaining MC diagnoses E, and a geometric decay fit to $\Omega(S)$, stationary inputs,* FRACTAL$^{\mathrm{P}}$

Each plot in Fig. 5.9 shows a single probing session with a single initial observation. Figure 5.10 shows the goodness-of-fit criterion R^2 vs. the decay rate constant p for all the 100 observations and each of the 10 multiple runs of the Fig. 5.9 circuits.

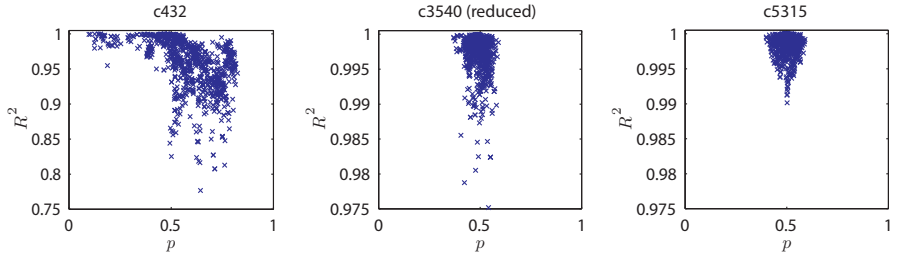


Figure 5.10: *Geometric decay rate vs. goodness-of-fit for $\Omega(S)$,* FRACTAL$^{\mathrm{P}}$

It is visible from Fig. 5.10 that the absolute values of R^2 are in most of the cases close to 1. This is an indicator that the probing experiments fit well the geometric decay model given in (5.10). Figure 5.10 shows a "bad" topology (c432 on the left) and a "good" topology (c5315 on the right) which achieves decay rate p close to 0.5 ($0.38 < p < 0.58$) with very high accuracy of the fit ($0.9896 \leq R^2 \leq 1$).

The expected number of remaining MC diagnoses is a good predictor of the actual number of MC diagnoses for all ISCAS85 circuits, as is shown in Table 5.10. The absolute values, again depend on the topology and we can see smaller correlation ρ for some c432 observations. In most of the cases, however, the correlation is significant, e.g., for all circuits and observations except c432 we have $\rho > 0.95$.

In the second and third columns of Table 5.10 we can see the minimum and average correlations between E and $\Omega^{\leq}(S)$. The third and fourth cases specify the fraction of observations for which we have $\rho > 0.95$ and $\rho > 0.975$, respectively. Columns 6 – 9 repeat this data for the reduced 74XXX/ISCAS85 circuits.

Table 5.11 summarizes the decay rate p and the goodness-of-fit criterion R^2 for

all observations and circuits. For c432, the values of p and R^2 are more dispersed, while in the other experiments p strongly resembles that of "chained-elements" (i.e., p is close to 0.5). The minimum, maximum and average values of p (per circuit) are given in columns $p_{\min}$, $p_{\max}$, and p_{avg}, respectively.

5.6.4 Experimental Summary

If we compare Table 5.5 and Table 5.10 we can see that the average correlation ρ_{avg} decreases significantly. Hence assuming limited observability (assuming that not all internals are measurable) decreases the quality of E as a predictor of $\Omega(S)$. The increased statistical dispersion of ρ is visible from the increased range $\rho_{\max} - \rho_{\min}$ (cf. Table 5.5, $\rho_{\max}$ is always 1). For example, if we consider c2670, the standard deviation of all E vs. $\Omega(S)$ correlation coefficients ρ is $\sigma_\rho = 0.0031$ for FRACTAL$^{\text{P}}$ and $\sigma_\rho = 0.0783$ for FRACTAL$^{\text{A}}$. The difference in dispersion of correlation coefficients is significant for all circuits, with smallest values for c432, where it is 0.0038 for FRACTAL$^{\text{P}}$ and 0.0825 for FRACTAL$^{\text{A}}$.

By comparing Table 5.6, Table 5.8, and Table 5.11 we can see that the mean decay rates of FRACTAL$^{\text{A}}$, FRACTAL$^{\text{G}}$, and FRACTAL$^{\text{P}}$ are similar (the average p of FRACTAL$^{\text{G}}$ is 0.7 while the average p of FRACTAL$^{\text{A}}$ is 0.73). The average goodness-of-fit criterion R^2 for exponential decays is always good (0.88 for FRACTAL$^{\text{G}}$, 0.84 for FRACTAL$^{\text{A}}$), and almost perfect in probing (0.97).

The summary of our experiments is best shown in Fig. 5.11. To factor out sampling error and to be able to perform exhaustive computations we have chosen the smallest 74182 circuit. The original 74182 (a 4-bit carry-lookahead generator) has 19 components, 9 inputs, and 5 outputs. We have turned 4 of the inputs into controls (hence, $|\text{IN}| = 4$ and $|\text{CTL}| = 4$).

We have considered a random control policy in addition to FRACTAL$^{\text{P}}$, FRACTAL$^{\text{A}}$, and FRACTAL$^{\text{G}}$. With random control policy, at each step, a random value is assigned to each control variable. We have also shown an exhaustive control search where the expected number of remaining MC diagnoses is computed at each step and for each possible control combination. This works with 74182 but blows-up with any other larger circuit.

To reduce the stochastic error when plotting Fig. 5.11 we have replaced the sampling for computing an expected number of remaining MC diagnoses with an exhaustive method (this is possible as $|\text{IN}| = 5$). The only randomized decision is to choose the actual fault from the initial ambiguity group. To reduce the error due to this stochastic fault injection, we have tested each of the 5 control policies 100 times.

We can see in Fig. 5.11 that the least informed control policy (the random control policy simply does not use E) shows the worst decay in the number of remaining diagnoses. On the other extreme, the exhaustive control policy achieves the best decay. The price for this policy in terms of computational effort, however, is prohibitive. FRACTAL$^{\text{G}}$ achieves decay rates comparable to the exhaustive policy with affordable average-case complexity. FRACTAL$^{\text{A}}$ has better complexity

Table 5.10: *Correlation coefficient ρ of the expected number of remaining MC diagnoses E and the actual number of remaining diagnoses $\Omega^{\leq}(S)$, stationary inputs,* FRACTAL$^{\mathrm{P}}$

	Original				Reduced			
Name	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$	$\rho_{\min}$	ρ_{avg}	$\rho > 0.95$	$\rho > 0.975$
74182	0.83	0.95	0.64	0.6	1	1	1	1
74L85	0.77	0.97	0.87	0.67	0.83	0.99	0.92	0.87
74283	0.97	0.99	1	1	0.83	0.98	0.83	0.76
74181	0.96	0.99	1	0.97	0.92	0.99	0.95	0.92
c432	0.66	0.97	0.83	0.67	0.62	0.96	0.76	0.62
c499	0.97	1	1	1	0.91	0.98	0.87	0.76
c880	0.98	1	1	1	0.92	0.99	0.99	0.96
c1355	0.99	1	1	1	0.86	0.98	0.88	0.79
c1908	0.98	1	1	1	0.65	0.97	0.86	0.68
c2670	0.98	1	1	1	0.7	0.96	0.72	0.55
c3540	0.97	1	1	1	0.97	1	1	1
c5315	0.99	1	1	1	0.7	0.98	0.91	0.81
c6288	0.92	1	1	1	0.98	1	1	1
c7552	0.95	1	1	0.99	0.82	0.96	0.7	0.51

Table 5.11: *Decay rate p (minimal, maximal, and average) and goodness-of-fit R^2 (average) of geometric decay best-fit to $\Omega(S)$, stationary inputs,* FRACTAL$^{\mathrm{P}}$

	Original				Reduced			
Name	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}	$p_{\min}$	$p_{\max}$	p_{avg}	R^2_{avg}
74182	0.26	0.64	0.54	0.95	0.5	0.5	0.5	1
74L85	0.21	0.7	0.52	0.97	0.25	0.55	0.45	0.97
74283	0.31	0.64	0.49	0.99	0.4	0.58	0.49	0.96
74181	0.3	0.66	0.5	0.99	0.27	0.56	0.42	0.99
c432	0.1	0.82	0.58	0.96	0.11	0.84	0.55	0.95
c499	0.4	0.57	0.5	1	0.25	0.6	0.46	0.98
c880	0.36	0.61	0.51	1	0.2	0.67	0.46	0.99
c1355	0.39	0.6	0.51	1	0.25	0.59	0.46	0.98
c1908	0.39	0.58	0.5	1	0.13	0.81	0.55	0.96
c2670	0.37	0.6	0.51	1	0.22	0.85	0.65	0.89
c3540	0.38	0.58	0.5	1	0.37	0.59	0.49	1
c5315	0.4	0.59	0.5	1	0.18	0.89	0.52	0.96
c6288	0.92	1	1	1	0.98	1	1	1
c7552	0.95	1	1	0.99	0.82	0.96	0.7	0.51
Average	0.41	0.69	0.58	0.99	0.35	0.71	0.55	0.94

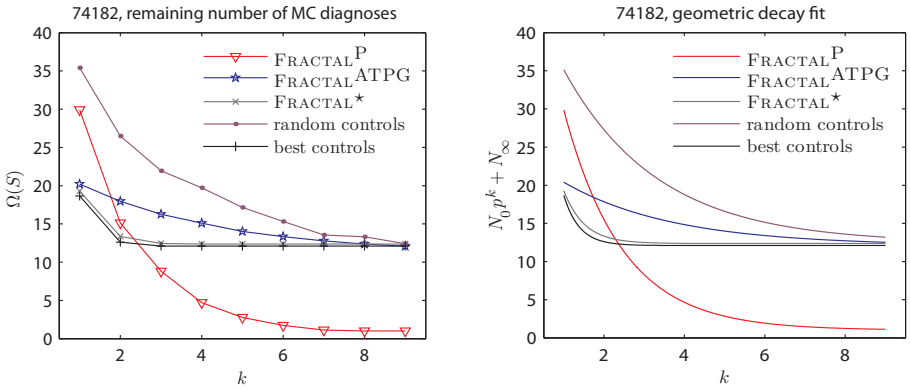


Figure 5.11: *Comparison of all control policies*

than FRACTAL$^{\mathrm{G}}$, but the whole decay rate curve of FRACTAL$^{\mathrm{A}}$ is bounded from below by the one computed by FRACTAL$^{\mathrm{G}}$.

Probing does not compare to active testing as both approaches have different assumptions on the observability of the model. Figure 5.11 shows the decay rate of probing to illustrate the different decay curves depending on the observability assumptions. In this experiment the probing decay rate geometric fit with $p = \frac{1}{2}$ almost perfectly fits the actual number of remaining MC diagnoses.

5.7 Summary

We have devised an algorithm, FRACTAL$^{\mathrm{G}}$, for active testing that is (1) computationally efficient and (2) rapidly reduces the diagnostic uncertainty (measured as the number of remaining MC diagnoses) by manipulating a set of control variables. As fully optimizing (2) leads to a combinatorial blow-up, FRACTAL$^{\mathrm{G}}$ achieves a compromise between (1) and (2) by using a greedy approximation approach for searching over the space of control assignments and a stochastic sampling method for computing the number of remaining MC diagnoses. The result is a fast algorithm (optimizing a whole FRACTAL scenario takes between 1 s for 74182 and 40 min for c6288 that decreases the diagnostic uncertainty according to a geometric decay curve. This geometric decay curve fits the FRACTAL data well (the goodness-of-fit criterion R^2 is 0.88 on average) and provides steep decay (the average decay rate p is 0.7).

We have compared the optimality and performance of FRACTAL$^{\mathrm{G}}$ to an ATPG-based algorithm for sequential diagnosis, FRACTAL$^{\mathrm{A}}$. While the average decay rate of both algorithms is similar (average p of FRACTAL$^{\mathrm{A}}$ is 0.73), the average goodness-of-fit criterion R^2 of FRACTAL$^{\mathrm{A}}$ is lower (0.84) which means that FRACTAL$^{\mathrm{G}}$ is consistently closer to the optimal solution than is FRACTAL$^{\mathrm{A}}$. FRACTAL$^{\mathrm{G}}$

has achieved better exponential decay compared to all algorithms except exhaustive control search. For example, the difference in the decay rate p between FRACTAL^{G} and exhaustive search for 74182 is 5.4%. The exhaustive control approach, however, takes minutes to complete even for a circuit as simple as 74182, and times-out with any model having more than 20 controls. As a result, we can conclude that FRACTAL^{G} trades off a small decrease in p for a significant performance speedup.

Chapter 6

Conclusions

In this dissertation we presented a new family of MBD reasoning algorithms that significantly improve the computational performance of state-of-the-art existing ones while minimizing the loss of optimality. To achieve this we have used approximation methods, and a number of techniques such as "inverted" search, greedy methods, and stochastic sampling.

6.1 Contributions

In summary, our major contributions can be stated as follows:

Framework and Formalism: This thesis represents multiple MBD problems such as minimal-diagnosis, MFMC, and test sequencing in a common consistency-based theoretical framework. This facilitates comparison and evaluation of results, and analysis and reuse of techniques.

Algorithms and Empirical Evaluation: The three major algorithms of this thesis (SAFARI, MIRANDA, and FRACTAL^{G}) efficiently solve worst-case *NP*-hard or worse problems by trading small degradation in the optimality of the diagnostic results for much larger gain in the computational performance (in some cases several orders-of-magnitude improvement in speed). This efficient trade-off is partly due to exploitation of specific properties in the search landscape of the diagnostic problems. All three algorithms achieve good optimality at low computational cost in diagnostic search problems that exhibit certain amount of continuity. Empirical evaluation on the ISCAS85/74XXX benchmark of combinational circuits quantitatively supports our claim that benchmark diagnostic problems are amenable to the greedy stochastic approaches described in this thesis.

The major highlights and contributions of the algorithms introduced in this thesis are outlined as follows:

SAFARI: With this algorithm we propose a novel approximation approach for computing multiple-fault diagnoses. SAFARI sacrifices guarantees of optimality, but for diagnostic systems in which faults are described in terms of an arbitrary deviation from nominal behavior SAFARI can compute diagnoses several orders of magnitude faster than competing algorithms.

We have demonstrated the performance gains of SAFARI on models having 19 up to 3 512 components, involving up to 7 232 variables. The maximum time for computing a single minimal diagnosis varies between 1 ms and 7 s. For comparison, CDA* times out on most experiments involving circuits with more than 383 components and HA* times out on most models with more than 546 components. Further, the computational performance of all the algorithms we have compared to, rapidly deteriorates when increasing the cardinality of the minimal-cardinality diagnosis, whereas the performance of SAFARI does not change significantly. Even SLS-based Max-SAT diagnostic methods are at least an order of magnitude slower than SAFARI (or less optimal).

The decrease of optimality in SAFARI due to the approximation approach is small. With single-faults and weak-fault models of the 74XXX/ISCAS85 circuits, the average cardinality of the diagnoses computed by SAFARI is 1.13 (13% worse than the optimum), and with double-faults, it is 2.22 (11% worse than the optimum). Furthermore we have shown (analytically and empirically) that the performance of SAFARI does not depend on the cardinality of the cardinality-minimal diagnosis.

To achieve optimality, SAFARI relies on a degree of continuity in the diagnostic model. Theory shows that a large class of models (weak-fault models) are continuous. Experiments with "stuck-at" 74XXX/ISCAS85 models show that observations either lead to zero diagnoses (inconsistent model and observation) or they lead to large number of diagnoses due to continuous diagnostic subspaces. This implies the existence of a phase transition region in the computation of minimal diagnosis, similar to the one in SAT [Gent and Walsh, 1994] where all diagnostic instances derived from combinational circuits are in the "easy" region.

SAFARI has been validated on a wide range of combinational circuits (weak- and strong-fault models of different size) but not on temporal or multi-valued encodings [Feldman et al., 2006]. Models with state are known to be less constrained than combinational ones [Hamscher and Davis, 1984], hence we expect SAFARI to achieve a similar optimality vs. performance trade-off.

We have applied SAFARI to two real-world problems: ADAPT (cf. Appendix B and PIM (cf. Appendix C) where we have modeled the physical

systems using the LYDIA modeling language (cf. Appendix A). The systems are beyond trivial, having tens of components. The results show that the SAFARI approach provides good precision metrics for only a fraction of the computational time of competing algorithms. In addition to the applications outlined in this thesis, the LYDIA/SAFARI approach has been successfully applied to production-level troubleshooting of semiconductor manufacturing equipment [Pietersma and van Gemund, 2007].

MIRANDA**:** MIRANDA computes observation vectors leading to faults of high cardinality as an approximation to MFMC observation vectors. The MFMC values computed by MIRANDA are optimal for 74XXX and 88% of the minimum for c432 (c432 has been brute-forced by de Kleer [2008]). For small combinational circuits (74XXX) MIRANDA computes MFMC observations several orders of magnitude faster than exhaustive algorithms. With large circuits, MIRANDA was able to compute an MFMC observation vector leading to a diagnosis of approximate cardinality of 36 in less than 6 min (c5315). The reason for the good MIRANDA performance is the fact that it exploits continuity in the space of output assignments in a class of well-formed benchmark circuits.

The implementation of MIRANDA in this thesis uses as a diagnostic oracle SAFARI, configured to compute subset-minimal diagnoses. This, and the greedy nature of MIRANDA result in observation vectors that are "best effort" approximations of MFMC diagnoses. Hence, MIRANDA overestimates the MFMC numbers due to the use of SAFARI and underestimates the MFMC numbers due to its greedy nature. The combination of the above error factors results in a smaller overall error. Due to the fact that computing MFMC vectors is Σ_2^P-complete (cf. Sec. 4.3) and the fact that MIRANDA accepts fault-models of circuits implementing arbitrary Boolean functions, we hypothesize that there is no algorithm below Σ_2^P that can compute bounded approximations of MFMC observation vectors.

In this thesis and in related publications [Kurtoglu et al., 2009], we have seen empirical evidence that the performance of existing MBD algorithms quickly deteriorates when increasing the cardinality of the injected faults (an MBD algorithm based on W-MAXSATZ, for example, was unable to compute c499 diagnoses of cardinality higher than 1 in 5 min). As the observations leading to high-cardinality faults were computed by MIRANDA, we can conclude that our implementation of MIRANDA delivers observation vectors of high optimality at acceptable computational cost.

FRACTAL**:** We have devised an algorithm, FRACTAL$^{\mathrm{G}}$, for sequential diagnosis, that computes near-optimal control assignments for reducing the diagnostic uncertainty (the number of remaining MC diagnoses). As fully optimizing the decay of the number of remaining MC diagnoses leads to a combinatorial blow-up, FRACTAL$^{\mathrm{G}}$ offers an optimality vs. speed trade-off by using

a greedy approximation approach for searching in the space of control assignments, and a stochastic sampling method for estimating the number of remaining MC diagnoses. The result is a fast algorithm (computing a whole FRACTAL scenario takes between 1 s for 74182 and 15 min for c7552) that decreases the diagnostic uncertainty according to a near-geometric decay curve. The reason for the efficiency of FRACTAL^{G} is that it exploits continuity in the space of control assignments. Experimentation shows that this is true even for a small number of control inputs.

FRACTAL^{G} has achieved better exponential decay compared to alternative approaches, except exhaustive control search. The difference in the decay rate p between FRACTAL^{G} and exhaustive search for 74182 is 5.4%. The exhaustive control approach, however, takes minutes to complete even for 74182 and times out with any model having more than 20 controls. As a result we can conclude that FRACTAL^{G} trades a small decrease in p for a significant performance speedup.

There are two challenges in applying MBD to real-world problems: (1) the modeling challenge [Narasimhan, 2002] and (2) the computational complexity of MBD. The two challenges are related. Models of complex physical systems have thousands to millions of variables [Provan and Wang, 2007] and abstracting at such magnitude is a significant challenge to modelers. The burden of manually abstracting (reducing the model complexity) can be alleviated by creating algorithms that can efficiently solve large systems. This would also facilitate, to some extent, automated translation from design specifications. This thesis helps solving the computational problem thus bringing MBD one step closer to large-scale, practical diagnosis of complex physical systems.

6.2 Improvements and Future Work

This thesis involves computer scientific, mathematical, and statistical disciplines such as complexity theory, satisfiability, ATPG, automated reasoning, optimization, experimental design, and others. As the main purpose of this dissertation is to substantiate the claims made in the introduction, not all aspects have been treated exhaustively.

What follows is a list of proposals for improving each of the algorithms presented in this thesis, followed by other suggestions for future work.

Algorithms Presented in this Thesis

SAFARI**:** The search performance of SAFARI can be improved for multiple diagnoses by adding "beam-search" capabilities to the algorithm. This means that once a subset-minimal diagnosis is found, the search does not restart, but continues with perturbing the diagnosis, thus searching for new diagnoses in the neighborhood of the existing one. This will lead to nearly

quadratic improvement in the time-complexity of SAFARI. Our estimate is based on (1) the average complexity of SAFARI and (2) the practical observation that faults usually propagate in their immediate neighborhood. Interestingly, (2) is also supported by analyzing combinational circuits. ISCAS85 observations, for example, lead more often to cliques of faulty components (these are subsets of components that can fail in any combination) than to faults dispersed throughout the whole circuit. The latter is due to circuit design and is related to model continuity as discussed in this thesis.

An approach to increase the utility metric of SAFARI diagnoses is to use a more precise ordering of the diagnostic results based on different prior probabilities and conflicts. It is straightforward to convert SAFARI to work on the dual problem of computing minimal diagnoses: computing minimal conflicts. One can show that in this case most properties and bounds of SAFARI still apply. After computing diagnoses from conflicts, one can use a Bayesian technique such as BARINEL [Abreu et al., 2009] for computing a posteriori fault probabilities. Considering only the first k diagnoses with highest a posteriori probability will improve precision-based metrics such as number of classification errors and utility.

MIRANDA**:** The MIRANDA algorithm offers ample opportunities for further development. Unlike the other two algorithms, the absolute CPU performance of MIRANDA is of secondary importance and the main goal is to improve the cardinality of the MIRANDA observations. One way to improve MIRANDA optimality is to use model decomposition. We have seen in Chapter 4 that MFMC is related to model decomposition (i.e., the number of variables shared by subsystems) and we hypothesize that the error in the MFMC approximations of MIRANDA can be bounded by model decomposition parameters (e.g., the maximum number of variables shared by different subsystems). The optimality of the MFMC approximations can be further improved by studying, quantifying, and using the notion of circuit "well-formedness". For example, the number of reconverging outputs in a circuit can be used for adjusting the MFMC search policy.

FRACTAL**:** The computational efficiency of the FRACTAL algorithm can be improved at least an order-of-magnitude by eliminating unnecessary computation. In particular, during search, FRACTAL$^{\mathrm{G}}$, can compute and cache single input values ruling out certain diagnoses. The consistency of assignments containing these cached values can be determined quickly (in logarithmic time) by performing cache look-ups. This approach is similar to conflict learning in satisfiability or structure exploitation in diagnosis [Feldman and van Gemund, 2006].

The performance of FRACTAL$^{\mathrm{G}}$ can be improved by using topological properties of the model. For example, if two controls are independent, they can be applied simultaneously for a faster decay of the number of diagnoses.

The decay speed of FRACTAL^{G} can be also improved by "learning" probabilities of observations from multiple applications of the algorithm on a given system.

Other Suggestions for Future Work

The algorithms presented in this thesis can be generalized and combined with approaches from other AI and computer science disciplines. In what follows we outline several potential high-impact applications of the approximation algorithms we have proposed.

Diagnosis of Systems with State: This thesis largely abstracts from the temporal behavior of systems. Although some elementary time-related properties can be modeled in LYDIA (for example by adding an extra array dimension to each variable), LYDIA has no built-in mechanisms for representing time and temporal constraints. As a step toward fusion of combinational MBD and Discrete Event Systems (DES) algorithms [Sampath et al., 1994], one can introduce time delays and/or flip-flops in LYDIA. Flip-flops in LYDIA would allow experiments with SAFARI on models of the ISCAS89 [Brglez et al., 1989] sequential circuits.

Provan [2009] has introduced a mechanism for translating hybrid automata to LYDIA propositional models that has been applied to a small industrial case [Behrens et al., 2009]. The results so far do not extend to larger models. Applying SAFARI to discrete models of complex hybrid systems will result to at least an order-of-magnitude speedup compared to deterministic algorithms.

Max-Fault ATPG: Automated testing of Very Large-Scale Integration (VLSI) circuits [Bushnell and Agrawal, 2000] is a multi-million dollar industry. MIRANDA offers near-theoretical compaction of ATPG sequences which is a significant progress compared to the single-fault "don't care" compaction methods used today. We hypothesize that, depending on the IC topology, stochastic MFMC-based ATPG algorithms offer near-optimal coverage and at least 70% reduction in the size of the testing sequences. This is a conservative estimate based on study of existing ATPG compaction techniques. Corno et al. [1997] show a method that performs static compaction of ATPG sequences and achieves 50% to 62% improvement in the size of the test set. The latest dynamic compaction methods report a 23% improvement compared to static methods [Czutro et al., 2009] and an MFMC-based ATPG would approach the theoretical limit which, so far, has not been established.

Discretization and Learning of Sensor Data: As is visible from the diagnostic competition, Appendix B, and other real-world examples, many sensor readings such as temperature, voltage, current, or pressure, are supplied as continuous values in the real domain. The MBD approaches described in

this thesis (and most of the referred algorithms) work with discrete variables. Hence, a real-world application has to discretize continuous sensor readings outside the MBD framework. In our application experiments with LYDIA we do this in an ad-hoc manner and existing literature and applications largely neglect this aspect of diagnosis.

The commercial package for MBD RODON [Lunde et al., 2006] uses interval-based logic but leaves the problem of computing the interval boundaries to the user. Even hybrid algorithms like HyDE [Narasimhan and Brownston, 2007] that allow real valued variables need to manually include conditions and landmark values for modeling system transitions. A potential future research with high impact on applying MBD techniques to real-world applications would be to use learning and signal processing methods to discretize sensor values. Automatic identification of nominal sensor readings is at least as complex as computation of diagnosis but if implemented, it would simplify the modeling process.

Automated Design for Reliability: Approximation MBD techniques can be used for tasks beyond diagnosis such as model-based design and redesign. We have already demonstrated that a technique based on MBD provides better trade-offs in adding system redundancy for increasing system reliability compared to traditional approaches [Feldman et al., 2009a]. Model-based design and redesign problems typically expose higher complexity than MBD problems. For example, one can formulate problems of generating or modifying system descriptions that are undecidable, and approximation algorithms have no alternative in approaching the intractability of these problems. We have already seen successful application of AI approaches like satisfiability to Computer-Aided Design (CAD) [Gu and Puri, 1995]. Although diagnostic and abductive reasoning is more complex than satisfiability, advances in algorithmic performance like the one presented in this thesis would allow the application of MBD to large-scale design and redesign problems leading to reduction in design labor and more optimal troubleshooting of design specifications.

As some of the above examples show, we expect approximation MBD algorithms to perform well on problems beyond diagnosis. Most of the MBD problems described in this thesis are close to logic-based abduction [Console et al., 1991] and can be solved with methods from abductive reasoning. This implies to some extent that the diagnostic techniques described in this thesis can be used for addressing a wider spectrum of abduction problems.

Appendix A

The LYDIA Modeling Language

We have designed a low-level language for system modeling called LYDIA (Language for sYstem DIAgnosis). LYDIA is also the framework in which are developed all the algorithms in this thesis. The LYDIA tool-kit also contains model translators (e.g., to CNF, DNF, OBDDs, etc), utilities, and reference implementation of algorithms such as CDA*, HA*, and others. The initial design and implementation of LYDIA can be found in van Gemund [2002, 2003]. All the material in this thesis corresponds to version 2.0 of LYDIA. This appendix presents an up-to-date overview of the LYDIA language syntax and semantics.

LYDIA models are collections of systems. Each system represents a component or a subsystem. LYDIA systems are introduced in Sec. A.1 where we also show the model of a small combinational circuit (a full-adder). In this early example we use variables and constraints intuitively. Section A.2 explains the basic LYDIA expressions. LYDIA variables and data types are discussed in detail in Sec. A.3. LYDIA expressions are discussed in Sec. A.4. Finally, Sec. A.5 discusses LYDIA predicates.

A.1 Systems and Subsystems

Synthetic and real-world systems have hierarchical structure which is represented in a LYDIA model by using systems, subsystems and system instantiations. The structure of a LYDIA model can be represented as multidigraph in which each system is a nodes and each instantiation is an edge. LYDIA preserves this structural information until, explicitly "flattened-out" in the model translation pipeline as hierarchy exploitation has the potential of speeding-up the reasoning process.

A system declaration includes the keyword **system**, the system name and a list of formal parameters. System declarations conform to the following syntax:

system <*name*> (<*type*$_1$> <*formal*$_1$>, [<*type*$_2$>] <*formal*$_2$>, ...)
{
⋮
}

Any valid LYDIA identifier[1] can be used for the system name. A system may have zero, one or more formals. Similar to when declaring local variables, subsequent identical formal types may be omitted.

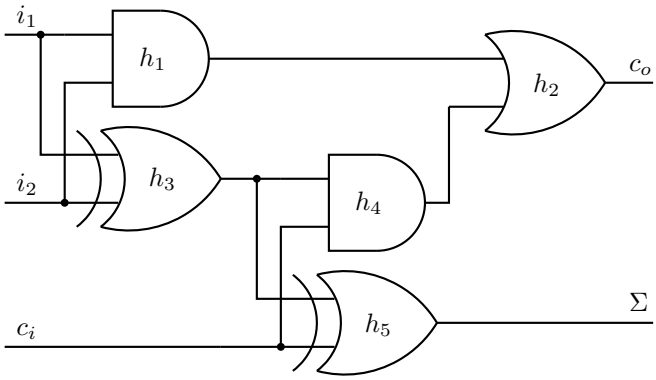


Figure A.1: *A full adder*

We show the use of systems and subsystems and some basic LYDIA constructs by modeling the Boolean full-adder shown in Fig. A.1. A full-adder consists of five logic-gates: an or-gate, two and-gates, and two xor-gates. Each of these three component types will be modeled as a LYDIA system. Note that LYDIA does not have an explicit notion of a component. LYDIA uses assumable (or health) variables to denote components. What follows are weak-fault[2] models of the three basic gate types.

```
system xor2(bool o, i1, i2)
{
    bool h;
    attribute probability(h) = (h ? 0.9999 : 0.0001);
    attribute health(h) = true;

    h => (o = (i1 != i2));
}
```

[1] Valid LYDIA identifiers are sequences of alphanumeric characters or the underscore, starting with a letter.

[2] For a discussion on fault-modeling strength, cf. Chapter 2.

```
system and2(bool o, i1, i2)
{
    bool h;
    attribute probability(h) = (h ? 0.9999 : 0.0001);
    attribute health(h) = true;

    h => (o = (i1 and i2));
}

system or2(bool o, i1, i2)
{
    bool h;
    attribute probability(h) = (h ? 0.9999 : 0.0001);
    attribute health(h) = true;

    h => (o = (i1 or i2));
}
```

Each of the above systems has three formal parameters[3]: an output *o* and two inputs (*i1* and *i2*). In each system there is a Boolean variable *h* that has the extra "health" and "probability" attributes. These two attributes make all *h* variables assumables. Finally, a single LYDIA predicate models the nominal behavior of the gate. We will discuss variables and constraints in the sections that follow.

A full-adder is composed of two half-adder, each of which consists of an xor-gate and an and-gate. A model of a half-adder is shown below.

```
system halfadder2(bool i1, i2, sum, carry)
{
    system and2 A(carry, i1, i2);
    system xor2 X(sum, i1, i2);
}
```

Finally, two half-adders and an or-gate are connected (instantiated) in the final model:

```
system fulladder2(bool i1, i2, ci, sum, carry)
{
        bool f, p, q;

        system halfadder2 HA1, HA2;

        HA1(i1, i2, f, p);
        HA2(ci, f, sum, q);

        system or2 O(carry, p, q);
}
```

[3]Referred to as ports in HDL languages such as Verilog™.

A.2 Basic Expressions

A LYDIA expression is a formula over the standard propositional operators $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication), and $\Leftrightarrow$ (equivalence). Negation is typed as "!" or, alternatively, as the keyword "**not**". Conjunction can be typed as "&&" or textually as "**and**" and disjunction is denoted both by the symbolic "||" and mnemonic "**or**". Implication is denoted as "=>" and equivalence either as "=" or "==".

To illustrate the use of the above five operators we will use simulation to compute the value of the Boolean function

$$f(x, y, z) \equiv x \wedge \neg y \vee (z \Rightarrow y) \Leftrightarrow z \tag{A.1}$$

In our example we will evaluate $f(1, 1, 0)$. The function is implemented in the following four-variable model:

```
system expr(bool f, x, y, z)
{
    attribute observable(x, y, z) = true;

    f == (x && !y || (z => y) = z);
}
```

The value $(1, 1, 0)$ at which we want to see f evaluated is specified as the following observation:

```
observation alpha_1
{
    x && y && !z;
}
```

Finally, we invoke LYDIA with the simulation option. The result is shown below.

```
lydia> sim expr alpha_1
x = true, y = true, f = false, z = false
```

A.3 Data Types

A LYDIA model is variable-centric, that is, it specifies the properties of a system as a set of constraints over one or more variables. In the reasoning process, these variables are always treated separately (i.e., early in the translation process arrays and structures are broken down into their elements). The LYDIA language allows rich variable-type semantics, resulting in short and expressive models. In what follows we discuss the built-in LYDIA variable data-types and the primitives for defining user-types.

A.3.1 Atomic Data Types

LYDIA has two atomic variable types: Booleans and enumerations. The former exists for convenience and for more aggressive optimization in the tool-kit (if the Boolean type was not built-in, a user could have easily introduced an enumeration defining it). A LYDIA modeler needs to declare all variables before using them. A variable declaration can be done either in the list of formal parameters of a system or in the system body.

system Bar(**bool** v_1, v_2, **bool** v_3, ..., v_n)
{
    ⋮
}

Good modeling style avoids unnecessary variables in the formal parameter list of a system – sharing variables makes models difficult to read and hinders the performance of some hierarchical diagnostic reasoners [Feldman and van Gemund, 2006].

The order of LYDIA statements is in most cases irrelevant for the model. LYDIA even does not make mandatory to declare a type or a variable before using it (declarations are mandatory, no matter if a declaration is before or after a corresponding defintion). An enumeration, for example, can be defined anywhere in the global scope of a model. An enumeration specifies a set of symbols that are allowed for a variable of a given type. Consider an example where a variable of type *color* can be either red or green or blue:

type color = **enum** { red, green, blue };

Once we have declared the *color* type, it can be used anywhere in the model as illustrated in the following model excerpt:

```
system ColorMixer(color in1, in2, out1, out2)
{
    color common;
        ⋮
}
```

Constant types should be always explicitly specified as in the right hand side of the LYDIA expression shown below. Having the *color* type declaration from above, we can assert the value of the internal variable *common* in the following manner:

```
common = color.red;
```

Note, that due to the semantics of the enumerations, a variable of type *color* can be either red or green or blue but it can assume exactly one value. Hence, any system containing an expression like:

```
(common = color.red) && (common = color.blue);
```

is trivially inconsistent, while the following constraint simplifies to true:

```
(common = color.red) || (common = color.green) || (common = color.blue);
```

A.3.2 Composite Data Types

LYDIA modelers can use arrays and structures to group related variables. In addition to that, the LYDIA language allows type aliasing.

Arrays

A LYDIA model can use arrays of any type and dimension. These arrays are "broken down" into sets of variables at compile[4] time, hence only statically-defined arrays are allowed. Unlike other languages, arrays may start from any index. We will illustrate the use of arrays by solving the well-known N-Queens problem. The model, shown below, defines the chess-board as a two-dimensional array of type **bool**. Placing a queen on the chess board will be expressed as assigning **True** to the respective member of the array. Below is the full N-Queens LYDIA model:

```
const int N = 6;

system nqueens()
{
    bool board[1:N][1:N];

    forall (i in 1 .. N) {
        exists (j in 1 .. N) { /* At Least One */
            board[i][j];
        }
        forall (k in 1 .. N - 1) { /* At Most One */
            forall (l in k + 1 .. N) {
                !board[i][k] || !board[i][l];
                !board[k][i] || !board[l][i];
            }
        }
    }

    forall (i in 1 .. N - 1) { /* At Most One */
        forall (j in 1 .. N - i) {
            forall (k in 1 .. N - i - j + 1) {
                !board[i + j - 1][j] || !board[i + j + k - 1][j + k];
                !board[j][i + j - 1] || !board[j + k][i + j + k - 1];
                !board[N - i - j + 2][j] || !board[N - i - j - k + 2][j + k];
                !board[N - j + 1][i + j - 1] ||
```

[4] We mean non-strict compilation, that is a translation to some normal form, e.g., a Conjunctive Normal Form (CNF).

```
                !board[N − j − k + 1][i + j + k − 1];
            }
        }
    }
}
```

To solve the N-Queens problem we have to impose a number of constraints. To do this we use quantifiers (cf. Sec. A.5.3) that specify that there is at least one queen in every row (or column) and at most one queen per row, column and diagonal of the board. Note, that for convenience, we started our arrays from one.

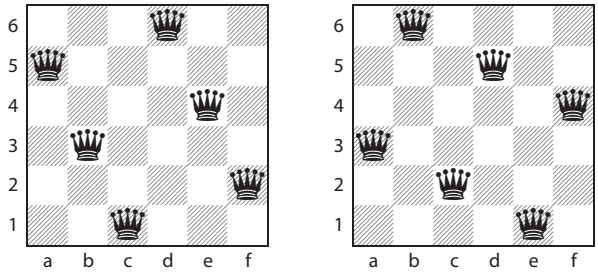


Figure A.2: *Two of the four solutions of the 6-queens problem*

An array declaration, like a variable declaration, can appear anywhere in a system and should adhere to the following syntax:

$<type> <name> [s_1 : e_1][s_2 : e_2] \ldots [s_n : e_n];$

In the syntax definition above, <type> is any atomic or composite type, <name> is a valid LYDIA identifier and s_i, e_i $(i = 1, 2, \ldots, n)$ are integers specifying the first and last elements of the array[5]. Alternatively, one can define arrays in a C-like manner, specifying only the size of each dimension.

$<type> <name> [d_1][d_2] \ldots [d_n];$

In the excerpt below, the declarations of the cubeX and cubeY arrays are of equivalent type.

```
bool cubeX[0:7][0:7][0:7];
bool cubeY[8][8][8];
```

[5]It is not necessary that $s_i < e_i$.

Structures

Variables can be grouped in structures. The latter are simply naming conveniences, as variables in structures share common suffixes (the names of the structures). The structures are expanded by LYDIA into separate variables at compile time. Consider several of the type definitions from the X-34 domain shown next.

```
type range = enum { below, inside, above };
type fluid = enum { helium, LO2, RP1 };

type line = struct
{
    range pressure,
    range temperature,
    fluid contents
};
```

The third type definition combines the *pressure*, *temperature* and *contents* variables in a structure defining a pipe. LYDIA imposes no restrictions on the types of the structure members. In the example above, the three member variables are all of enumeration types, while a structure member can be also a (nested) structure, or even an array (cf. Sec. A.3.2). This allows for a fairly complex composite LYDIA data types.

Continuing our *line* structure example, we declare *line* instances and use these instances for defining system constraints. This is illustrated in the following model fragment:

```
system PneumaticValveAndMicroSwitch(line i, o, c)
{
    valveState state;
    microSwitchState microSwitch;

                        ⋮

    if ((state = valveState.open) or (state = valveState.stuckOpen)) {
        o = i;
    }
    if ((state = valveState.closed) or (state = valveState.stuckClosed)) {
        o.pressure = range.below;
    }
}
```

The above excerpt (from a model of a pneumatic valve) defines three formal variables of the structure type *line*. These are named *i*, *o* and *c*. The consequent of the first **if** construct states that all member variables of *i* should be equal to all member variables of *o*. Hence, o = i is a shortcut for the following set of

constraints:

```
i.pressure = o.pressure;
i.temperature = o.temperature;
i.contents = o.contents;
```

Type Aliases

Type aliasing is part of the LYDIA typing system. Here is an example of how we can define a type *bar*, introduce a type *foo* that is an alias of *bar* and use the new *foo* type for instantiating a variable:

```
type bar = enum { red, green, blue };
type foo = bar;

system main()
{
    foo color = bar.green;
}
```

A.3.3 Variable Attributes

A LYDIA variable may carry attributes. Attributes have no universal semantics in the LYDIA language. Attributes can be used to carry application specific information for the end user, or, for applications using LYDIA.

An attribute assigns data to each value a variable may assume. This is best illustrated with an example. Suppose we have a variable *k* of type *color* (cf. the type definition of *color* in Sec. A.3.1). The variable *k*, can assume the values *red*, *green*, or *blue*. We can declare an attribute **rgb** of type **int** and assign a numeric value for each of the *k* values (*red*, *green*, or *blue*). After the diagnosis (or simulation) process computes values for *k*, the respective numerical attributes can be retrieved by the application that uses LYDIA.

There are three special built-in variable attribute types in LYDIA that do not have to be declared: *probability*, *health*, and *observable*. We have already seen these three attributes in the preceding examples and will discuss them in more detail below. These three attributes are used by all diagnostic solvers.

LYDIA attribute are typed. Attributes can be of type **bool**, **int**, **float**, and **string**. The variable attributing mechanism is limited in expressiveness, and does not allow user-defined types. A variable attribute declaration has the following syntax:

attribute <*type*> <*name*>

A LYDIA attribute assigns a value for each possible value a variable may assume. Any expression for which *all* values of a variable can be *precomputed* at model compilation time can be used as a variable attribute. Note that all except the three built-in attribute types should be declared at the global scope. Variable

attributes are defined in the same scope where the variables are defined (the order is irrelevant). Both formal and local variables can have attributes. What follows is the syntax of a variable attribute declaration.

attribute <*name*> (<*variable*$_1$>, <*variable*$_2$>, ..., <*variable*$_n$>) = <*expression*>

The simplest variable attributes assign constant values to variables.

We next discuss the three built-in attribute types. We have already mentioned that the **observable** attribute has a special meaning in LYDIA. Variables, which have this attribute specified, may get their values from the user, i.e., they can be *observed*.

Consider a model of an inverter. It has its input and output variables i and o marked as observables and starts as follows:

```
system inverter(bool i, o)
{
    attribute observable(i, o) = true;
                ⋮
}
```

We use h for the health of the above inverter. To mark a variable as *health* or *assumable* (the latter two are synonyms) we have to assign a pair of built-in attributes: **probability** and **health**. The former is of type **float**, while the latter is of type **bool**. We continue the inverter model with assigning those two attributes to h:

```
                ⋮
    attribute probability(h) = (h ? 0.95 : 0.05);
    attribute health(h) = h;
```

The first attribute specifies that h is 0.95 for h = **true**, and 0.05 for h = **false**. Note that all a priori probabilities must sum to unity (otherwise the LYDIA model compiler will produce a warning message). The health attribute specifies when a component is "healthy". In the model continuation below the inverter will be healthy when h = **true**, otherwise, for h = **false**, it is broken (or faulty).

As an alternative to the above two attributes we can use f = **false** to denote a faulty gate (thus inverting the health logic) in the following way:

```
                ⋮
    attribute probability(f) = (f ? 0.01 : 0.99);
    attribute health(f) = !f;
```

The two lines above specify that the prior fault probability of f = **false** is 0.99 and that of f = **true** is 0.01. The meaning of the health attribute is inverted in comparison to h, i.e., f is faulty for f = **false**.

Having discussed the three built-in attributes, we continue our discussion with an example on the use of user-defined attributes. The model excerpt below declares three user type attributes. These can be used in a model, where, for example, we would like to attach user-supplied data to some health variables. After the computation of diagnosis, this data can be used for locating the faulty component or computing cost of repair, etc.

```
    ⋮
attribute string location;
attribute float cost;
attribute bool input;
    ⋮
```

The part from the LYDIA system description, shown next, uses the three newly defined attribute types (*location*, *cost*, and *input*) to attach user-supplied values to the variable *h*.

```
    ⋮
attribute cost(h) = h ? 97.5 : 2.5;
attribute input(h) = true;
attribute location(h) = "quadrant 12";
    ⋮
```

In the above case the *input* and *location* attributes are constant for every value of *h*.

As we have seen from the **observable** attribute, it is possible to attribute multiple variables with a single statement. In our example, however, the attribute had a constant value. The probability, for example, is different depending on the value of *h*, hence the expression h ? 0.95 : 0.05. The above example would not work if we try to attribute multiple variables at the same time due to the fact that *h* is mentioned in the right-hand side of the attribute expression. For example, trying to compile:

```
system main()
{
    bool h1, h2;

    attribute probability(h1, h2) = h1 ? 0.95 : 0.05;
}
```

would lead to the following error:

```
err.sys:5: error: can't evaluate probability(h2) for h2 = false
err.sys:5: error: (Each non-evaluated value is reported only once
err.sys:5: error: for each attribute it appears in.)
```

As the alternative of attributing each variable separately is cumbersome, LYDIA allows the use of attribute aliases in the attribute expression as illustrated next:

```
attribute probability(h1, h2) = \x x ? 0.95 : 0.05;
```

In this case x will alias first *h1* and then *h2* and both *h1* and *h2* will receive the same a priori probability. Only variable of the same type can be attributed in this way. The same technique may be used for attributing variable arrays, as is shown in the example below:

```
bool g[4][4];

attribute probability(g) = \x x ? 0.99 : 0.01;
attribute health(g) = true;
```

When attributing arrays, it is also possible to attribute all the elements of the arrays separately, or in groups:

```
bool z[3];

attribute observable z[0:1] = true;
attribute health z[2] = true;
```

Variables grouped in structures (cf. Sec. A.3.2) can be attributed by using attribute aliases if all variables are of the same type. The same applies for complex data types consisting of structures and arrays. What follows is an example of attributing all the variables in a structure:

```
type flow = struct { bool fsign, bool fder };

attribute float d;

system main()
{
    flow f;

    attribute d(f) = \x x ? 10 : 100;
}
```

The above mechanism would not work if the variables in a structure are of different

types. In this case we can use typed aliases as illustrated next:

```
type content = enum { t1, t2, t3 };
type flow = struct { bool fsign, bool fder, content c1, content c2 };

attribute float d;

system main()
{
    flow f;

    attribute d(f) = \x::bool x ? 10 : 100;
    attribute d(f) = \x::content cond(x) (content.t1 -> 1; content.t2 -> 2)
}
```

As we have seen, variable attributes provide a powerful macro-mechanism for attaching user-defined data to variables. In the examples we have used some LYDIA expressions to compute values for each value a variable may assume. Note, that the aliasing mechanism which we have seen and the **int**, **float**, and **string** attribute types present in the attribute processing mechanism only and they can not be used outside of attribute expressions.

A.4 Expressions

LYDIA expressions are used for specifying variable constraints.

A.4.1 Conditional Expressions

The syntax of the arithmetic if in LYDIA is similar to the one in C:

$<expression_1>$? $<expression_2>$: $<expression_3>$

Arithmetic if expressions can be nested. Each arithmetic if is internally translated to $(expression_1 \Rightarrow expression_2) \wedge (\neg expression_1 \Rightarrow constraints_3)$. The conditional if in the inverter model below stipulates that inverter should be healthy if the value of its input i is equivalent to the value of its output o.

```
system inverter(bool h, i, o)
{
    attribute observable(i, o) = true;
    attribute health(h) = h;
    attribute probability(h) = (h ? 0.95 : 0.05);

    h ? (i == o) : (i != o);
}
```

An expression without equivalent in other languages is the conditional switch. A conditional switch expression has the following syntax:

cond (<*expression*$_1$>) (
    <*expression*$_2$> –> <*expression*$_3$>
                ⋮
    <*expression*$_{2n}$> –> <*expression*$_{2n+1}$>
    default –> <*expression*$_{2n+2}$>
)

Denote the propositional (possibly multi-valued [Feldman et al., 2006]) **Wff** equivalent to <*expression*$_i$> as e_i ($1 \leq i \leq 2n+2$). The LYDIA compiler rewrites each arithmetic switch expression to the following system of equations:

$$\left|\begin{array}{l} (e_1 \Leftrightarrow e_2) \Rightarrow e_3 \\ (e_1 \Leftrightarrow e_4) \wedge \neg(e_1 \Leftrightarrow e_2) \Rightarrow e_5 \\ (e_1 \Leftrightarrow e_6) \wedge \neg(e_1 \Leftrightarrow e_4) \wedge \neg(e_1 \Leftrightarrow e_2) \Rightarrow e_7 \\ \qquad\qquad \vdots \\ (e_1 \Leftrightarrow e_{2n}) \wedge \neg(e_1 \Leftrightarrow e_2) \wedge \neg(e_1 \Leftrightarrow e_4) \wedge \cdots \wedge \neg(e_1 \Leftrightarrow e_{n-1}) \Rightarrow e_{2n+1} \\ \neg(e_1 \Leftrightarrow e_{2n}) \wedge \neg(e_1 \Leftrightarrow e_2) \wedge \neg(e_1 \Leftrightarrow e_3) \wedge \cdots \wedge \neg(e_1 \Leftrightarrow e_{n-1}) \Rightarrow c_{2n+2} \end{array}\right. \tag{A.2}$$

The use of the arithmetic switch is exemplified by the following model excerpt (part of a model of an airplane fuel switch valve):

```
system selectorValve(selectorPosition selector, bool enginePumpOn, ···)
{
                    ⋮
    if (h == hSelector.nominal) {
        flowOut == cond (selector) (
            selectorPosition.off –> mass.zero;
            selectorPosition.left –> flowLeft;
            selectorPosition.right –> flowRight;
            default –> mass.zero);
    }
}
```

The above arithmetic switch specifies that if the switch selector is in "off" position, then the value of the *flowOut* variable should be equivalent to the constant *mass.zero*. Similarly, if the switch is in position "left" the value of the *flowOut* variable should be equivalent to the value of the variable *flowLeft*, etc. Note that the default case can be omitted if there are corresponding expressions for all the values *expression*$_1$ can assume. If this is not the case and a default expression is not specified, LYDIA will produce an error.

A.4.2 Qualitative Inequalities

LYDIA enumeration variables can be compared. Consider the following model:

```
type letter = enum { a, b, c, d };

system qi()
{
    letter x, y;

    bool h;

    attribute observable(x, y) = true;
    attribute health(h) = h;
    attribute probability(h) = (h ? 0.95 : 0.05);

    if (h) {
        x < y;
    }
}
```

and the two observation vectors alpha_2 and alpha_3:

```
observation alpha_2
{
    (x == letter.a) && (y == letter.c);
}

observation alpha_3
{
    (x == letter.c) && (y == letter.a);
}
```

Observation alpha_2 results in the empty set of diagnoses while swapping x and y in alpha_3 results in a negative value for h as demonstrated by the transcript below:

```
lydia> diag qi alpha_1
lydia> fm
d1 = {  }
lydia> diag qi alpha_2
lydia> fm
d1 = { h = false }
```

Qualitative inequalities should be used with care as they may lead to combinatorial blow-ups. Consider an enumeration that can assume n values:

type <*name*> = enum { t_1, t_2, ..., t_n };

A qualitative inequality such as $x \leq y$ would be internally expanded to

$$\begin{array}{l} [(x \Leftrightarrow t_1) \wedge (y \Leftrightarrow t_1)] \vee \\ \{(x \Leftrightarrow t_2) \wedge [(y \Leftrightarrow t_1) \vee (y \Leftrightarrow t_2)]\} \vee \\ \qquad\qquad \vdots \\ \{(x \Leftrightarrow t_n) \wedge [(y \Leftrightarrow t_1) \vee (y \Leftrightarrow t_2) \vee \cdots \vee (y \Leftrightarrow t_n)]\} \end{array} \tag{A.3}$$

which is difficult for translation to a normal form (CNF or DNF).

A.5 Predicates

LYDIA constraints are specified in the system bodies as sequences of predicates separated by semicolons. After converting all predicates to a normal form, LYDIA uses the normal forms conjunction as a system model.

A.5.1 Basic Predicates

Basic predicates contain expressions. Consider a model of a buffer:

```
system buffer(bool o, i)
{
    bool h;

    attribute probability(h) = (h ? 0.99 : 0.01);
    attribute health(h) = h;

    h => (o = i);
}
```

The last line of the above model (h => (o = i)) is a basic predicate.

A.5.2 Conditional Predicates

The **if** predicate is one of the most-commonly used constructs in LYDIA. The **if** syntax is shown below.

```
if (<expression_1>) {
      <constraints_1>
} else if (<expression_2>) {
      <constraints_2>
            ⋮
} else if (<expression_n−1>) {
      <constraints_n−1>
} else {
      <constraints_n>
}
```

The **if** construct is a verbose way to write conditional expressions (cf. Sec. A.4.1). In the next excerpt, which comes from the plane domain, we show some typical use of the **if** predicate. The model below shows a way to build qualitative models of sensors, where one or more measurements should coincide with the sensor readings (iff the sensor is functioning normally). Furthermore, the reasoner can discern different sensor faults (the model below is of a fuel sensor), given the sign of the time-derivative of a reading.

```
system sensor(mass real, deltaReal, indicated, deltaIndicated)
{
                ⋮
    if (h = sensorState.nominal) {
        indicated == real;
        deltaIndicated == deltaReal;
    }

    if (h = sensorState.stuckLow) {
        (indicated = mass.zero) || (indicated = mass.low);
        deltaIndicated = mass.zero;
    }                                                          10

    if (h = sensorState.stuckHigh) {
        indicated = mass.high;
        deltaIndicated = mass.zero;
    }
}
```

It is often more elegant to model a succession of **if** and **else if** predicates as a single **switch** construct. The reader ought not to be confused by the resemblance of the LYDIA **switch** construct to the one in some procedural languages (e.g., C and Java). In LYDIA the **switch** construct is used in a more powerful, higher-level, way than a sequence of **if** and **else if** predicates. The syntax of the **switch** construct is the following:

switch (<*expression*$_1$>) {
    <*expression*$_2$> –> {
        <*constraints*$_1$>
    }
            ⋮
    <*expression*$_n$> –> {
        <*constraints*$_{n-1}$>
    } **default** –> {
        <*constraints*$_n$>
    }
}

If we denote the propositional (possibly multi-valued [Feldman et al., 2006]) **Wff** equivalent to <*expression*$_i$> as e_i and the conjunction of **Wff** contained in <*constraints*$_j$> as c_j, then the LYDIA compiler rewrites every **switch** construct in a model to the following system of equations:

$$\left|\begin{array}{l} (e_1 \Leftrightarrow e_2) \Rightarrow c_1 \\ (e_1 \Leftrightarrow e_3) \wedge \neg(e_1 \Leftrightarrow e_2) \Rightarrow c_2 \\ (e_1 \Leftrightarrow e_4) \wedge \neg(e_1 \Leftrightarrow e_2) \wedge \neg(e_1 \Leftrightarrow e_3) \Rightarrow c_3 \\ \qquad\qquad\qquad \vdots \\ (e_1 \Leftrightarrow e_n) \wedge \neg(e_1 \Leftrightarrow e_2) \wedge \neg(e_1 \Leftrightarrow e_3) \wedge \ldots \wedge \neg(e_1 \Leftrightarrow e_{n-1}) \Rightarrow c_{n-1} \\ \neg(e_1 \Leftrightarrow e_n) \wedge \neg(e_1 \Leftrightarrow e_2) \wedge \neg(e_1 \Leftrightarrow e_3) \wedge \ldots \wedge \neg(e_1 \Leftrightarrow e_{n-1}) \Rightarrow c_n \end{array}\right. \tag{A.4}$$

The practical use of the LYDIA **switch** construct is illustrated in a component model excerpt from the International Berthing and Docking Mechanism (IBDM) hard-docking system.

```
if (((motorSelector = EMAMotorSelection.primary) ?
                      stateMotor1 :
                      stateMotor2) = EMAState.healthy) {
    switch (regime) {
        EMARegime.latching ->
        {
            (current = range.inside) => (gearOut = ringGear.latching);
            (current = range.above) => (gearOut = ringGear.forceLatching);
        }
        EMARegime.latched ->                                              10
        {
            (current = range.below) => (gearOut = ringGear.latched);
        }
        EMARegime.unlatching ->
        {
            (current = range.inside) => (gearOut = ringGear.unlatching);
            (current = range.above) => (gearOut = ringGear.forceUnlatching);
        }
        EMARegime.unlatched ->
        {                                                                 20
            (current = range.below) => (gearOut = ringGear.unlatched);
        }
    }
}
```

A.5.3 Quantifiers

LYDIA allows quantifiers over the elements of an array. These are expanded at compile time. There are two quantifiers: existential (**exists**) and universal (**forall**). The use of both should conform to the syntax shown next.

<**forall**|**exists**> (*index* in *start* .. *end*)
{
    <*constraints*>
}

Here, *index* is a variable identifier and *start* and *end* are both integer expressions (*start* does not need to be greater than *end*). Nesting of **forall** and **exists** blocks is allowed.

1		3	
2			
			3
	2		1

1	4	3	2
2	3	1	4
4	1	2	3
3	2	4	1

Figure A.3: *An example* 4×4 *Sudoku puzzle (left) and its solution (right)*

The use of the universal qualifier (**forall**) can be best illustrated in solving a small 4×4 Sudoku puzzle. The actual problem we will solve and its solution are shown in Figure A.3. After we define the main data type of type *tile* in the model that follows below we need a system that constraints the elements of a 4-element vector to always assume different values. This is done with the help of the two nested **forall** constructs.

```
type tile = enum { V1, V2, V3, V4 };

system Everywhere(tile g[4])
{
    forall (i in 0 .. 3) {
        forall (j in i + 1 .. 3) {
            g[i] != g[j];
        }
    }
}
```

Note the use of the index variables (in this example i and j). Their scope is restricted to the body of the respective **forall** block. The index variables are always integers and they can be used in integer expressions as far as static evaluation of all array subscripts can be performed at model compilation time.

Having the basic inequality constraint (which we are going to apply row-wise, column-wise and block-wise) we can define the Sudoku grid. Solving this empty

grid, will in fact generate all possible Sudoku instances.

```
system Grid(tile g[4][4])
{
    system Everywhere row[4], col[4], blk[4];

    forall (i in 0 .. 3) {
        row[i]([ g[i][0], g[i][1], g[i][2], g[i][3] ]);
        col[i]([ g[0][i], g[1][i], g[2][i], g[3][i] ]);
    }
    blk[0]([ g[0][0], g[0][1], g[1][0], g[1][1] ]);
    blk[1]([ g[0][2], g[0][3], g[1][2], g[1][3] ]);
    blk[2]([ g[2][0], g[2][1], g[3][0], g[3][1] ]);
    blk[3]([ g[2][2], g[2][3], g[3][2], g[3][3] ]);
}
```

In the above system, the **forall** construct is used to repeat constraints for each row and column of the grid. What remains is to initialize the hint tiles and to feed the resulting system to a solver. The next and final excerpt from this example defines the top-level Sudoku problem, supplying constraints for the known hints.

```
system Sudoku()
{
    tile g[4][4];
    attribute health(g) = true;
    attribute probability(g) = 0.25;

    system Grid grid(g);

    g[0][0] = tile.V1; g[0][2] = tile.V3;
    g[1][0] = tile.V2;
    g[2][3] = tile.V3;
    g[3][1] = tile.V2; g[3][3] = tile.V1;
}
```

The last system does not include any quantifiers, but we will use it to complete our example. The solution of the Sudoku puzzle specified above will be the only satisfiable term of the Disjunctive Normal Form (DNF) of the above system.

Solving the example above with, for example, the LYDIA dense-encodings DNF solver indeed yields the correct solution (cf. Figure A.3).

```
lydia> modes
g[0][0] = V1, g[0][1] = V4, g[0][2] = V3, g[0][3] = V2,
g[1][0] = V2, g[1][1] = V3, g[1][2] = V1, g[1][3] = V4,
g[2][0] = V4, g[2][1] = V1, g[2][2] = V2, g[2][3] = V3,
g[3][0] = V3, g[3][1] = V2, g[3][2] = V4, g[3][3] = V1
```

The next two excerpts are from a model of the IBDM domain. The first one stipulates that if some condition is satisfied, then, for two arrays, one or more

elements of each array should assume a certain value[6].

```
if ((primaryRingGear = ringGear.forceLatching) or
    (primaryRingGear = ringGear.forceUnlatching)) {
    exists (i in 0 .. 11) {
        (primaryGearBoxState[i] = gearBoxState.jammed) or
        (primaryLatchState[i] = latchState.jammed);
    }
}
```

Finally, we illustrate the workings of the **forall** construct with one more example from the IBDM domain. In this excerpt we couple a dozen of gear boxes to a dozen of structural latch assemblies (these are all systems which share variables).

```
forall (i in 0 .. 11) {
    gearBox[i](primaryGearBoxState[i], ..., latchAssemblySecondaryCrank[i]);
    structuralLatchAssembly[i](primaryLatchState[i], ..., latchRoller[i]);
}
```

[6]The intended meaning of the model is that an increased force when moving the primary or secondary ring gears implies the existence of a gear box (there are twelve gear-boxes driven by the two ring gears) or a latch which is mechanically jammed.

Appendix B

Case Study: ADAPT EPS

In this appendix we demonstrate the applicability of LYDIA/SAFARI to real-world applications, such as the diagnosis of the Electrical Power System (EPS) testbed in the ADAPT lab at NASA Ames Research Center [Poll et al., 2007].

ADAPT EPS has been used for the evaluation of the engineering and application aspects of diagnosis at the First International Diagnostic Competition (DXC'09) [Kurtoglu et al., 2009]. LYDIA/SAFARI competed with twelve other Diagnostic Algorithms (DAs) (model-based or not); many of those DAs optimizing one or more of the nine diagnostic metrics computed by the organizers.

Section B.1 offers of a description of ADAPT EPS and the diagnostic scenarios from which we have obtained observation vectors. Section B.2 describes the LYDIA ADAPT EPS model and can be used as a guideline for building models of electrical systems. Section B.3 presents experiments and comparison to other DXC'09 DAs.

B.1 System Overview

The ADAPT EPS testbed provides a means for evaluating DAs through the controlled insertion of faults in repeatable failure scenarios. The EPS testbed incorporates low-cost Commercial Off-The-Shelf (COTS) components connected in a system topology that provides the functions typical of aerospace vehicle electrical power systems: energy conversion/generation (battery chargers), energy storage (three sets of lead-acid batteries), power distribution (two inverters, several relays, circuit breakers, and loads) and power management (command, control, and data acquisition).

The EPS delivers Alternating Current (AC) and Direct Current (DC) power to loads, which in an aerospace vehicle could include subsystems such as the avionics, propulsion, life support, environmental controls, and science payloads. A data acquisition and control system commands the testbed into different configurations

and records data from sensors that measure system variables such as voltages, currents, temperatures, and switch positions. Data are presently acquired at a 2 Hz rate.

The scope of the ADAPT EPS testbed used in this case study is shown in Fig. B.1. There are two systems specified in the same physical testbed: ADAPT-Lite and ADAPT.

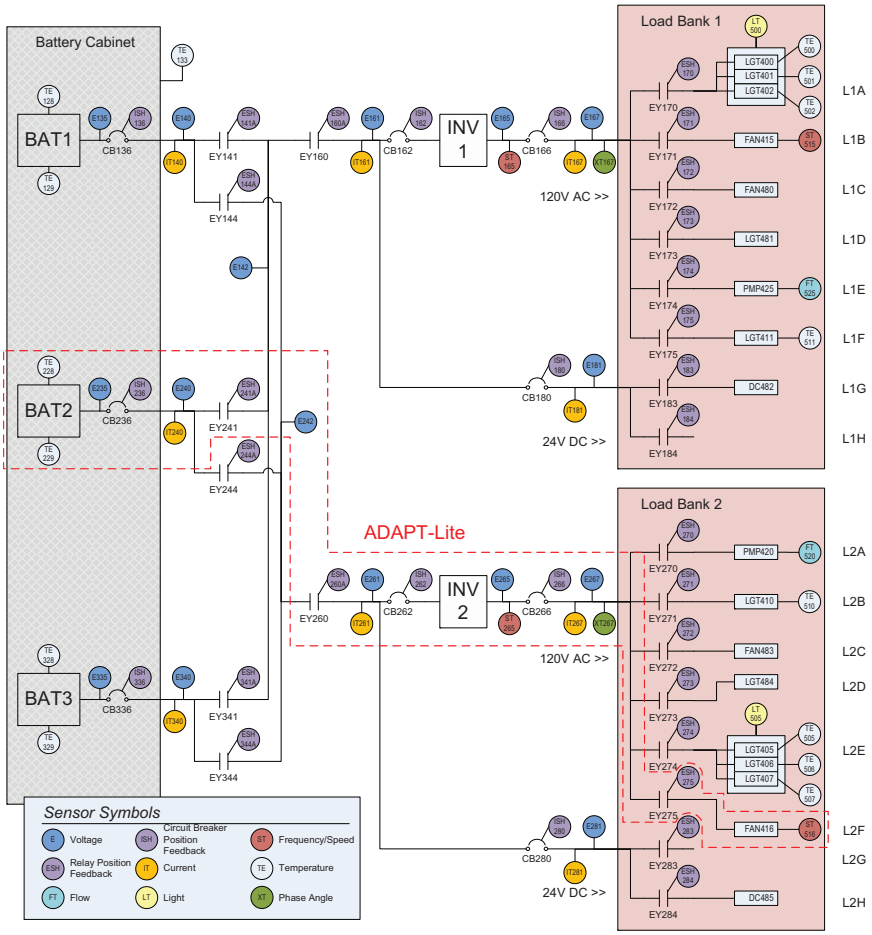


Figure B.1: *ADAPT EPS (diagram courtesy of NASA Ames Research Center)*

System ADAPT-Lite includes a single battery and a single load as indicated by the dashed lines in the schematic (Fig. B.1). The initial configuration for ADAPT-Lite

data has all relays and circuit breakers closed and no nominal mode changes are commanded during the scenarios. Hence, any noticeable changes in sensor values may be correctly attributed to faults injected into the scenarios. Furthermore, ADAPT-Lite is restricted to single faults.

Table B.1: *ADAPT and ADAPT-Lite differences*

Aspect	ADAPT-Lite	ADAPT
|COMPS|	37	173
# of modes	93	430
relays initially	closed	open
circuit-breakers initially	closed	closed
nominal mode changes	no	yes
multiple faults	no	yes

System ADAPT includes all batteries and loads in the EPS. The initial configuration for ADAPT has all relays open and nominal mode changes are commanded during the scenarios. The commanded configuration changes result in adjustments to sensor values as well as transients which are nominal and not indicative of injected faults, in contrast to ADAPT-Lite. Finally, multiple faults may be injected in ADAPT. The differences between ADAPT-Lite and ADAPT are summarized in Table B.1.

ADAPT supports the repeatable injection of faults into the system in three ways:

Hardware-Induced Faults: These faults are physically injected at the testbed hardware. A simple example is tripping a circuit breaker using the manual throw bars. Another is using the power toggle switch to turn off an inverter. Faults may also be introduced in the loads attached to the EPS. For example, the valve can be closed slightly to vary the back pressure on the pump and reduce the flow rate.

Software-Induced Faults: In addition to hardware faults, ADAPT supports software simulation of faulty behavior. Software fault injection includes one or more of the following: (1) sending commands to the testbed that are not intended for nominal operations; (2) blocking commands sent to the testbed; and (3) altering the testbed sensor data.

Real Faults: In addition the aforementioned two methods, real faults may be injected into the system by using actual faulty components. A simple example includes a burned out light bulb. This method of fault injection was not used in this study.

For results presented in this case study, only abrupt discrete (change in operating mode of component) and parametric (step change in parameter value) faults

are considered. Nominal and failure scenarios are created using hardware and software-induced fault injection methods. The DAs are tested against a number of scenarios, each approximately four minutes in length.

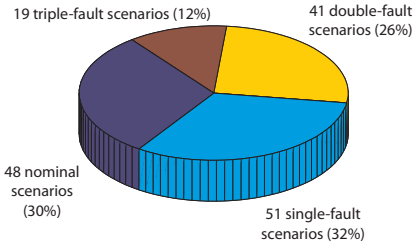


Figure B.2: *Fault-cardinality distribution of the ADAPT scenarios*

The ADAPT-Lite experiments include 36 nominal and 56 single-fault scenarios. The ADAPT experiments have 48 nominal and 111 fault scenarios, which include single-, double-, and triple-faults. Figure B.2 shows the fault-cardinality distribution of the ADAPT scenarios. Table B.2 summarizes the type of faults used for ADAPT. The majority of the ADAPT faults involve sensors (102) and loads (30).

B.2 ADAPT EPS Model

The Lydia ADAPT-Lite and ADAPT models share a common type and component libraries. These two libraries are domain independent and can be used in models of other electrical systems. The topological description of ADAPT-Lite is simple and we have created it manually, but for ADAPT we have used automatic translation from DXC XML system descriptions [Feldman et al., 2010].

The listing below shows the top-level ADAPT-Lite system. The component library we have used is not fully compositional due to time limitations for modeling. Creating fully compositional component libraries is a topic of ongoing research and is known to be challenging [Struss et al., 1996]. This is the reason for the global constraints at the end of the top-level ADAPT-Lite system.

```
#include "types.sys"
#include "comps.sys"

system ADAPTLite()
{
    Wire WI235; // BAT2 <–> CB236
    Wire WI240; // CB236 <–> EY244
          ⋮
```

Table B.2: *ADAPT-Lite and ADAPT faults*

Type	Subtype	Fault	ADAPT-Lite	ADAPT
battery	-	degraded	3	1
circuit breaker	-	failed-open	5	18
inverter	-	failed-off	2	10
load	basic	failed-off	–	1
	fan	failed-off	2	5
		over-speed	2	2
		under-speed	2	3
	light bulb	failed-off	–	14
	pump	failed-off	–	3
		blocked	–	2
relay		stuck-closed	6	3
		stuck-open	–	26
sensor	position	stuck	11	26
	current, flow, light, phase angle, speed, temperature, voltage	offset	12	35
		stuck	11	41
			–	
		Total:	56	190

```
Rotation rotationFAN416;
Load loadFAN416, loadINV2, loadedINV2;
Temperature heatBAT2;

system Battery BAT2(WI235, heatBAT2);
system CircuitBreakerCmd CB236(WI235, WI240, EY236_OP, posCB236);
system Relay EY244(WI240, WI242, EY244_CL, posEY244);
    ⋮
system Fan FAN416(WI275, rotationFAN416, loadFAN416);

CmdCircuitBreakerCmd EY236_OP;
CmdRelay EY244_CL, EY260_CL, EY275_CL;

system TemperatureSensor TE228(heatBAT2), TE229(heatBAT2);
system VoltageSensor E235(WI235);
    ⋮
system SpeedTransmitter ST516(rotationFAN416);

Position posCB236, posEY244, posEY260, posCB262, posCB266, posEY275;
```

```
    system PositionSensor ISH236(posCB236), ESH244A(posEY244);
          ⋮
    if (loadFAN416 == Load.Nominal) {
        WI267.current == Current.Nominal;
        WI267.phaseAngle == PhaseAngle.Nominal;
        WI265.frequency == Frequency.Nominal;
        loadedINV2 == Load.Nominal;
    } else {
        WI267.current == Current.Low;
        WI267.phaseAngle == PhaseAngle.OffNominal;
        WI265.frequency == Frequency.Low;
        loadedINV2 == Load.Low;
    }
          ⋮
}
```

The ADAPT-Lite and ADAPT models use a number of user-type variables (cf. Sec A.3). These variable types (some of which are shown in the excerpt below) represent quantities like current and voltage or discrete states like sensor positions. We have created a rudimentary mechanism for discretizing the DXC sensor data and mapping it to the qualitative values shown below. This mechanism uses a map from real-valued intervals to LYDIA qualitative values for converting the DXC sensor readings into SAFARI observation vectors.

```
type HealthBattery = enum { Nominal, Degraded };
type HealthCircuitBreakerCmd = enum { Nominal, FailedOpen, StuckClosed };
          ⋮
type CmdRelay = enum { Opened, Closed };
type CmdCircuitBreakerCmd = enum { Opened, Closed };
type CmdInverter = enum { Off, On };
          ⋮
type Position = enum { Opened, Closed };
type Voltage = enum { Low, Nominal, High };
type Current = enum { Low, Nominal, High };
          ⋮
type Wire = struct { Voltage voltage, Current current, Frequency frequency, ··· };
```

The listing below shows a LYDIA model of a commandable circuit breaker. The circuit breaker has two fault states: failed open and stuck closed. According to the DXC specification, the circuit breaker can be also remotely set in an opened or closed position. Connected to each circuit breaker is a position sensor, the model of which is shown later in this appendix. The value of the *pos* variable reflects the actual position of the circuit breaker. The model of the coupled position sensor

allows SAFARI to infer the sensor health by comparing the value of *pos* to the observed sensor reading.

```
system CircuitBreakerCmd(Wire i, o, CmdCircuitBreakerCmd commanded,
                         Position pos)
{
    HealthCircuitBreakerCmd h;

    attribute health(h) = cond (h) (
        HealthCircuitBreakerCmd.Nominal -> true;
        HealthCircuitBreakerCmd.FailedOpen -> false;
        HealthCircuitBreakerCmd.StuckClosed -> false
    );
    attribute probability(h) = cond (h) (
        HealthCircuitBreakerCmd.Nominal -> 0.8;
        HealthCircuitBreakerCmd.FailedOpen -> 0.1;
        HealthCircuitBreakerCmd.StuckClosed -> 0.1
    );

    switch (h) {
        HealthCircuitBreakerCmd.Nominal ->
        {
            switch (commanded) {
                CmdCircuitBreakerCmd.Closed ->
                {
                    pos = Position.Closed; o.voltage = i.voltage;
                }
                CmdCircuitBreakerCmd.Opened ->
                {
                    pos = Position.Opened;
                }
            }
        }
        HealthCircuitBreakerCmd.FailedOpen ->
        {
            pos = Position.Opened;
        }
        HealthCircuitBreakerCmd.StuckClosed ->
        {
            pos = Position.Closed; o.voltage = i.voltage;
        }
    }
}
```

The ADAPT EPS uses two off-the-shelf inverters. There is an inverter in each ADAPT DC or AC circuit. In the listing below we show a LYDIA model of an inverter. We have introduced the *loadOut* variable because the inverters switch

to a sleep mode when there is no load and the respective AC voltage sensors show no voltage. The model constraints specify the output voltage depending on the state of the inverter.

```
system Inverter(Wire i, o, CmdInverter commanded, Load loaded, loadOut)
{
    HealthInverter h;

    attribute health(h) = cond (h) (
        HealthInverter.Nominal -> true;
        HealthInverter.FailedOff -> false
    );
    attribute probability(h) = cond (h) (
        HealthInverter.Nominal -> 0.99;
        HealthInverter.FailedOff -> 0.01
    );

    switch (h) {
        HealthInverter.Nominal ->
        {
            switch (commanded) {
                CmdInverter.On ->
                {
                    if (i.voltage == Voltage.Nominal) {
                        o.voltage == Voltage.Nominal;

                        if (loaded == Load.Nominal) {
                            loadOut == Load.Nominal;
                        }
                    } else {
                        o.voltage == Voltage.Low;
                        loadOut == Load.Low;
                    }
                }
                CmdInverter.Off ->
                {
                    o.voltage == Voltage.Low;
                    loadOut == Load.Low;
                }
            }
        }
        HealthInverter.FailedOff ->
        {
            loadOut == Load.Low;
        }
    }
}
```

What is shown in the LYDIA excerpt below provides a very coarse abstraction of a position sensor. A sensor has two variables – one that shows the intended sensor reading and another one that shows the actual measurement. There is only one constraint in the model below. It specifies that if the sensor is healthy, what is displayed by the sensor should be equal to the actually measured value. The measured value is an observable and is supplied by the DXC framework (DXC sensor readings for position sensors are Booleans).

```
system PositionSensor(Position sensed)
{
    HealthPositionSensor h;

    attribute health(h) = cond (h) (
        HealthPositionSensor.Nominal -> true;
        HealthPositionSensor.Stuck -> false
    );
    attribute probability(h) = cond (h) (
        HealthPositionSensor.Nominal -> 0.9;
        HealthPositionSensor.Stuck -> 0.1
    );

    Position measured;
    attribute observable(measured) = true;

    switch (h) {
        HealthPositionSensor.Nominal -> { sensed == measured; }
        HealthPositionSensor.Stuck -> { }
    }
}
```

The rest of the ADAPT components are modeled in a similar way. The resulting model is very coarse-grained, and, as we will see in the following section, suffers from a number of weaknesses but it is simple and a subject of further development.

B.3 Experimental Results

The DXC'09 experiments allowed comparison of LYDIA to a number of DAs. The GDE [de Kleer and Williams, 1987] family includes NGDE [de Kleer, 2009], Wizards of Oz [Grastien and Kan-John, 2009], and the commercial RODON [Bunus et al., 2009]. Another commercial DA showing very good results is GoalArt. GoalArt is based on multilevel flow models [Larsson, 1996]. HyDE (Hybrid Diagnosis Engine) and HyDE-S [Narasimhan and Brownston, 2007] implement hybrid diagnosis and supports multiple reasoning algorithms such as constraint propagation, CDA* [Williams and Ragno, 2007], and others. FACT [Roychoudhury et al., 2009] is a model-based diagnosis system that uses hybrid bond graphs.

ProADAPT [Mengshoel, 2007] uses Bayesian systems models [Pearl, 1988] and an inference engine based on arithmetic circuits [Darwiche, 2003]. RulesRule is a rule-based isolation-only algorithm. Fault Buster is based on a combination of multivariate statistical methods, for the generation of residuals. RacerX is a detection-only algorithm which detects a percentage change in individual filtered sensor values to raise a fault detection flag. StanfordDA is an optimization-based approach for estimating fault states.

B.3.1 Additional ADAPT Metrics

In addition to the M_{err}, M_{utl}, M_{cpu}, and M_{mem} metrics (cf. Sec. 2.5), for ADAPT, we compute fault isolation time M_{fi} and a number of detection metrics (M_{fd}, M_{fn}, M_{fp}, and M_{da}). We next show how these new metrics are computed. For more information and a discussion on metrics, cf. Feldman et al. [2010].

The ADAPT experiments are performed with the help of the DXC'09 experimental framework, called DXF [Feldman et al., 2010]. Table B.3 summarizes the data collected by this framework for each scenario. These data are used for computing the various diagnostic metrics.

Table B.3: *Scenario execution summary data*

Variable	Type	Description
t_d	time-stamp	time of first detection
t_i	time-stamp	time of last isolation
$\omega^\star$	set	injected fault
$t^\star$	time-stamp	time of first fault injection
$t_i^\star$	time-stamp	time of injection of fault i
Ω	set of sets	candidate diagnoses
W	set of reals	candidate weights

The set $\Omega = \{\omega_1, \omega_2, \cdots, \omega_n\}$ contains all diagnoses computed by the DA at time t_i. If a DA never asserts the isolation signal I (i.e., $t_i = \infty$), it is assumed that $\Omega = \emptyset$. Each candidate in Ω is accompanied by a weight W. We denote the set of weights of all diagnoses in Ω as $W = \{W(\omega_1), W(\omega_2), \cdots, W(\omega_n)\}$. It is ensured that:

$$\sum_{\omega \in \Omega} W(\omega) = 1 \tag{B.1}$$

by dividing each original weight with the sum of all weights.

Fault Detection Time

The *fault detection time* (the reaction time for a diagnostic engine to detect an anomaly) is directly measured as:

$$M_{\text{fd}} = t_d \tag{B.2}$$

The fault detection time is reported in milliseconds and is computed only for non-nominal scenarios for which a DA asserts the time detection signal 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:

$$M_{\text{fn}} = \begin{cases} 1, & \text{if } t_d = \infty \\ 0, & \text{otherwise} \end{cases} \tag{B.3}$$

False Positive Scenario

The *false positive scenario* metric penalizes DAs which announce spurious faults and is defined as:

$$M_{\text{fp}} = \begin{cases} 1, & \text{if } t_d < t^\star \\ 0, & \text{otherwise} \end{cases} \tag{B.4}$$

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 have discussed in Sec. 2.5.

Scenario Detection Accuracy

The *scenario detection accuracy* metric is computed from M_{fn} and M_{fp}:

$$M_{\text{da}} = 1 - \max(M_{\text{fn}}, M_{\text{fp}}) \tag{B.5}$$

M_{da} is 1 if the scenario is true positive or true negative and 0 otherwise (equivalently, $M_{\text{da}} = 0$ if $M_{\text{fn}} = 1$ or $M_{\text{fp}} = 1$, and $M_{\text{da}} = 1$ 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).

Fault Isolation Time

Consider an injected fault $\omega^\star = \{c_1, c_2, \cdots, c_n\}$ with the individual component faults injected at times $T^\star = \langle t_1^\star, t_2^\star, \cdots, t_n^\star \rangle$. Next, from the isolation signal, we construct a sequence of isolation times for each component. This sequence containing time-stamps of rising edges of the isolation signal is denoted as T_i ($T_i = \langle t_1, t_2, \cdots, t_n \rangle$). Note that $t_k^\star < t_i$ for $1 \leq k \leq n$. The *fault isolation time* is then computed as:

$$M_{\text{fi}} = \frac{1}{n} \sum_{k=1}^{n} t_k - t_k^\star \tag{B.6}$$

If there is no isolation for specific fault (i.e., a fault is missed) then there is no difference $t_k - t_k^\star$ computed for that fault. E.g., if in a fault $\omega^\star = \langle c_1, c_2, c_3 \rangle$, c_1 is isolated, c_2 is not, and c_3 is; the isolation time $t_2 - t_2^\star$ is undefined and not included in the average ($n = 2$).

The *fault isolation time* is reported in milliseconds and is computed only for non-nominal scenarios for which a DA asserts the time isolation signal at least once.

System Metrics

The "per system" metrics $\bar{M}_{\text{fd}}$, $\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, and $\bar{M}_{\text{da}}$ are defined in a way analogous to $\bar{M}_{\text{utl}}$ (cf. Sec. 2.5.5).

Note that M_{fn}, M_{fp}, and M_{da} are called false negative scenario, false positive scenario and scenario detection accuracy, respectively. The analogous "per system" metrics $\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, and $\bar{M}_{\text{da}}$ are called *false negative rate*, *false positive rate*, and *detection accuracy*. $\bar{M}_{\text{da}}$, for example, represents the ratio of the number of correctly classified cases to the total number of cases. The latter "per system" metrics ($\bar{M}_{\text{fn}}$, $\bar{M}_{\text{fp}}$, and $\bar{M}_{\text{da}}$) are equivalent to the ones in Kurtoglu et al. [2009]. In this paper we first define each metric "per scenario" and then "per system".

B.3.2 Benchmarking Results

The DA benchmarking results for ADAPT-Lite are shown in Table B.4, with graphical depictions of some of the tabular data presented in Fig. B.3. No DA dominates over all metrics used in benchmarking; nine of eleven DAs tested are best or second best with respect to at least one of the metrics.

LYDIA is second in memory consumption and the average CPU time spent per scenario is approx. 1.5 s. With ADAPT-Lite, LYDIA/SAFARI show low false positive and false negative rate but large number of classification errors. The reason for that is the ambiguity (lack of constraints) in the ADAPT-Lite model. LYDIA does not discern between fault-detection and fault-isolation times, hence it is one of the fastest algorithms in these two metrics. One way to reduce the large number of classification errors without modifying the model is to use FRACTAL.

Table B.4: *ADAPT-Lite metrics*

	Detection				Isolation			Computation	
DA	$\bar{M}_{\text{fd}}$	$\bar{M}_{\text{fn}}$	$\bar{M}_{\text{fp}}$	$\bar{M}_{\text{da}}$	$\bar{M}_{\text{fi}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
FACT	1 785	0	0.11	0.89	10 798	11	0.975	15 815	4 271
Fault Buster	155	0.5	0.01	0.68	–	56	0.685	1 951	2 569
HyDE	13 355	0.46	0	0.72	13 841	45	0.79	23 418	5 511
HyDE-S	121	0.04	0.38	0.6	683	66	0.791	573	5 366
Lydia	232	0.18	0.01	0.88	232	100.3	0.785	1 410	1 861
NGDE	194	0.13	0.03	0.89	14 922	44.5	0.833	21 937	73 031
ProADAPT	4 732	0.05	0.01	0.96	7 104	10	0.955	1 905	1 226
RacerX	77	0.2	0.03	0.85	–	56	0.685	146	3 619
RODON	4 204	0.04	0.01	0.97	12 364	4	0.983	12 050	28 870
RulesRule	949	0.09	0.33	0.62	949	63	0.818	167	3 784
Wizards of Oz	12 202	0.5	0	0.7	12 327	43	0.769	1 153	1 682

As is evident from the definition of the metrics in Sec. 2.5, a DA that has low false positive and negative rates has high detection accuracy. False positives are counted in the following two situations: for nominal scenarios where the DA declares a fault; and for faulty scenarios where the DA declares a fault before any fault is injected. Noise in the data and incorrect models are the main causes of false positives.

The existing ADAPT Lydia model does not allow distinguishing between sensor-stuck and sensor-offset faults. In many scenarios, the sensor-stuck faults are set to the minimum or maximum value of the sensor or held at its last reading. The latter case presents difficulties to Lydia.

Note that $\bar{M}_{\text{fd}} \leq \bar{M}_{\text{fi}}$, hence the ADAPT-Lite and ADAPT plots of $\bar{M}_{\text{fd}}$ and $\bar{M}_{\text{fi}}$, shown in Fig. B.3, give the isolation time stacked on the detection time (assume that part of the time goes into detection first and then into isolation).

The empirical DA benchmarking results for ADAPT are shown in Table B.5. The four ADAPT plots in the bottom part of Fig. B.3 show (1) $\bar{M}_{\text{err}}$ by DA (top-left), (2) $\bar{M}_{\text{sry}}$ and $\bar{M}_{\text{dru}}$ by DA (top-right), (3) $\bar{M}_{\text{fd}}$ and $\bar{M}_{\text{fi}}$ by DA (bottom-left), and (4) $\bar{M}_{\text{fn}}$ and $\bar{M}_{\text{fp}}$ (bottom-right). Five of eight DAs tested are best or second best with respect to at least one of the metrics for ADAPT.

The low ADAPT diagnostic accuracy of Lydia is due to (1) no filtering of the noise in the sensor data and (2) the use of basic "connectivity" concepts in the model instead of modeling of electric laws from first principles. Many false positives result from nominal commanded mode changes in which the relay feedback did not change status as of the next data sample after the command.

The detection and isolation times are generally within the same order of magnitude for the different DAs (cf. the ADAPT $\bar{M}_{\text{da}}$ plot of Fig. B.3). Some DAs

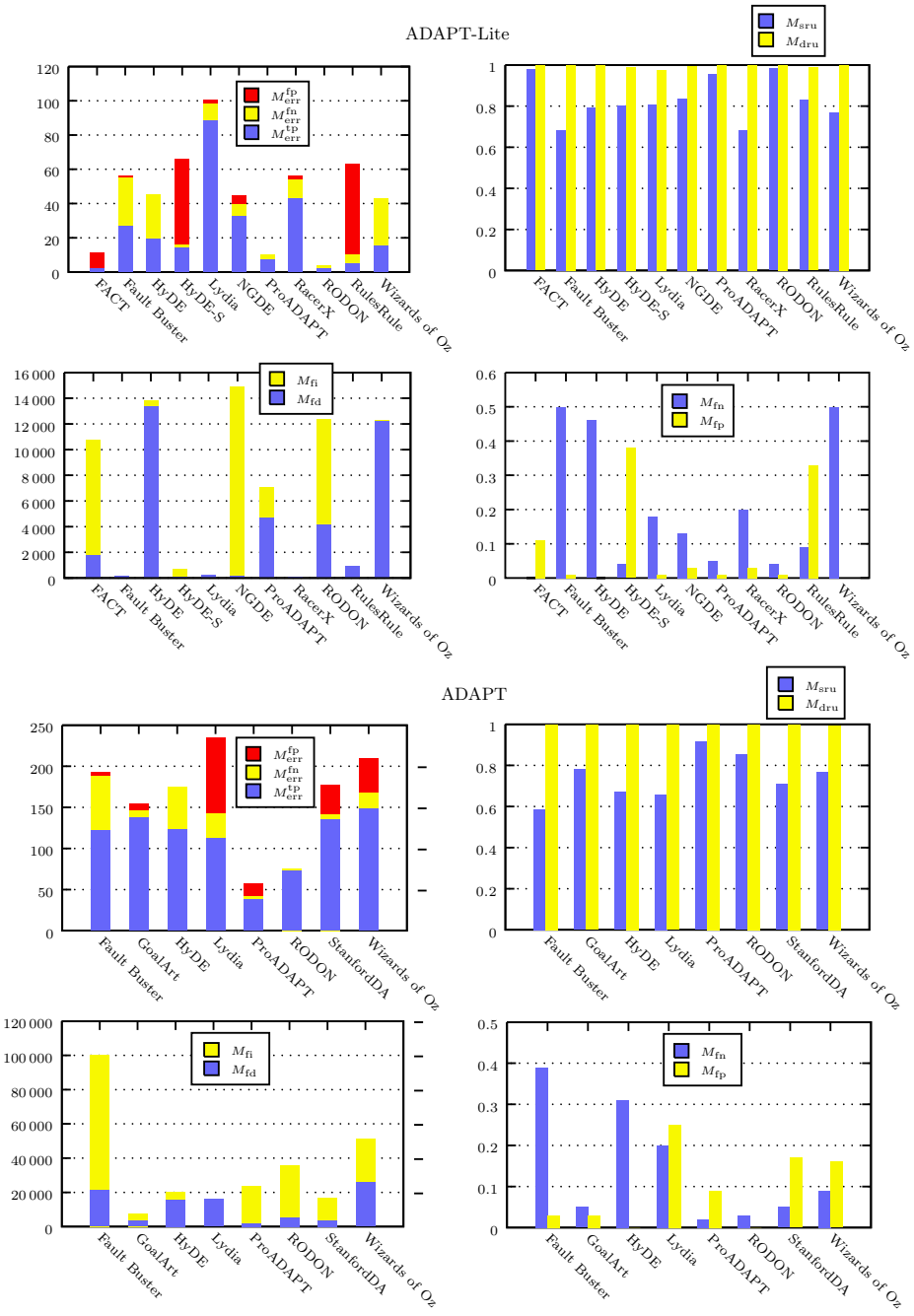


Figure B.3: *ADAPT-Lite and ADAPT metrics*

Table B.5: *ADAPT metrics results*

DA	Detection				Isolation			Computation	
	$\bar{M}_{\text{fd}}$	$\bar{M}_{\text{fn}}$	$\bar{M}_{\text{fp}}$	$\bar{M}_{\text{da}}$	$\bar{M}_{\text{fi}}$	$\bar{M}_{\text{err}}$	$\bar{M}_{\text{utl}}$	$\bar{M}_{\text{cpu}}$	$\bar{M}_{\text{mem}}$
Fault Buster	21 255	0.39	0.03	0.70	100 292	193	0.587	10 051	7 119
GoalArt	3 268	0.05	0.03	0.93	7 805	154	0.776	149	6 784
HyDE	15 612	0.31	0	0.79	20 114	174.3	0.668	28 807	19 135
LYDIA	16 135	0.2	0.25	0.62	16 135	234.9	0.653	5 715	3 412
ProADAPT	1 743	0.02	0.09	0.90	23 544	57	0.915	4 260	778
RODON	5 543	0.03	0	0.98	35 792	75.6	0.853	85 331	31 459
StanfordDA	3 826	0.05	0.17	0.79	16 816	176.6	0.706	1 012	2 213
Wizards of Oz	25 695	0.09	0.16	0.77	50 980	209.2	0.76	17 111	3 390

have isolation times that are similar to their detection times while others show isolation times that are much greater than the detection times. This could reflect differences in reasoning strategies or differences in policies for when to declare an isolation based on accumulated evidence.

The left plots of Figure B.4 show detection accuracy for all DAs by fault type for ADAPT-Lite and ADAPT. In general, $\bar{M}_{\text{da}}$ is not very sensitive to the component type, except in the case of load and sensor faults where it is lower. The data on the battery detection accuracy is not representative due to the limited number of fault scenarios containing battery faults (cf. Table B.2). It is visible from Fig. B.4 that LYDIA has lower diagnostic accuracy for sensor and load faults. The latter is due to the limited modeling of loads.

The right plots of Figure B.4 show classification errors for all DAs by fault type for ADAPT-Lite and ADAPT. The number of classification errors in sensor fault scenarios is quite high. Part of the reason, of course, is simply that there are more scenarios involving sensors that other fault types (cf. Table B.2). Another reason is due to the fact that most DAs reported either stuck or offset consistently or they reported both with equal weight.

The total number of classification errors is significantly larger in ADAPT compared to ADAPT-Lite (to see this, multiply $\bar{M}_{\text{err}}$ by the number of fault modes in each system). One of the reasons is because of the presence of multiple faults.

B.4 Summary

In this appendix we have shown the use of LYDIA/SAFARI for solving the ADAPT EPS diagnostic challenge. The results demonstrate that the LYDIA/SAFARI approach is applicable to models beyond digital circuits. SAFARI has outperformed many of the DAs in the ADAPT-Lite challenge. We have identified modeling as

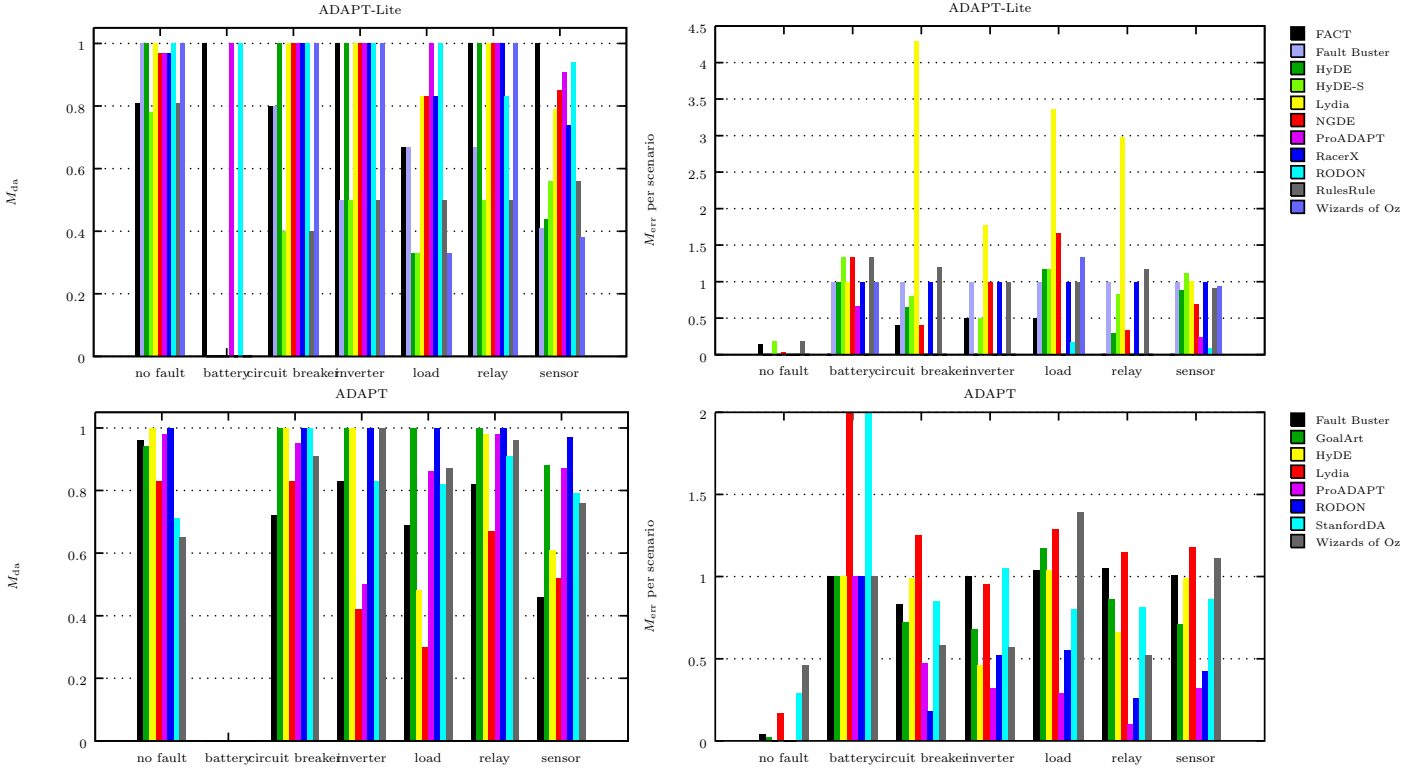


Figure B.4: *M_{da} and M_{err} per fault type for all DAs*

the main reason for the worse ADAPT results. The average running time of SAFARI for diagnosing ADAPT, a system with 723 variables and 1567 clauses, was below 6 s which makes SAFARI one of the fastest MBD algorithms. As a result of the coarse LYDIA model, SAFARI has computed large diagnostic sets which lead to an average of 1.48 classification errors per scenario. This diagnostic uncertainty can be improved by using the FRACTAL approach as illustrated in the appendix that comes next.

Appendix C

Case Study: Paper Input Module

In this appendix we demonstrate the applicability of FRACTAL to real-world applications, such as the diagnosis of a Paper Input Module (PIM), part of a heavy-duty printer. We show that (1) FRACTAL computes intuitive results given the PIM model and an initial observation and (2) FRACTAL leads to a steep reduction in the number of diagnoses even with very coarse-grained models and in the lack of prior failure probabilities.

C.1 System Overview

Before we describe the PIM we introduce a related printer modeling and diagnosis project. Figure C.1 shows the Tightly Integrated Parallel Printer (TIPP) fixture, devised by the Palo Alto Research Center (PARC) and used for studying MBD techniques and algorithms from, amongst others, planning, probing, testing, and control theory.

The PIM is similar to the TIPP fixture but smaller in size, thus enabling the development of a higher-precision model. The PIM, as shown in Fig. C.2, is used to supply paper to a printing module or to a finisher. Up to three PIM devices can be chained to supply paper to the printing unit and one PIM can be placed between the printer and the finisher to insert blank pages when requested.

A PIM has four paper trays which are placed vertically and numbered 1 to 4, starting from the top. Each tray can hold some amount of paper. The PIM has its own power supply. It receives commands via a Controller-Area Network (CAN) bus. A command can instruct the PIM to separate a sheet from tray n, horizontally transport a sheet (from a chained tray or from a manual feed), etc.

The PIM uses compressed air to separate the sheets of paper. The air is provided by an electrical fan. A dozen of solenoid valves (three per tray) control the air flow when a separation is requested. After a sheet of paper is separated,

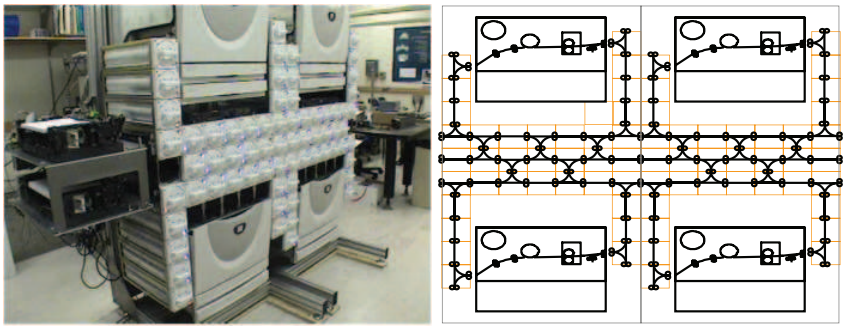


Figure C.1: *The TIPP fixture (left) and its schematics (right) (photo and diagram courtesy of PARC)*

it is transported vertically to the top-left part of the PIM (looked from the front) after which the sheet starts its horizontal journey to the output of the PIM.

Each tray (cf. Fig C.3) contains a motor which can elevate the paper stack up for separation and down for opening/refilling. There are two sensors which detect the up/down position of the paper stack. In addition to that there is a safety switch for the uppermost position which, under normal conditions, should never detect the tray as the motor should be switched-off after the up sensor reports the completion of an elevation operation.

In addition to the up and down sensors a paper tray has two paper position sensors (in reality the paper tray has also sensors for detecting the paper size but these are not included in the current model). These two position sensors detect a sheet of paper upon leaving the paper tray.

The third and fourth tray, in addition to separation, have a preseparation mode. When commanded to preseparate, the air-valves separate a sheet of paper, its transport starts, the preseparation clutch is connected but the paper is not placed on the belt. This is done to save time for the next separation operation as the sheet from the lower trays has to travel a longer distance in the vertical trajectory.

After a sheet of paper has been separated with compressed air, a clutch connects the transportation of the sheet from the tray to the vertical transport. In the vertical transport subsystem the sheet is placed on the vertical transportation belt (cf. Fig. C.2). The vertical transport has sensors for detecting the sheet leaving the tray. The belt in the vertical transport is driven by an electrical motor.

The same motor which drives the vertical transportation belt drives the horizontal transport. The horizontal transport has three sensors. A paper sensor detects a sheet of paper fed to the horizontal input (from a chained PIM or from a manual feed). Another sensor detects a sheet of paper coming from the vertical transport. Finally, the third sensor detects a sheet of paper leaving the device.

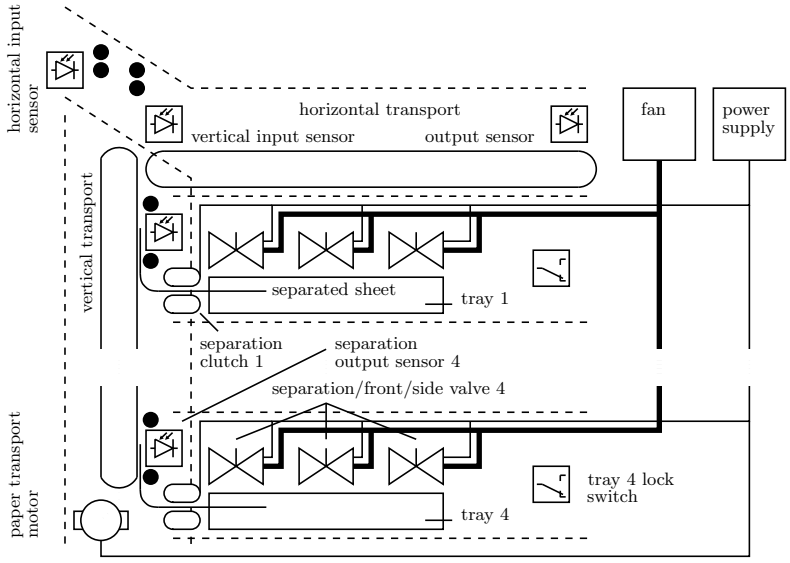


Figure C.2: *Schematic overview of a PIM*

In the top-level PIM module are placed the power supply and the fan. The air-duct and the solenoids controlling the air for separation are as well placed in the top-level PIM. Finally, there is a special sensor which detects double sheets.

C.2 PIM Model

We briefly discuss a qualitative PIM model (for a detailed description cf. [Feldman and van Gemund, 2007a]) that supports all queries in the FRACTAL algorithms. The model is built in the LYDIA language (cf. Appendix A). A LYDIA component model of a switch, for example, is shown in Fig. C.4. All PIM health variables are of the built-in Boolean type, with a positive literal denoting a healthy state for the respective component and a negative literal denoting the non-healthy state. Depending on the desired PIM action (e.g., separate a sheet from tray n or transport horizontally a sheet) each active PIM system is set to its respective state (motors can be on or off, clutches connected or disconnected, etc.). As we will see below, in each state a different set of constraints is applied for inferring the state of the system.

Table C.1 provides an overview of all component types which are used in the PIM model.

We model all the components as input/output devices. All components have an output of some (qualitative) type. Some of the components have one or more

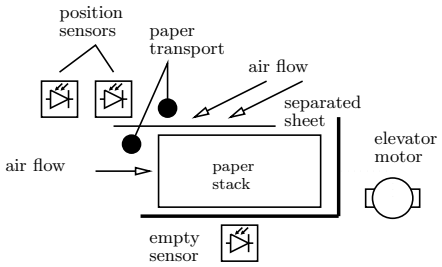


Figure C.3: *Schematic overview of a paper tray*

```
system Switch(switchValue actual)
{
    bool h;

    switchValue sensed;

    h => (sensed == actual);
}
```

Figure C.4: LYDIA *modeling example*

inputs. The data types for the inputs (if such exist) and outputs of the components are shown in the second and third columns of Table C.1. In some of the component models, the electrical power input is considered and this is noted in the fourth column of Table C.1. Two of the components (the active components) have more than one state depending on a control value. These sates are listed in the fifth column of Table C.1.

The rightmost column of Table C.1 shows the fault probability Pr for each component. Note that these probabilities are approximate [de Kleer, 1990] as they are not based on, for example, Failure Modes and Effects Analysis (FMEA) analysis. The LYDIA solvers use these probabilities for a more precise partial ordering of the result compared to diagnosis cardinality. These crude probabilities, however, reflect some human diagnostician experience in the likelihood of failure of different components.

We next briefly discuss all component models.

Sensor: We have marked all sensor readings as observable variables. The model stipulates that a sensor is healthy if the observed value is equal to the actual value. Strengthening techniques (e.g., strong-fault models) can reduce the uncertainty inherent to this very coarse-grained model.

Table C.1: *Overview of the PIM component models.*

Component	Input	Output	Powered	States	Pr
sensor	-	sensor value	no	-	0.05
switch	-	switch value	no	-	0.05
clutch	paper	paper	yes	connected, disconnected	0.01
motor	-	torque	no	-	0.01
power supply	-	power	no	-	0.01
fan	-	flow	yes	-	0.01
valve	flow	flow	yes	opened, closed	0.025

Switch: In the PIM abstraction, the LYDIA model of a switch is equivalent to the one of a sensor. The naming difference reflects the strict LYDIA typing system. All sensors and switches are passive components from the modeling viewpoint. They have only one healthy state, the subsystems which instantiate these sensors and switches provide constraints for the expected values depending on the mode of the PIM.

Clutch: The clutch is an active mechanical device with two states: "connected" and "disconnected". This state is determined by giving a value to a control variable. When a clutch is connected, a paper from the respective tray is fed to the vertical transport belt. A disconnected clutch does not result in a sheet of paper being transported.

In reality, a clutch has no input and output. Indeed, one way to model a clutch is to build a component model with state (e.g., if the clutch is healthy, and it is commanded "connect", and there is an input power it changes its state to "connected"). Such an approach, however, would necessitate the use of constraints in the containing subsystem (in this case the vertical transport) setting the presence of a sheet on the belt depending on the fact if a clutch is connected or not.

A simpler, though arguably physically wrong approach, is to represent a clutch with an input and an output. In this case when the clutch is healthy, commanded "connected", and powered, the output would be the same as the input, i.e., the clutch would transport a sheet of paper. Otherwise, if, for example the mechanism is jammed, there would not be a sheet at the output.

Valve: The PIM uses solenoid valves to direct air when separating sheets. The valve can be commanded to change its state to "opened" or "closed". If

the valve is healthy and it is commanded open, then the valve's input flow should equal the valve's output flow. Otherwise, if commanded closed (and the valve not leaking), there should be a lack of output flow. What follows is a model of a valve. Note that this model can be used for a wider variety of valves and devices.

Fan: The fan, needs an electrical power and if it is healthy and commanded on, it produces a positive air flow. A healthy fan, should produce a zero air flow if commanded off.

Motor: A nominally-functioning motor, if commanded on, translates electrical power into torque. The motor should produce a zero torque if commanded off.

Power supply: A special note is needed for the model of the power supply. In the case of the PIM (which lacks redundancy in the power supply) fault reasoning in the case of a fault in the power supply would be impossible to perform by the PIM itself (neither it would be possible to record observation data). In the PIM model the model of a power supply is extended to, for example, the machinery and cabling used for feeding power to the various components and subsystems. This way, the notion of a power failure can be extended to a failure in the power connector of a valve for example.

The power supply is simply a source of electrical power (we neglect the fact that the power supply is itself connected to the power grid and do not proceed with modeling of the power grid). There is no need to provide a power on-off switch, as we assume that the diagnosis will be done only when the main power supply is switched on and connected to the power main.

The PIM has four subsystems: tray, horizontal and vertical transports and a top-level system. Determining the scope of a subsystem for a model is always subjective and in this model we have tried to follow the mechanical and electrical design decomposition. This is different from the decomposition in the software design, for example, as decomposition decisions there may be driven by bandwidth of communication buses, locality of computational resources, etc.

Each subsystem participates in the paper transport, i.e., it contains mechanical devices, electronics, software, etc., involved in the sheet separation. Hence each subsystem provides at least one input, an output and a health variable. A sheet of paper may jam in the vertical transport for example. The probability of a failure for all the four subsystems is 0.005.

Table C.2 shows the components used by the different subsystems. There are 61 components in total. The seven subsystem instantiations have their own health variables, resulting in a total of 68 health variables. In total the model has 209 variables (of which 43 observables) and we have compiled it to a CNF representation with 605 clauses.

We briefly discuss all subsystems.

Table C.2: *Component instantiations*

Component	Instantiations	Instantiated By
sensor	28	PIM, horizontal & vertical transports, tray
switch	8	PIM, tray
clutch	6	vertical transport
motor	5	vertical transport, tray
power supply	1	PIM
fan	1	PIM
valve	12	PIM

Tray: The main function of a paper tray is to hold a stack of sheets. A paper tray also has a motor which elevates the paper stack up for separation, and down for reloading the tray. Up and down sensors and a safety limit switch control this motor. Two separation sensors per tray detect the presence of a sheet when separating. Finally, an empty sensor detects the presence of a paper stack in the tray. There are many aspects of the tray which we have omitted from the model. In reality, analog sensors determine the paper size, multiple LED indicators show the amount of paper in the tray, etc.

The paper tray is interfaced in the following manner. It receives power from the top-level PIM system. The PIM redistributes the power to its components (in particular to the elevator motor). When commanded to separate a single sheet of paper, the tray provides a sheet of paper to its output (given that the tray holds paper and no fault occurs). The Lydia excerpt below shows the component instantiations and the constraints governing the paper transport of a tray.

Depending on the input control command the tray has to set the states of all active components, it instantiates (active components in the PIM model are motors and clutches). This is done by the constraints that follow (the latter are normally part of the tray system but we show them separately for clarity).

Finally, each mode sets values for the tray sensors and switches. The expected sensor values for each mode are given by the Lydia constraints that follow. These constraints are, as well, part of the tray subsystem (we have discussed the control and sensor constraints outside of the the tray model for clarity).

Vertical transport: The vertical subsystem acts as a demultiplexer for moving a sheet of paper from a tray to the transport belt. After being separated

from the tray a sheet is placed on the belt and transported to the output of the vertical subsystem. The rotation of the belt is provided by an electrical motor, the same motor is used to rotate the belt in the horizontal system, hence the model supplies an output of the torque of the motor.

The constraints below control the state of the motor and the separation and pre-separation clutches depending on the value of the "cmdIn" variable.

The vertical subsystem contains multiple sensors for determining the position of the paper. The correct values of all the sensors (in each state) are set by the constraints that follow.

Horizontal transport: The horizontal transport connects the vertical transport (or, alternatively, another pre-stacked PIM) to the output of the PIM. The model of the horizontal transport subsystem is given below. The horizontal transport has three modes: idle, connecting the vertical transport, and connecting the pre-stacked PIM input. The horizontal transport (shown below) is, essentially, a multiplexor. It can transport a sheet either from the vertical transport or from the PIM input. There are sensors at both inputs of the subsystem and at the output.

As is evident from the LYDIA excerpt above, depending on the mode of the horizontal transport, it sets the actual sensor values. For example, if the PIM is idle (and the horizontal transport is idle respectively), then the three sensor which are in this subsystem should sense no paper.

The horizontal transport has its own health in addition to the health of the components. In practice, a failure of the sheet to reach the output of the PIM (horizontal transport) may occur due to a paper jam, due to a mechanical malfunction, etc. In this case the diagnostic system should label the horizontal transport as faulty. Note that this kind of modeling, adds uncertainty to the model, hence it decreases the diagnostic precision.

Top-level system: The top-level PIM system instantiates the horizontal and vertical transports and the four paper trays. It also holds a number of components, in particular the power supply, the fan and a dozen of air valves for separating the sheets. The control part of the PIM top-level system, similar to the horizontal and vertical subsystems, sets the modes of all the top-level subsystems and components.

The top-level system contains a double-sheet sensor and a number of mechanical safety-switches which prevent more than one tray to be ejected at a time. The latter feature ensures the mechanical stability of the PIM. The correct values of the double-sheet sensor and the switches are given by the constraints below.

Note that all the health variables in the PIM model are Boolean. This gives us a hybrid (Boolean health variables, multi-valued constraints) weak-fault model (or model with ignorance of abnormal behavior) for which the

Minimal-Diagnosis Hypothesis holds [de Kleer et al., 1992a]. As a result the subset minimal diagnoses of the PIM model fully-characterize a diagnostic solution.

C.3 Experimental Results

In the transcript that is shown below we can see the output from a SAFARI scenario in which the PIM is commanded to separate a sheet from tray 1 and the sheet is not sensed by a vertical transport sensor [Feldman and van Gemund, 2007a]. With the existing granularity of the model and the order-of-magnitude prior probabilities for the components, it is not possible to refine the candidate set to less than 7 diagnoses.

```
lydia> diag pim vout1_sensor_failure_separate_tray1
lydia> fm
d1 = { !verticalTransport.seOut[1].h }
d2 = { !separationValve[1].h }
d3 = { !sideBlowValve[1].h }
d4 = { !frontBlowValve[1].h }
d5 = { !fan.h }
d6 = { !h }
d7 = { !verticalTransport.h }
```

The next transcript shows the output of FRACTAL, given the initial observation from the above SAFARI scenario. In order to validate the experimental results, we need to know the actual fault. This information is missing in the original scenario (our scenarios are artificial as we had no access to a hardware or a software simulator or the physical device). Hence we first randomly choose a fault from the set of faults consistent with the observation. The actual fault allows us to verify the correctness of the implementation of Alg. 9 (cf. Chapter 5). In this example, a failed-off fan explains the initial observation.

Second, FRACTAL computes a control assignment which best decreases the number of expected MC diagnoses (in this case single-faults). The resulting control setting is a command to separate sheet from tray 2. FRACTAL's output is intuitive as in this case, (1) half of the single faults are ruled-out (i.e., they are not consistent with SD and the initial observation) and (2) the remaining 3 single-faults contain the original fault.

```
fractal> controls pim vout1_sensor_failure_separate_tray1
vout1_sensor_failure_separate_tray1: 7 initial diagnoses
injected = { !fan.h }
gamma_1 = { cmdIn = sepTray2 }
fractal> next pim vout1_sensor_failure_separate_tray1
d1 = { !fan.h }
d2 = { !h }
d3 = { !verticalTransport.h }
```

Note that in the PIM model there are many dependencies between the control variables, e.g., a command to separate a sheet from tray n implies a command to switch the fan on, a separate command setting the position of each valve, etc. If we require FRACTAL to compute a position for each of those low-level controls, FRACTAL would be unable to compute a control assignment decreasing the number of diagnoses due to its greedy nature. In practice however, controls are either independent given the space of possible diagnoses, or they are organized in a hierarchical fashion as in our example.

```
lydia> diag pim double_sensor_failure_separate_tray3
lydia> fm
d1 = { !tray[3].sePos2.h, !verticalTransport.seOut[3].h }
d2 = { !tray[3].sePos2.h, !sideBlowValve[3].h }
d3 = { !tray[3].sePos2.h, !frontBlowValve[3].h }
d4 = { !tray[3].sePos2.h, !separationValve[3].h }
d5 = { !tray[3].sePos2.h, !fan.h }
d6 = { !tray[3].sePos2.h, !h }
d7 = { !tray[3].sePos2.h, !verticalTransport.h }
```

The above scenario shows an initial-observation leading to 7 double-faults. As it is visible from the transcript shown below, FRACTAL^{G} computes a control assignment that optimally reduced the initial candidate space. This control assignment is a switch from tray 3 to tray 1. Continuing the scenario, we set that printing from tray 1 results in a nominal observation, which decreases the original set of 7 double-faults to 4 double-faults (in the transcript below, this is after the "next" command). The result cannot be further improved without adding new sensors. This example shows that FRACTAL works with faults of arbitrary multiple cardinality.

```
fractal> controls pim double_sensor_failure_separate_tray3
double_sensor_failure_separate_tray3: 7 initial diagnoses
injected = { !tray[3].sePos2.h, !separationValve[3].h }
gamma_1 = { cmdIn = sepTray1 }
fractal> next pim double_sensor_failure_separate_tray3
d1 = { !tray[3].sePos2.h, !verticalTransport.seOut[3].h }
d2 = { !tray[3].sePos2.h, !sideBlowValve[3].h }
d3 = { !tray[3].sePos2.h, !frontBlowValve[3].h }
d4 = { !tray[3].sePos2.h, !separationValve[3].h }
```

C.4 Summary

In this appendix we have shown the use of LYDIA and FRACTAL for solving PIM active testing problems. FRACTAL^{G} has computed control assignments that reduce initial candidate sets containing 7 diagnoses each, to candidate sets containing MC diagnoses of higher cardinality only. The running time of FRACTAL^{G} was negligible–less than 1 s per control assignment. Furthermore, this appendix demonstrated that FRACTAL is insensitive to the cardinality of the initial minimal-cardinality diagnoses. All control assignments computed by FRACTAL^{G} were

optimal, i.e., no further reduction in the number of initial diagnoses was possible. All this shows that FRACTAL^{G} effectively reduces the diagnostic uncertainty in real-world models with large number of variables, under-constrained models such as weak-fault models, or models of sensor-lean systems.

Bibliography

Ashraf M. Abdelbar. Approximating cost-based abduction is NP-hard. *Artificial Intelligence*, 159(1-2):231–239, 2004.

Ashraf M. Abdelbar, Sarah H. Gheita, and Heba A. Amer. Exploring the fitness landscape and the run-time behaviour of an iterated local search algorithm for cost-based abduction. *Experimental & Theoretical Artificial Intelligence*, 18(3): 365–386, 2006.

Miron Abramovici. A maximal resolution guided-probe testing algorithm. In *Proc. DAC'81*, pages 189–195, 1981.

Rui Abreu. *Spectrum-Based Fault Localization in Embedded Software*. PhD thesis, Delft University of Technology, 2009.

Rui Abreu, Peter Zoeteweij, and Arjan van Gemund. A new bayesian approach to multiple intermittent fault diagnosis. In *Proc. IJCAI'09*, pages 653–658, 2009.

Rajeev Alur, Costas Courcoubetis, and Mihalis Yannakakis. Distinguishing tests for nondeterministic and probabilistic machines. In *Proc. ACM Symposium on Theory of Computing*, pages 363–372, 1995.

Josep Argelich, Chu Min Li, and Felip Manyá. An improved exact solver for partial Max-SAT. In *Proc. NCP'07*, pages 230–231, 2007.

Mihiar Ayoubi. Fuzzy systems design based on a hybrid neural structure and application to the fault diagnosis of technical processes. *Control Engineering Practice*, 4(1):35–42, 1996.

James Bailey and Peter J. Stuckey. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proc. PADL'05*, pages 174–186, 2005.

Roberto J. Bayardo and Joseph Daniel Pehoushek. Counting models using connected components. In *Proc. AAAI'00*, pages 157–162, 2000.

Marion Behrens, Gregory Provan, Menouer Boubekeur, and Alie Mady. Model-driven diagnostics generation for industrial automation. In *Proc INDIN'09*, pages 708–714, 2009.

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.

Franc Brglez, David Bryan, and Krzysztof Koźmiński. Combinational profiles of sequential benchmark circuits. In *Proc. ISCAS'89*, volume 3, pages 1929–1934, 1989.

Mark Brodie, Irina Rish, Sheng Ma, and Natalia Odintsova. Active probing strategies for problem diagnosis in distributed systems. In *Proc. IJCAI'03*, pages 1337–1338, 2003.

Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley, 1984.

Peter Bunus, Olle Isaksson, Beate Frey, and Burkhard Münker. RODON - a model-based diagnosis approach for the DX diagnostic competition. In *Proc. DX'09*, pages 423–430, 2009.

Michael L. Bushnell and Vishwani D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000.

Tom Bylander, Dean Allemang, Michael Tanner, and John Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49:25–60, 1991.

Eugene Charniak and Solomon Eyal Shimony. Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66(2):345–374, 1994.

SAE Aerospace Propulsion Systems Health Management Committee E-32. Health and usage monitoring metrics, monitoring the monitor. Technical Report ARP5783, February 2008.

Luca Console and Pietro Torasso. A spectrum of logical definitions of model-based diagnosis 1. *Computational Intelligence*, 7(3):133–141, 1991.

Luca Console, Daniele Theseider Dupre, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic Computation*, 1(5):661–690, 1991.

Luca Console, Claudia Picardi, and Marina Ribaudo. Diagnosis and diagnosability analysis using PEPA. In *Proc. ECAI'00*, pages 131–135, 2000.

Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. STOC'71*, pages 151–158, 1971.

Fulvio Corno, Paolo Prinetto, Maurizio Rebaudengo, and Matteo Sonza Reorda. New static compaction techniques of test sequences for sequential circuits. In *Proc. EDTC'97*, pages 1–37, 1997.

Alexander Czutro, Ilia Polian, Piet Engelke, Sudhakar M. Reddy, and Bernd Becker. Dynamic compaction in SAT-based ATPG. In *Proc. ATS'09*, pages 187–190, 2009.

Adnan Darwiche. A differential approach to inference in Bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003.

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

Adnan Darwiche and Pierre Marquis. A knowlege compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

Nando de Freitas. Rao-blackwellised particle filtering for fault diagnosis. In *Proc. AEROCONF'02*, volume 4, pages 1767–1772, 2002.

Johan de Kleer. An improved approach for generating Max-Fault Min-Cardinality diagnoses. In *Proc. DX'08*, pages 247–252, 2008.

Johan de Kleer. Minimum cardinality candidate generation. In *Proc. DX'09*, pages 397–402, 2009.

Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127–162, 1986a.

Johan de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28(2): 197–224, 1986b.

Johan de Kleer. Using crude probability estimates to guide diagnosis. *Artificial Intelligence*, 45(3):381–291, 1990.

Johan de Kleer and Brian Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

Johan de Kleer, Alan Mackworth, and Raymond Reiter. Characterizing diagnoses and systems. *Artificial Intelligence*, 56(2-3):197–222, 1992a.

Johan de Kleer, Olivier Raiman, and Mark Shirley. One step lookahead is pretty good. In *Readings in Model-Based Diagnosis*, pages 138–142. Morgan Kaufmann Publishers, San Francisco, 1992b.

Somnath Deb, Sudipto Ghoshal, Venkata N. Malepati, and David L. Kleinman. Tele-diagnosis: Remote monitoring of large-scale systems. In *Proc. AEROCONF'00*, volume 6, pages 31–42, 2000.

Richard O. Duda and Edward H. Shortliffe. Expert systems research. *Science*, 220(4594):261–268, 1983.

Charles Ebeling. *An Introduction to Reliability and Maintainability Engineering.* McGraw-Hill, 1997.

Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proc. SAT'03*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

Michael Esser and Peter Struss. Fault-model-based test generation for embedded software. In *Proc. IJCAI'07*, 2007.

Alexander Feldman and Arjan van Gemund. A two-step hierarchical algorithm for model-based diagnosis. In *Proc. AAAI'06*, 2006.

Alexander Feldman and Arjan van Gemund. Building a LYDIA model of an océ printer's paper input module. Technical Report TUD-SERG-2007-16, Delft University of Technology, 2007a.

Alexander Feldman and Arjan van Gemund. LYDIA user guide. Technical Report ES-2009-05, Delft University of Technology, 2007b.

Alexander Feldman and Arjan van Gemund. Reducing the diagnostic uncertainty of a paper input module by active testing. Technical Report ES-2009-04, Delft University of Technology, 2009.

Alexander Feldman, Jurryt Pietersma, and Arjan van Gemund. A multi-valued SAT-based algorithm for faster model-based diagnosis. In *Proc. DX'06*, 2006.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. Generating manifestations of max-fault min-cardinality diagnoses. In *Proc. DX'07*, pages 83–90, 2007a.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. In *Proc. SARA'07*, pages 139–154, 2007b.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. A framework and algorithm for model-based active testing. In *Proc. PHM'08*, 2008a.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proc. AAAI'08*, pages 919–924, 2008b.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing observation vectors for max-fault min-cardinality diagnoses. In *Proc. AAAI'08*, pages 911–918, 2008c.

Alexander Feldman, Gregory Provan, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund. Automated redesign with the General Redesign Engine. In *Proc. SARA'09*, 2009a.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. FRACTAL: Efficient fault isolation using active testing. In *Proc. IJCAI'09*, 2009b.

Alexander Feldman, Gregory Provan, and Arjan van Gemund. Solving strong-fault diagnostic models by model relaxation. In *Proc. IJCAI'09*, 2009c.

Alexander Feldman, Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund. Empirical evaluation of diagnostic algorithm performance using a generic framework. *International Journal of Prognostics and Health Management*, 2010. Submitted for Review.

Amir Fijany, Farrokh Vatan, Anthony Barrett, Mark James, Colin Williams, and Ryan Mackey. A novel model-based diagnosis engine: Theory and applications. In *Proc. IEEE Aerospace'03*, volume 2, pages 901–910, 2003.

Kenneth Forbus and Johan de Kleer. *Building Problem Solvers*. MIT Press, 1993.

Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

Jon W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

Eugene C. Freuder, Rina Dechter, Matthew L. Ginsberg, Bart Selman, and Edward P. K. Tsang. Systematic versus stochastic constraint satisfaction. In *Proc. IJCAI'95*, volume 2, pages 2027–2032, 1995.

Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Physical impossibility instead of fault models. In *Proc. AAAI'90*, pages 331–336, 1990.

Ian P. Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proc. AAAI'93*, pages 28–33, 1993.

Ian P. Gent and Toby Walsh. The SAT phase transition. In *Proc. ECAI'94*, pages 105–109, 1994.

Enrico Giunchiglia and Marco Maratea. Solving optimization problems with DLL. In *Proc. ECAI'06*, pages 377–381, 2006.

Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. In *Proc. IJCAI'07*, pages 2293–2299, 2007.

Alban Grastien and Priscilla Kan-John. Wizards of oz – description of the 2009 DXC entry. In *Proc. DX'09*, pages 409–413, 2009.

Jun Gu and Ruchir Puri. Asynchronous circuit synthesis with boolean satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 14(8):961–973, Aug 1995.

Walter Hamscher and Randall Davis. Diagnosing circuits with state: An inherently underconstrained problem. In *Porc. AAAI'84*, pages 142–147, 1984.

Mark Hansen, Hakan Yalcin, and John Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test*, 16(3):72–80, 1999.

Frederick Hayes-Roth. Rule-based systems. *Communications of the ACM*, 28(9): 921–932, 1985.

David Heckerman, John S. Breese, and Koos Rommelse. Decision-theoretic troubleshooting. *Communications of the ACM*, 38(3):49–57, 1995.

Stefan Heinz and Martin Sachenbacher. Using model counting to find optimal distinguishing tests. In *Proc. of COUNTING'08*, pages 91–106, 2008.

Ulrich Heller and Peter Struss. G^+DE - the generalized diagnosis engine. In *Proc. DX'01*, pages 79–86, 2001.

Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

Miki Hermann and Reinhard Pichler. Counting complexity of propositional abduction. In *Proc. IJCAI'07*, pages 417–422, 2007.

Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

Holger Hoos. SAT-encodings, search space structure, and local search performance. In *Proc. IJCAI'99*, pages 296–303, 1999.

Holger Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers Inc., 2004.

Joseph L. A. Hughes. Multiple fault detection using single fault test sets. *IEEET-CAD*, 7(1):100–108, 1988.

Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *Proc. CP'02*, pages 233–248, 2002.

Oscar H. Ibarra and Sartaj K. Sahni. Polynomially complete fault detection problems. *IEEE Trans. on Computers*, 24(3):242–249, 1975.

Rolf Isermann. Supervision, fault-detection and fault-diagnosis methods – an introduction. *Control Engineering Practice*, 5(5):639–652, 1997.

HoonSang Jin, HyoJung Han, and Fabio Somenzi. Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit. In *Proc. TACAS'05*, pages 287–300, 2005.

L. Karin, R. Lunde, and B. Münker. Model-based failure analysis with RODON. In *Proc. ECAI'06*, 2006.

Kalev Kask and Rina Dechter. Stochastic local search for Bayesian networks. In *Proc. AISTAT'99*, pages 113–122, 1999.

Alex Kean and George K. Tsiknis. Clause management systems. *Computational Intelligence*, 9:11–40, 1993.

Mark W. Krentel. The complexity of optimization problems. In *Proc. STOC'86*, pages 69–76, 1986.

Ken Kubiak and W. Kent Fuchs. Multiple-fault simulation and coverage of deterministic single-fault test sets. In *Proc. of the IEEE International Test Conference*, pages 956–962, 1991.

Lukas Kuhn, Bob Price, Johan de Kleer, Minh Do, and Rong Zhou. Pervasive diagnosis: Integration of active diagnosis into production plans. In *Proc. DX'08*, pages 106–119, 2008.

T. K. Satish Kumar. A model counting characterization of diagnoses. In *Proc. DX'02*, pages 70–76, 2002.

O. Erhun Kundakcioglu and Tonguç Ünlüyurt. Bottom-up construction of minimum-cost and/or trees for sequential fault diagnosis. *IEEE Trans. on SMC*, 37 (5):621–629, 2007.

Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, and Alexander Feldman. First international diagnosis competition - DXC'09. In *Proc. DX'09*, pages 383–396, 2009.

Jan Eric Larsson. Diagnosis based on explicit means-end models. *Artificial Intelligence*, 80(1), 1996.

Wen-Shing Lee, Doris Grosh, Frank A. Tillman, and Chang H. Lie. Fault tree analysis, methods, and applications – a review. *IEEE Trans. on Reliability*, R-34(3):194–202, 1985.

Mark H. Liffiton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In *Proc. SAT'05*, pages 173–186, 2005.

Karin Lunde, Rüdiger Lunde, and Burkhard Münker. Model-based failure analysis with RODON. In *Proc. ECAI'06*, 2006.

Panagiotis Manolios and Daron Vroon. Efficient circuit to CNF conversion. In *Proc. SAT'07*, pages 4–9, 2007.

João P. Marques-Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proc. EPIA'99*, pages 62–74, 1999.

David A. McAllester. Truth maintenance. In *Proc. AAAI'90*, volume 2, pages 1109–1116, 1990.

Ole Mengshoel. Designing resource-bounded reasoners using Bayesian networks: System health monitoring and diagnosis. In *Proc. DX'07*, pages 330–337, 2007.

Tsoline Mikaelian, Brian C. Williams, and Martin Sachenbacher. Model-based monitoring and diagnosis of systems with software-extended behavior. In *Proc. AAAI'05*, pages 327–333, 2005.

Gordon E. Moore. Cramming more components onto integrated circuits. pages 56–59, 2000.

Igor Mozetič and Christian Holzbaur. Controlling the complexity in model-based diagnosis. *Annals of Mathematics and Artificial Intelligence*, 11(1):297–314, 1994.

Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

Sriram Narasimhan. *Model-Based Diagnosis of Hybrid Systems*. PhD thesis, Vanderbilt University, 2002.

Sriram Narasimhan and Lee Brownston. HyDE - a general framework for stochastic and hybrid modelbased diagnosis. In *Proc. DX'07*, pages 162–169, 2007.

Christos H. Papadimitriou. On selecting a satisfying truth assignment. In *Proc. FOCS'91*, pages 163–169, 1991.

James D. Park. MAP complexity results and approximation methods. In *Proc. UAI'02*, pages 388–396, 2002.

Krishna Pattipati and Mark Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Trans. on SMC*, 20(4): 872–887, 1990.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA, USA, 1988.

Jurryt Pietersma and Arjan van Gemund. Temporal versus spatial observability in model-based diagnosis. *SMC*, 6:5325–5331, 2006.

Jurryt Pietersma and Arjan van Gemund. Benefits and costs of model-based fault diagnosis for semiconductor manufacturing equipment. In *Proc. INCOSE'07*, pages 1–12, June 2007.

Scott Poll, Ann Patterson-Hine, Joe Camisa, David Garcia, David Hall, Charles Lee, Ole Mengshoel, Christian Neukom, David Nishikawa, John Ossenfort, Adam Sweet, Serge Yentus, Indranil Roychoudhury, Matthew Daigle, Gautam Biswas, and Xenofon Koutsoukos. Advanced diagnostics and prognostics testbed. In *Proc. DX'07*, pages 178–185, 2007.

Mukul R. Prasad, Philip Chong, and Kurt Keutzer. Why is ATPG easy. In *Proc. DAC'99*, pages 22–28, 1999.

Gregory Provan. Test generation for model-based diagnosis. In *Proc. ECAI'08*, pages 199–203, 2008.

Gregory Provan. Model abstractions for diagnosing hybrid systems. In *Proc. DX'09*, pages 321–328, 2009.

Gregory Provan and Jun Wang. Automated benchmark model generators for model-based diagnostic inference. In *Proc. IJCAI'07*, pages 513–518, 2007.

Vijaya Raghavan, Mojdeh Shakeri, and Krishna Pattipati. Optimal and near-optimal test sequencing algorithms with realistic test models. *IEEE Trans. on SMC*, 29(1):11–26, 1999.

Anavai Ramesh, George Becker, and Neil Murray. CNF and DNF considered harmful for computing prime implicants/implicates. *Journal of Automated Reasoning*, 18(3):337–356, 1997.

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

Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82 (1-2):273–302, 1996.

J. Paul Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of R & D*, 10:278–291, 1966.

Indranil Roychoudhury, Gautam Biswas, and Xenofon Koutsoukos. Designing distributed diagnosers for complex continuous systems. *IEEE Trans. on Automation Science and Engineering*, 6(2):277–290, 2009.

Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete event systems. In *Proc. on ICAOS'94*, volume 199, pages 73–79, 1994.

Tian Sang, Paul Beame, and Henry A. Kautz. A dynamic approach for MPE and weighted MAX-SAT. In *Proc. IJCAI'07*, pages 173–179, 2007.

Eugene Santos Jr. A linear constraint satisfaction approach to cost-based abduction. *Artificial Intelligence*, 65(1):1–28, 1994.

Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: a compendium. *SIGACT News*, 2002.

Eugene F. Schuster and William R. Sype. On the negative hypergeometric distribution. *International Journal of Mathematical Education in Science and Technology*, 18(3):453–459, 1987.

Bart Selman and Hector J. Levesque. Abductive and default reasoning: a computational core. In *Proc. AAAI'90*, pages 343–348, 1990.

Mojdeh Shakeri. *Advances in System Fault Modeling and Diagnosis*. PhD thesis, University of Connecticut, 1996.

Mojdeh Shakeri, Vijaya Raghavan, Krishna R. Pattipati, and Ann Patterson-Hine. Sequential testing algorithms for multiple fault diagnosis. *IEEE Trans. on SMC*, 30(1):1–14, Jan 2000.

Sajjad Siddiqi and Jinbo Huang. Hierarchical diagnosis of multiple faults. In *Proc. IJCAI'07*, pages 581–586, 2007.

Alexander Smith, Andreas Veneris, and Anastasios Viglas. Design diagnosis using Boolean satisfiability. In *Proc. ASP-DAC'04*, pages 218–223, 2004.

Alexander Smith, Andreas Veneris, Moayad Fahim Ali, and Anastasios Viglas. Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 24(10):1606–1621, 2005.

Paul Stephan, Robert Brayton, and Alberto Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(9):1167–1176, 1996.

Peter Struss. Testing physical systems. In *Proc. AAAI'94*, pages 251–256, 1994.

Peter Struss and Oskar Dressler. Physical negation: Integrating fault models into the general diagnosis engine. In *Proc. IJCAI'89*, pages 1318–1323, 1989.

Peter Struss and Oskar Dressler. "Physical negation" - integrating fault models into the General Diagnostic Engine. In *Readings in Model-Based Diagnosis*, pages 153–158. Morgan Kaufmann Publishers Inc., 1992.

Peter Struss, Andreas Malik, and Martin Sachenbacher. Qualitative modeling is the key to automated diagnosis. In *Proc. IFAC'96*, pages 43–48, 1996.

Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In *Proc. CP'04*, pages 663–678, 2004.

Dave A. D. Tompkins and Holger H. Hoos. UBCSAT: An implementation and experimentation environment for SLS algorithms for SAT and MAX-SAT. In *Proc. SAT'04*, pages 306–320, 2005.

Gregory Tseitin. On the complexity of proofs in propositional logics. In Jörg Siekmann and Graham Wrightson, editors, *Automation of Reasoning: Classical Papers in Computational Logic (1967–1970)*, volume 2. Springer-Verlag, 1983.

Fang Tu and Krishna Pattipati. Rollout strategies for sequential fault diagnosis. *IEEE Trans. on SMC*, 33(1):86–99, 2003.

Arjan van Gemund. The Lydia approach to diagnostic systems modeling. Technical Report PDS-2002-004, Delft University of Technology, 2002.

Arjan van Gemund. Lydia version 1.1 tutorial. Technical Report PDS-2003-001, Delft University of Technology, 2003.

John Waicukauski and Eric Lindbloom. Failure diagnosis of structured VLSI. *IEEE Design & Test of Computers*, 6(4):49–60, Aug 1989.

Brian Williams and Robert Ragno. Conflict-directed A* and its role in model-based embedded systems. *Journal of Discrete Applied Mathematics*, 155(12): 1562–1595, 2007.

Ramin Zabih and David McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proc. AAAI'88*, pages 155–160, 1988.

Hantao Zhang and Mark Stickel. Implementing the Davis-Putnam method. *JAR*, 24(1-2):277–296, 2000.

Hantao Zhang and Mark E. Stickel. An efficient algorithm for unit propagation. In *Proc. AI-MATH'96*, pages 166–169, 1996.

Summary

Approximation Algorithms for Model-Based Diagnosis

Aleksandar Feldman

Model-based diagnosis is an area of abductive inference that uses a system model, together with observations about system behavior, to isolate sets of faulty components (diagnoses) that explain the observed behavior, according to some minimality criterion. This thesis presents greedy approximation algorithms for three problems closely related to model-based diagnosis: (1) computation of cardinality-minimal diagnoses, (2) computation of max-fault min-cardinality observation vectors, and (3) computation of control assignments that optimally reduce the expected number of remaining cardinality-minimal diagnoses. All three problems are *NP*-hard or worse (for example, problem (1) is known to be $\Sigma_2^{\mathbf{P}}$-complete for arbitrary propositional models), thus making deterministic algorithms impractical for solving large models.

The three main algorithms of this thesis are SAFARI (StochAstic Fault diagnosis AlgoRIthm) for problem (1), MIRANDA (Max-fault mIn-caRdinAlity observatioN Deduction Algorithm) for problem (2), and FRACTAL$^{\mathrm{G}}$ (FRamework for ACtive Testing ALgorithms - Greedy) for problem (3). These three algorithms, the analysis of their properties, the empirical research about their performance, and the comparison to other approaches are the main theoretical contributions of this thesis.

SAFARI, MIRANDA, and FRACTAL$^{\mathrm{G}}$ employ approximation methods such as greedy stochastic search, stochastic sampling (in the case of FRACTAL$^{\mathrm{G}}$) and, as a result, efficiently solve worst-case *NP*-hard or worse problems by trading relatively small degradation in the optimality of the diagnostic results for much larger gain in the computational performance (in some cases several orders-of-magnitude

improvement in speed). This efficient trade-off is partly due to exploitation of specific properties in the search landscape of the diagnostic problems. All three algorithms achieve good optimality at low computational cost in diagnostic search problems that exhibit certain amount of continuity.

We have validated the theoretical claims related to SAFARI, MIRANDA, and FRACTAL^{G} with extensive experimentation on fault-models of 74XXX/ISCAS85 combinational circuits. These models are of variable size (19 – 3 512 components), different weakness (ignorance of abnormal behaviour, stuck-at), and different topology.

We show analytically that for a large class of models (weak-fault models), SAFARI can be configured to efficiently compute one subset-minimal diagnosis. In the general case (strong-fault model) and in the case of multiple diagnoses we show topologically-dependent probabilistic properties of SAFARI. For example, in the case of multiple diagnoses, SAFARI computes diagnoses of minimal-cardinality with probability which is negative exponential to the cardinality of the minimal-cardinality diagnosis.

SAFARI is computationally very efficient. For example, SAFARI computes a subset-minimal diagnosis in a weak-fault model of a digital circuit having more than 1 500 gates in less than 1 s and the memory footprint is less than 24 Mb of RAM. The average degradation in the optimality of the diagnoses computed by SAFARI is 13% only (averaged over all models and observations leading to a single or double-fault).

We have compared SAFARI to a range of deterministic and stochastic algorithms for computation of cardinality-minimal diagnoses. State-of-the-art deterministic algorithms such as HA* and CDA* are typically several orders-of-magnitude slower than SAFARI. Furthermore HA* and CDA* often time out with higher cardinality faults (and larger circuits) whereas SAFARI is not sensitive to the cardinality of the cardinality-minimal diagnosis.

SAFARI has resemblance to Max-SAT and to further empirically analyze the performance and optimality of SAFARI, we have implemented a method for computing cardinality-minimal diagnoses that uses a deterministic partial Max-SAT solver. In addition to that we have compared SAFARI to a diagnostic algorithm based on SLS Max-SAT. We have established that the performance of the complete Max-SAT and the optimality of the SLS-based Max-SAT degrade when increasing the circuit size or the cardinality of the faults. For example, a diagnosis algorithm based on W-MAXSATZ computed diagnoses with only 9.2% of the c880 observation vectors, whereas SAFARI always computes at least one nearly optimal diagnosis. Algorithms based on SLS-based Max-SAT, typically did not compute any diagnoses or computed very suboptimal diagnoses (for example one of the best performing Max-SAT algorithms, SAPS, found a single-fault diagnosis in c7552 after 77 264 restarts, while SAFARI computed the same diagnosis with 11 restarts only).

In this thesis, we propose an algorithm, MIRANDA, for creating a benchmark of "difficult" observation vectors for testing the performance and optimality of

diagnostic algorithms such as SAFARI. MIRANDA computes observation vectors leading to faults of high cardinality. These, so-called MFMC observation vectors, lead to diagnoses of cardinality close to the maximally distinguishable given the fault-model. For small combinational circuits (74XXX) the approximations of MIRANDA coincide with the optimal MFMC observation vectors (in terms of diagnostic cardinality) and these results are computed several orders of magnitude faster than exhaustive algorithms. With large circuits, MIRANDA computes an MFMC observation vector leading to a diagnosis of approximate cardinality of 36 in less than 6 min (c5315). The reason for the good performance of MIRANDA is the fact that it exploits continuity in the space of output assignments in a class of well-formed benchmark circuits.

MIRANDA searches the space of all possible observations, and, at each step, uses SAFARI to estimate the cardinality of the cardinality-minimal diagnosis. By varying the number of MIRANDA and SAFARI retries, we have empirically established lower bounds on the approximation error of MIRANDA. The theoretical analysis of the cardinality estimation optimality of SAFARI leads us to believe that the small lower bound on the error of MIRANDA is a good approximation of the overall approximation error and hence, the MFMC values and observation vectors computed by MIRANDA are nearly optimal.

FRACTAL$^{\text{G}}$ is an algorithm that, given a model and an observation leading to multiple minimal-cardinality diagnoses, computes a control assignment (a test) that optimally reduces the set of diagnoses. As the problem is computationally hard, we use greedy stochastic search and stochastic sampling. The result is a fast algorithm (computing a whole FRACTAL scenario takes between 1 s for 74182 and 15 min for c7552) that decreases the diagnostic uncertainty according to a near-geometric decay curve. The reason for the efficiency of FRACTAL$^{\text{G}}$ is that it exploits continuity in the space of control assignments. Experimentation shows that this is true even for a small number of control inputs.

FRACTAL$^{\text{G}}$ has achieved better exponential decay compared to alternative approaches, except exhaustive control search. The difference in the decay rate between FRACTAL$^{\text{G}}$ and exhaustive search for 74182 is 5.4%. The exhaustive control approach, however, takes minutes to complete even for 74182 and times out with any model having more than 20 controls.

We have implemented a tool kit for system modeling called LYDIA (Language for sYstem DIAgnosis). LYDIA is also the framework in which we have developed all the algorithms in this thesis. In addition to the implementations of SAFARI, MIRANDA, and FRACTAL$^{\text{G}}$, the LYDIA tool kit also contains model translators (e.g., to CNF, DNF, OBDDs, etc), utilities, and all reference algorithms.

We have applied LYDIA to, amongst others, (1) a model of the Electrical Power System (EPS) testbed in the ADAPT lab at NASA Ames Research Center, as a participant in the First International Diagnostic Competition (DXC'09), and (2) a Paper Input Module (PIM), part of a heavy-duty printer of Océ Technologies. The ADAPT EPS consists of electrical components only and the PIM model is a combination of electrical, mechanical and pneumatic ones.

The results from the applications of LYDIA to models of physical systems such as ADAPT and PIM are consistent with the results from experiments on models of digital circuits. For example, the average running time of SAFARI for diagnosing ADAPT, a system that is modeled with 723 variables and 1 567 clauses, was below 6 s which makes SAFARI one of the fastest MBD algorithms today.

Samenvatting

Approximation Algorithms for Model-Based Diagnosis

Aleksandar Feldman

Model-gebaseerde diagnose is een vorm van abductie die gebruik maakt van een systeemmodel en observaties waarme verzamelingen van defecte onderdelen (een diagnose) worden afgeleid die het waargenomen gedrag verklaren volgens een bepaald minimaliteitscriterium. Dit proefschrift presenteert "greedy" benaderingsalgoritmen voor drie problemen die betrekking hebben op model-gebaseerde diagnose: (1) berekening van diagnoses van minimale kardinaliteit, (2) berekening van observatievectoren overeenkomend met een gemaximaliseerd aantal fouten en een minimale kardinaliteit, en (3) berekening van de besturingsvector die het verwachte aantal resterende diagnoses van minimale kardinaliteit minimaliseert. Al deze problemen zijn *NP*-hard of erger (bijvoorbeeld probleem (1) is $\Sigma_2^{\mathbf{P}}$-compleet voor willekeurige propositionele modellen), waardoor het onmogelijk is om deterministische algoritmen te gebruiken voor het oplossen van grote modellen.

De drie belangrijkste algoritmen van dit proefschrift zijn SAFARI (StochAstic Fault diagnosis AlgoRIthm) voor Probleem (1), MIRANDA (Max-fault mIncaRdinAlity observatioN Deduction Algorithm) voor Probleem (2), en FRACTAL$^{\mathrm{G}}$ (FRamework for ACtive Testing ALgorithms - Greedy) voor Probleem (3). De belangrijkste theoretische bijdragen van dit proefschrift zijn deze drie algoritmen, de analyse van hun eigenschappen, het empirisch onderzoek naar hun prestaties, en de vergelijking met andere algoritmen.

SAFARI, MIRANDA, en FRACTAL$^{\mathrm{G}}$ maken gebruik van benaderingsmethoden zoals "greedy stochastic search" en stochastische bemonstering (in het geval van FRACTAL$^{\mathrm{G}}$). Hiermee kunnen ze worst-case *NP*-harde of ergere problemen efficiënt oplossen met een relatief kleine afbreuk van de optimaliteit van de diagnosti-

sche resultaten in te ruilen voor veel grotere winst in de rekenkundige prestaties (in sommige gevallen meerdere orden van grootte verbetering in snelheid). Deze efficiënte uitruil is deels mogelijk door het uitbuiten van specifieke eigenschappen in het zoeklandschap van de diagnostische problemen. Alle drie de algoritmen bereiken een goede optimaliteit tegen lage berekeningskosten in diagnostische optimaliseringsproblemen die een zekere mate van continuïteit vertonen.

Door middel van uitgebreide experimentering met foutmodellen van 74XXX en ISCAS85 combinatorische schakelingen zijn de theoretische claims met betrekking tot SAFARI, MIRANDA en FRACTAL^{G} gevalideerd. Deze modellen hebben variërende grootte (19 – 3 512 componenten), verschillende zwakheden (onbekendheid van abnormaal gedrag, "stuck-at") en verschillende topologiën.

We hebben analytisch aangetoond dat voor een grote klasse van modellen (de zwakke foutmodellen), SAFARI kan worden geconfigureerd om efficiënt een minimale subset diagnose te berekenen. Voor algemene (sterke foutmodellen) en meervoudige diagnosegevallen hebben we topologie-afhankelijke voorspelbare eigenschappen laten zien van SAFARI. In het geval van meervoudige diagnoses bijvoorbeeld, berekent SAFARI diagnoses met minimale kardinaliteit met een kans die negatief exponentioneel is ten opzichte van de kardinaliteit van de diagnose met minimale kardinaliteit.

SAFARI is zeer efficient in haar berekeningen. SAFARI berekent bijvoorbeeld een minimale subset diagnose in een zwak foutmodel van een digitaal circuit met meer dan 1 500 poorten in minder dan 1 s met een geheugengebruik van minder dan 24 MB. De gemiddelde optimaliteitsvermindering van de diagnoses berekend door SAFARI is slechts 13 % (gemiddeld over alle modellen en observaties die leiden tot een enkele - of dubbele fout).

We hebben SAFARI vergeleken met een aantal deterministische en stochastische algoritmen voor de berekening van kardinaliteit-minimale diagnoses. Moderne deterministische algoritmen, zoals HA* of CDA* zijn meestal enkele orden van grootten minder efficiënt dan SAFARI. Bovendien werken HA* en CDA* vaak niet bij hogere-kardinaliteit fouten (en grotere circuits), terwijl SAFARI vrijwel niet beïnvloed wordt door de kardinaliteit van de kardinaliteit/minimale diagnose.

SAFARI is vergelijkbaar met Max-SAT. Om de prestaties en optimaliteit van SAFARI empirisch te analyseren, hebben we een methode geïnplementeerd voor het berekenen van kardinaliteit-minimal diagnoses dat gebruik maakt van deterministische partiële Max-SAT solver. Daarnaast hebben we SAFARI vergeleken met een diagnostisch algoritme gebaseerd op SLS Max-SAT. We hebben vastgesteld dat de prestaties van de complete Max-SAT en de optimaliteit van de SLS-gebaseerde Max-SAT verminderen wanneer de circuit grootte of de kardinaliteit van de fouten stijgen. Een algoritme gebaseerd op W-MAXSATZ berekent bijvoorbeeld diagnoses met slechts 9.2 % van de c880 observatievectoren, terwijl SAFARI altijd ten minste een bijna optimale diagnose berekent. SLS-gebaseerde algoritmen voor Max-SAT berekenen meestal geen diagnoses of berekenen zeer suboptimale diagnoses (een van de best presterende Max-SAT algoritmen, SAPS, vond bijvoorbeeld een enkele-fault diagnose in c7552 na 77 264 herstarts, terwijl

SAFARI dezelfde diagnose berekent met slechts 11 herstarts.

In dit proefschrift stellen we een algoritme genaamd MIRANDA voor, voor de berekening van een benchmark van "moeilijk" observatievectoren voor het testen van de prestaties en optimaliteit van diagnostische algoritmen, zoals SAFARI. MIRANDA berekent observatievectoren die leiden tot fouten van hoge kardinaliteit. Deze zogenaamde MFMC observatievectoren leiden tot diagnoses van kardinaliteit dicht bij die van het gegeven foutmodel. Voor kleine combinatorische circuits (74XXX) vallen de benaderingen van MIRANDA samen met de optimale MFMC waarnemingsvectoren (in termen van diagnostische kardinaliteit) en worden deze resultaten diverse orden van grootte sneller berekend dan uitputtende algoritmen. Voor grote circuits berekent MIRANDA een MFMC observatievector die leidt tot een diagnose met kardinaliteit van ongeveer 36 in minder dan 6 min (c5315). De reden van de goede prestaties van MIRANDA is het feit dat zij de continuïteit in de ruimte van de output opdrachten uitbuit die bestaat in de klasse van welgevormde benchmarkcircuits.

MIRANDA doorzoekt de ruimte van alle mogelijke observaties, en gebruikt bij elke stap SAFARI voor de berekening van de kardinaliteit van de diagnose met minimale kardinaliteit. Door het aantal MIRANDA en SAFARI pogingen te variëren, hebben we empirisch ondergrenzen van de approximatiefout van MIRANDA vastgesteld. De theoretische analyse van de optimaliteit van de kardinaliteitsschatting van SAFARI geeft ons vertrouwen dat de kleine ondergrens van de fout van MIRANDA een goede benadering is van de globale benaderingsfout, en daarmee dat de door MIRANDA berekende MFMC waarden en observatievectoren bijna optimaal zijn.

FRACTAL^{G} is een algoritme dat, gegeven een model en een observatie die leiden tot een verzameling van meerdere diagnoses met minimale kardinaliteit, een besturingsvector (test) berekent die die verzameling optimaal reduceert. Aangezien het probleem berekeningstechnisch moeilijk is, gebruiken we een "greedy" stochastisch zoekalgoritme samen met een stochastische bemonstering. Het resultaat is een snel algoritme (een geheel FRACTAL scenario neemt tussen 1 s voor 74182 en 15 min voor c7552) dat de diagnostische onzekerheid vermindert volgens een bijna-geometrische afname-curve. De reden voor de efficiëntie van FRACTAL^{G} is dat het de continuïteit in de ruimte van besturingsvectoren uitbuit. Experimenten tonen aan dat dit zelfs waar is voor een klein aantal besturingsvariabelen.

FRACTAL^{G} bereikt een betere exponentiële afname dan alternatieve algoritmen, met uitzondering van "exhaustive search". Het verschil in afnamesnelheid tussen FRACTAL^{G} en exhaustive search voor 74182 is 5.4 %. Daarentegen heeft exhaustive search minuten nodig voor 74182 (het kleinste model) en bereikt het de tijdslimiet voor ieder model met meer dan 20 besturingsvariabelen.

We hebben een toolkit voor systeemmodellering geïmplementeerd, genaamd LYDIA (Language for sYstem DIAgnosis). LYDIA is ook het raamwerk waarbinnen alle algorithmen in deze thesis zijn ontwikkeld. Naast implementaties van SAFARI, MIRANDA, en FRACTAL^{G}, bevat de LYDIA toolkit ook modelvertalers (bv. naar CNF, DNF, OBDDs etc.), utilities, and alle referentie-algorithmen.

We hebben LYDIA onder meer toegepast op (1) een model van het Electrical Power System (EPS) testbed van het ADAPT laboratorium van NASA Ames Research Center, als deelnemer aan de First International Diagnostic Competition (DXC'09), en (2) een Paper Input Module (PIM), onderdeel van een industriële printer van Océ Technologies. Het ADAPT EPS bestaat uit enkele electrische componenten, terwijl het PIM model een combinatie is van electrische, mechanische, en pneumatische componenten.

De resulaten van de toepassing van LYDIA op modellen van fysieke systemen zoals ADAPT en PIM komen overeen met die van experimenten met modellen van digitale circuits. Bijvoorbeeld, de gemiddelde executietijd van SAFARI voor het diagnostiseren van ADAPT, een systeem dat is gemodelleerd met 723 variabelen en 1 567 clauses, blijft onder de 6 s, hetgeen SAFARI vandaag-de-dag tot een van de snelste MBD algoritmen maakt.

Curriculum Vitae

Personal Data

Date of Birth: September 17, 1977

Address: Willem van Aelststraat 9/2
2612 HR Delft
The Netherlands

Tel.: +31 15 2129502

Cellular: +31 612 973650

email: `alex@llama.gs`

URL: `http://llama.gs/`

Education

9/2005–5/2010 *Ph.D., Computer Science*
Delft University of Technology, The Netherlands
Thesis: *Approximation Algorithms for Model-Based Diagnosis*
Advisor: Prof. Arjan van Gemund

9/2002–9/2004 *M.Sc. (cum laude), Computer Science (Technical Informatics)*
Delft University of Technology, The Netherlands
Thesis: *Hierarchical Approach to Fault Diagnosis*
Advisor: Prof. Arjan van Gemund

9/1997–6/2000 *B.Sc., Computer Science*
UE Varna, Bulgaria

Employment

5/2008–9/2008	Visiting Researcher Intelligent Systems Laboratory, Embedded Reasoning Area Palo Alto Research Center (PARC), Inc. California, USA
9/2005–5/2010	Doctoral Research Fellow Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology The Netherlands
4/2005–9/2005	Software Architect Science and Technology BV, Delft, The Netherlands
9/2001–4/2005	Senior Programmer Market Risk Management, ING Bank, Amsterdam The Netherlands
7/2000–9/2001	Senior Programmer Zend Technologies Ltd., Ramat Gan, Israel

Professional Activities

Reviewer	Journal on Artificial Intelligence (AIJ) Journal of Universal Computer Science (JUCS) Journal on Systems, Man and Cybernetics (SMC) IEEE Transactions on Reliability (TREL) International Workshop on Principles of Diagnosis 2010 (DX'10) International Workshop on Principles of Diagnosis 2009 (DX'09) International Conference on Prognostics and Health Management 2008 (PHM'08)
Organizer	Second International Diagnostic Competition (DXC'10) First International Diagnostic Competition (DXC'09)

Teaching Experience

9/2007–1/2008	Model-Based Computing, Delft University of Technology Teaching assistant
2/2007–3/2007	Model-Checking, Delft University of Technology Teaching assistant

Scholarship and Prizes

M.Sc. cum laude

Best paper award at the First International Conference on Prognostics and Health Management 2008 (PHM'08)

Publications

Conferences

[1] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing multiple minimal diagnoses. In *Proceedings of the First Annual Conference of the Prognostics and Health Management Society (PHM'09), San Diego, California, USA*, September 2009.

[2] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Solving strong-fault diagnostic models by model relaxation. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09), Pasadena, California, USA*, July 2009.

[3] Alexander Feldman, Gregory Provan, and Arjan van Gemund. FRACTAL: Efficient fault isolation using active testing. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI'09), Pasadena, California, USA*, July 2009.

[4] Alexander Feldman, Gregory Provan, and Arjan van Gemund. A framework and algorithm for model-based active testing. In *Proceedings of the First International Conference on Prognostics and Health Management (PHM'08), Denver, Colorado, USA*, October 2008. **Best Student Paper Award.**

[5] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing observation vectors for max-fault min-cardinality diagnoses. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08), Chicago, Illinois, USA*, pages 911–918, July 2008.

[6] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of the Twenty-Third National Conference on Artificial Intelligence (AAAI'08), Chicago, Illinois, USA*, pages 919–924, July 2008.

[7] Alexander Feldman, Marco Caporicci, Oscar Gracia, and André Bos. Advances in intelligent health reasoning and its application to IBDM. In *Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, USA*, March 2007.

[8] Alexander Feldman, Jurryt Pietersma, and Arjan van Gemund. All roads lead to fault diagnosis: model-based reasoning with Lydia. In *Proceedings of the Eighteenth Belgium-Netherlands Conference on Artificial Intelligence (BNAIC'06), Namur, Belgium*, October 2006.

[9] Alexander Feldman and Arjan 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.

[10] Jurryt Pietersma, Alexander Feldman, and Arjan van Gemund. Modeling and Compilation Aspects of Fault Diagnosis Complexity. In *Proceedings of IEEE AUTOTESTCON'06, Anaheim, California, USA*, September 2006.

Workshops & Symposia

[1] Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, and Alexander Feldman. First international diagnosis competition - DXC'09. In *Proceedings of the Twentieth International Workshop on Principles of Diagnosis (DX'09), Stockholm, Sweden*, pages 383–396, June 2009.

[2] Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund, and Alexander Feldman. Towards a framework for evaluating and comparing diagnosis algorithms. In *Proceedings of the Twentieth International Workshop on Principles of Diagnosis (DX'09), Stockholm, Sweden*, pages 373–382, June 2009.

[3] Alexander Feldman, Gregory Provan, and Arjan van Gemund. The LYDIA approach to combinational model-based diagnosis. In *Proceedings of the Twentieth International Workshop on Principles of Diagnosis (DX'09), Stockholm, Sweden*, pages 403–408, June 2009.

[4] Alexander Feldman, Gregory Provan, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund. Automated redesign with the general redesign engine. In *Proceedings of the Twentieth International Workshop on Principles of Diagnosis (DX'09), Stockholm, Sweden*, pages 307–314, June 2009.

[5] Alexander Feldman, Gregory Provan, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund. Automated redesign with the general redesign engine. In *Proceedings of the Eighth Symposium on Abstraction, Reformulation, and Approximation (SARA'09), Lake Arrowhead, California, US*, July 2009.

[6] Alexander Feldman, Gregory Provan, and Arjan van Gemund. A framework and algorithm for model-based active testing. In *Proceedings of the Nineteenth International Workshop on Principles of Diagnosis (DX'08), Blue Mountains, Australia*, pages 71–78, September 2008.

[7] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. In *Proceedings of the Seventh Symposium on Abstraction, Reformulation, and Approximation (SARA'07), Whistler, Canada*, pages 139–154, July 2007.

[8] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. In *Proceedings of the Eighteenth International Workshop on Principles of Diagnosis (DX'07), Nashville, Tennessee, USA*, pages 290–297, May 2007.

[9] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Generating manifestations of max-fault min-cardinality diagnoses. In *Proceedings of the Eighteenth International Workshop on Principles of Diagnosis (DX'07), Nashville, Tennessee, USA*, pages 83–90, May 2007.

[10] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Interchange formats and automated benchmark model generators for model-based diagnostic inference. In *Proceedings of the Eighteenth International Workshop on Principles of Diagnosis (DX'07), Nashville, Tennessee, USA*, pages 91–98, May 2007.

[11] Alexander Feldman, Jurryt Pietersma, and Arjan van Gemund. A multi-valued SAT-based algorithm for faster model-based diagnosis. In *Proceedings of the Seventeenth International Workshop on Principles of Diagnosis (DX'06), Peñaranda de Duero, Burgos, Spain*, June 2006.

[12] Alexander Feldman, Arjan van Gemund, and André Bos. A hybrid approach to hierarchical fault diagnosis. In *Proceedings of the Sixteenth International Workshop on Principles of Diagnosis (DX'05), Monterey, California, USA*, pages 101–106, June 2005.

Technical Reports

[1] Alexander Feldman, Gregory Provan, and Arjan van Gemund. A family of model-based diagnosis algorithms based on Max-SAT. Technical Report ES-2009-02, Delft University of Technology, 2009.

[2] Alexander Feldman and Arjan van Gemund. Reducing the diagnostic uncertainty of a paper input module by active testing. Technical Report ES-2009-04, Delft University of Technology, 2009.

[3] Alexander Feldman and Arjan van Gemund. LYDIA user guide. Technical Report ES-2009-05, Delft University of Technology, 2007.

[4] Alexander Feldman and Arjan van Gemund. Building a LYDIA Model of an Océ Printer's paper input module. Technical Report TUD-SERG-2007-16, Delft University of Technology, 2007.

Journal Papers in Review

[1] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research*, 2010. Submitted for Review.

[2] Alexander Feldman, Gregory Provan, and Arjan van Gemund. Stochastic algorithms for sequential model-based diagnosis. *Journal of Artificial Intelligence Research*, 2010. Submitted for Review.

[3] Alexander Feldman, Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Johan de Kleer, Lukas Kuhn, and Arjan van Gemund. Empirical Evaluation of Diagnostic Algorithm Performance Using a Generic Framework. *International Journal of Prognostics and Health Management*, 2010. Submitted for Review.

TU Delft

ISBN 978-90-9025023-6