

Structure Exploitation in Diagnosis

Sajjad Ahmed Siddiqi



A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

August 2009

Structure Exploitation in Diagnosis

Sajjad Ahmed Siddiqi



A thesis submitted for the degree of

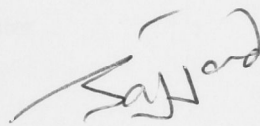
© Sajjad Ahmed Siddiqi

The Australian National University

Typeset in Palatino and Euler by T_EX and L^AT_EX 2_ε.

August 2009

Except where otherwise indicated, this thesis is my own original work.

A handwritten signature in black ink, appearing to read 'Sajjad', written in a cursive style.

Sajjad Ahmed Siddiqi

26 August 2009

Acknowledgements

In the name of Almighty Allah, the most wonderful and the most Merciful, who has allowed me to finish this work successfully. I pray to Him to show me the right path and grant success both in the world and the world hereafter.

My love and highest gratitude goes to my supervisor Sir. Iqbal Hussain, whose love and care was vital in the completion of this work, as well as his constant support and encouragement throughout during the time of research and writing.

Thanks go to my father, Amjad Ali, my mother, Mumtaz Begum, my brother, Iqbal, my sister, Ayesha, my friends, and my colleagues for providing much support and advice at various stages of this work.

To
My Father:
Muhammad Siddiq
and
My Mother:
Mamoon Akhter

My special thanks go to my family, especially my mother, who has always supported me in every way possible. I would like to dedicate this work to her. I also thank my friends and colleagues for their support and encouragement throughout the project. I particularly thank my supervisor, Sir Iqbal Hussain, for his guidance and support throughout the project. I would also like to thank my colleagues for their support and encouragement throughout the project.

Finally, I would like to thank Allah for his mercy and for providing me with the opportunity to complete this work. I would also like to thank my family and friends for their support and encouragement throughout the project.

Acknowledgements

In the name of Almighty Allah, the most merciful and the most beneficent, who enabled me to finish this work successfully. I pray to Him to show me the right path and grant me success both in the world and the world hereafter.

My first and foremost gratitude goes to my supervisor Dr. Jinbo Huang, whose technical help was vital in the completion of this work, as well as whose moral support kept me working hard even during the times of despair and hardship.

I thank John Slaney, Andrew Slater, Anbulagan, Sylvie Thiebaut, Jussi Rintanen, Alban Grastien, Arthur Choi, Mark Chavira and Brian Williams for providing much needed help and advice at various levels during the course.

I cannot forget the moral help and valuable advice given to me by my parents, brothers Rashid and Asaf, sisters and all my friends in and outside Australia. I feel proud to dedicate this work to my parents. At this moment, I particularly remember my father whose great vision of future led me throughout my academic career.

I deeply acknowledge the beautiful partnership of my wife Shaista who was always supportive even when I had to work long hours, and who always helped me keep my morale high during tough times.

Last but not least, I thank Asif Ali and Richard Hartley who provided help and motivation to earn a Ph.D. scholarship at the ANU.

Abstract

When a system behaves abnormally, a *diagnosis* is a set of system components whose failure explains the abnormality. Generally, the number of diagnoses can be exponential in the number of components and only one of them represents the actual faults.

In *Model-based diagnosis*, the functionality of the system is described by a knowledge base, called the *model* of the system. To find the failing components, *sequential diagnosis* takes a sequence of measurements of system variables and checks them against the system model until the actual faults can be inferred. It is desired to reduce the *diagnostic cost*, defined here as the number of measurements. Since computing an optimal plan of measurements is intractable in general, probabilistic heuristics are used to approximate it. For example, earlier approaches computed a set of *minimal diagnoses* of the system, as it can characterize the set of all diagnoses, and employed a heuristic based on reducing the entropy over the set of all diagnoses. This approach generally has good performance in terms of diagnostic cost, but can fail to diagnose large systems when the set of minimal diagnoses is too large.

Our goal is to scale the diagnosis to large systems by exploiting system structure, while achieving low-cost diagnosis. In principle, system models can be compiled into a tractable form, such as *decomposable negation normal form* (DNNF), which exploits system structure to compactly represent the functionality as well as the set of all diagnoses of the system. DNNF can be compact in size even when the number of diagnoses is exponentially large, while it supports efficient algorithms to perform common diagnostic reasoning. For large systems, however, compilation can become a bottleneck due to the large number of variables necessary to model the health of individual components. We further exploit system structure to scale the diagnosis.

We address the problem of diagnosing large systems using *structural abstraction*. The idea is to perform diagnosis on an abstraction of the system that contains fewer components. We propose a method of abstraction whose basic idea is to identify subsystems, called *cones*, that are dominated by single components, and model the health of each cone with a single health variable. When a cone is found to be possibly faulty, we diagnose it hierarchically by again identifying the cones inside it, and so on, until we reach a base case. This idea can significantly reduce the number of health variables

in the model allowing larger systems to be compiled and diagnosed. We show that our algorithm is sound and complete with respect to computing *minimum-cardinality diagnoses*.

We propose next a probabilistic heuristic to reduce the number of measurements required to find the actual faults. Our heuristic involves the posterior probabilities of component failures and the entropies of measurement points. Compared with the previous GDE framework, whose heuristic involves the entropy over the set of diagnoses and estimated posterior probabilities, the new method avoids the often impractical need to explicitly go through all diagnoses, and scales to much larger systems. All probabilities required for sequential diagnosis are computed exactly and efficiently once the system is modeled as a Bayesian network and compiled into a subset of DNNF known as *deterministic DNNF*. We show that our heuristic remains effective in hierarchical settings allowing it to be combined with abstraction, resulting in hierarchical sequential diagnosis which scales to larger benchmarks.

For the largest systems where even hierarchical diagnosis fails, we use a method that converts the system into one that has a smaller abstraction and whose diagnoses form a superset of those of the original system; the new system can then be diagnosed and the result mapped back to the original system. This approach allows measurement points to be computed and diagnosis performed on the largest benchmarks.

Finally, we apply the entropy-based approach to a branch-and-bound search algorithm for computing a common diagnosis query known as the *most probable explanation* (MPE). We study the impact of variable and value ordering on such a search algorithm. We study several heuristics based on the entropies of variables and on the notion of *nogoods*, and propose a new meta-heuristic that combines their strengths. The new heuristic significantly improves the search efficiency, allowing many hard problems to be solved for the first time.

Contents

Acknowledgements	vii
Abstract	ix
1 Introduction	1
1.1 Background	2
1.1.1 Model Based Diagnosis	2
1.1.2 Diagnosis by Compilation	3
1.1.2.1 Reiter's Approach	3
1.1.2.2 Diagnosis by Structure-based Compilation	4
1.1.3 Sequential Diagnosis	5
1.1.4 Most Probable Explanations	5
1.2 Contributions of This Thesis	6
1.2.1 Structural Abstraction and Hierarchical Diagnosis	6
1.2.2 Heuristic for Sequential Diagnosis	7
1.2.3 Reducing Abstraction Size with Cloning	7
1.2.4 Variable and Value Ordering for MPE Search	8
1.3 Thesis Structure	8
2 Hierarchical Diagnosis	9
2.1 Model-based Diagnosis	9
2.2 Diagnosis by Structure-based Compilation	13
2.2.1 Structure-based Compilation	14
2.2.2 Restricting the Compilation to the Given Observation	17
2.2.3 Projecting the Compilation over Health Variables	17
2.2.4 Smoothing	18
2.2.5 Minimization	18
2.2.6 Enumerating Diagnoses	19
2.3 Notation and Definitions	20
2.3.1 Circuits, Dominators, and Cones	20

2.3.2	Abstraction of Circuit	21
2.4	Hierarchical Diagnosis Algorithm	22
2.4.1	Step 1 (dominators)	22
2.4.2	Step 2 (cones and their inputs)	23
2.4.3	Step 3 (top-level diagnosis)	24
2.4.4	Step 4 (diagnosis of cones)	26
2.4.5	Avoiding Redundancy	28
2.4.6	Soundness and Completeness	30
2.5	Experimental Results	32
2.6	Conclusions	33
3	Adding Probabilities for Sequential Diagnosis	35
3.1	Probabilistic Framework	35
3.1.1	Joint Probability Distributions	35
3.1.2	Bayesian Network	37
3.1.3	Computing Posteriors by Compilation	38
3.1.3.1	Bayesian Network as Multi-Linear Functions	38
3.1.3.2	Converting the Bayesian Network into Arithmetic Circuit	40
3.1.3.3	Computing Probability of Evidence	42
3.1.3.4	Computing Posteriors	42
3.2	Previous Work	43
3.2.1	Sequential Diagnosis	43
3.2.2	GDE Framework	44
3.2.2.1	Minimizing Entropy	45
3.2.2.2	Computing Probabilities of Variables	45
3.2.2.3	Computing Expected Entropy	46
3.2.2.4	Computing Probabilities of Diagnoses	47
3.2.2.5	Drawbacks	47
3.3	The New Method	48
3.3.1	System Modeling and Compilation	49
3.3.1.1	Conditional Probability Tables	49
3.3.1.2	Propositional Modeling	50
3.3.2	Heuristic for Sequential Diagnosis	51
3.3.2.1	Heuristic Based on Entropy of Wire	52
3.3.2.2	Improving Heuristic Accuracy	52
3.3.3	The Algorithm	54

3.4	Experimental Results	55
3.4.1	Comparison with GDE	56
3.4.2	Larger Benchmarks	57
3.5	Conclusions	58
4	Hierarchical Sequential Diagnosis	59
4.1	Hierarchical Approach	59
4.1.1	Propositional Encoding	60
4.1.2	Prior Failure Probabilities for Cones	60
4.1.3	Measurement Point Selection and Stopping Criteria	61
4.1.4	The Algorithm	62
4.1.4.1	Example	64
4.2	Gate Cloning	65
4.2.1	Choices in Gate Cloning	66
4.2.2	Diagnosis with Gate Cloning	67
4.3	Experimental Results	68
4.3.1	Effectiveness of Abstraction	68
4.3.2	Effectiveness of Gate Cloning	69
4.4	Conclusions	71
5	Computing Most Probable Explanations	73
5.1	Introduction	73
5.2	Notation and Definitions	74
5.3	Computing MPE by Inference	76
5.3.1	Variable Elimination	76
5.3.2	Compilation	77
5.4	Computing MPE by Systematic Search	77
5.4.1	Computing Bounds using Mini-Buckets	78
5.4.2	Computing Bounds using Node Splitting	79
5.4.2.1	Branch-and-Bound Search	80
5.4.2.2	The Choice of Variables to Split	81
5.5	Variable and Value Ordering for MPE Search	81
5.5.1	Entropy-based Ordering	82
5.5.2	Nogood-based Ordering	83
5.5.2.1	Nogoods	83
5.5.2.2	Ordering Heuristic	84

5.6	Experimental Results	84
5.6.1	Grid Networks	85
5.6.2	Randomly Generated Networks	88
5.6.3	Networks for Genetic Linkage Analysis	88
5.6.4	Comparison with Other Tools	89
5.7	Conclusions	90
6	Related Work	91
6.1	Structural Methods	91
6.1.1	Diagnosis using Boolean Satisfiability	91
6.1.2	Hierarchical Methods	92
6.2	Probabilistic Methods	93
6.2.1	Optimal Policies using Heuristic Search and Entropy	93
6.2.2	Focusing on Probable Diagnoses	94
6.2.3	Improving Probability Estimates to Lower Diagnostic Costs	94
6.2.4	Decision-theoretic Troubleshooting	94
6.2.5	Adding Uncertainty to Model-based Diagnosis	95
6.2.6	Computing Most Probable Explanations	95
7	Conclusions and Future Work	97
	Bibliography	99

Introduction

When a system behaves abnormally, the purpose of *diagnosis* is to find which faulty components of the system are causing the abnormality. A diagnostic expert system may take a sequence of measurements of system variables to determine the faulty components, a process called *sequential diagnosis*. For this purpose, the expert has to maintain a knowledge base describing the behavior of the system so that measurements could be proposed and faults found by checking the measurement outcomes against it. Reasoning with a given knowledge base is not easy, in general; however it can be compiled into a tractable form, which can be used during the diagnosis process instead of the original knowledge base. As an example, the expert may compute a set of explanations (*diagnoses*) against a faulty behavior from the knowledge base, and keep narrowing it down by throwing away those explanations that contradict the measurement outcomes. When the expert is left with a single explanation, the actual diagnosis of the faulty behavior is found.

The challenges facing a computer-based expert include the following: First, the size of compiled forms of knowledge bases can often grow exponentially with system size, making the diagnosis of large systems difficult or even impossible. For example, a set of explanations may be too large to maintain by a computer even for a medium size system. Second, often there is trade-off between the size of the compilation and the efficiency of fault inference. Therefore, it is desired that the compilation be as compact as possible while still allowing a reasonably efficient fault inference. Third, it is desired to reduce the *diagnostic cost*, defined here as the number of measurements required before the faults are found. In this thesis we address the above mentioned challenges.

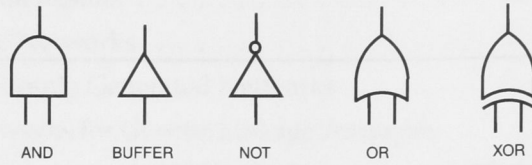


Figure 1.1: Some of the types of gates used in examples.

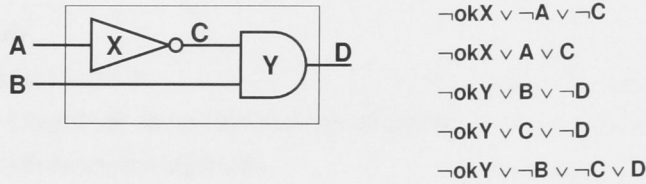


Figure 1.2: A circuit and its CNF encoding.

1.1 Background

In this section we establish the background necessary to understand our contributions. We mostly use combinational circuits for the purpose of giving examples. Some of the combinational gates used in our examples are given in Figure 1.1.

1.1.1 Model Based Diagnosis

In *model-based diagnosis*, the functionality of the system is described by a knowledge base, called the *model* of the system. Consider the example of a combinational circuit in Figure 1.2, reproduced from [Huang and Darwiche 2005], modeled as a propositional formula where each signal of the circuit translates into a propositional variable (A, B, C, D). For each gate an extra variable (okX, okY) is introduced to model its health. The propositional formula is such that when all health variables are true, the remaining variables are constrained to model the functionality of the gates. For instance, the first two clauses shown in the figure are equivalent to the sentence $okX \rightarrow (A \leftrightarrow \neg C)$, modeling the health of the inverter X .

Given a (typically abnormal) valuation of the inputs and outputs of the circuit, called an *observation*, a *consistency-based diagnosis* is then a valuation of the health variables that is consistent with the observation and system model. For instance, given the observation $\neg A \wedge B \wedge \neg D$, one diagnosis is $\neg okX \wedge okY$, meaning that the inverter X is broken and the and-gate Y is healthy, or in short the set $\{\neg okX\}$, where the health variables not appearing in the set are considered positive. In general the number of

diagnoses can be exponential in the number of system components.

Although propositional theories can be very compact, reasoning with them is generally computationally intractable. For example, it is generally not easy to compute the set of all diagnoses from a CNF (Conjunctive Normal Form) model of a system. In the following, we discuss the notion of *compilation* that can provide a solution to this problem.

1.1.2 Diagnosis by Compilation

To deal with the computational intractability of reasoning with general propositional theories, in principle, system models can be compiled into a tractable language, which supports efficient algorithms for a set of reasoning tasks according to the nature of that representation [Darwiche and Marquis 2002]. The idea is to split the reasoning into two phases: an offline phase which does the hard work of compilation, and an on-line phase which answers the queries regarding reasoning using the compilation instead of the original theory.

Here, we first discuss a historical approach followed by a recent structure-based approach to compilation.

1.1.2.1 Reiter's Approach

The set of all diagnoses normally contains many unlikely diagnoses, for example a diagnosis which declares all the components as faulty. Historically, a parsimonious approach has been used to filter out a preferred set of diagnoses. Particularly, Reiter is interested in only the minimal sets of failing components called *minimal diagnoses* [Reiter 1987]. A diagnosis is minimal if no proper subset of it is a valid diagnosis. The interesting property of minimal diagnoses is that, when the system is modeled as describing only the healthy behavior of components (as in our example), the set of minimal diagnoses characterizes the set of all diagnoses as every superset of a minimal diagnosis is also a diagnosis. When the system description contains the faulty behavior of components, called *fault models* [Struss and Dressler 1989], the set of minimal diagnoses fails to characterize all diagnoses, in which case a more general set of *kernel diagnoses* is computed based on a similar idea [de Kleer et al. 1992].

However, computing the set of minimal/kernel diagnoses is not easy. To compute minimal diagnoses, first a set of *minimal conflicts* of the system are computed, where a minimal conflict is a minimal set of components such that not all of them

can be functioning correctly under the given observation. Formally, a minimal conflict is a disjunction (clause) α of a minimal set of negated health variables, which if all be set to true, conflict with the observation and the model. Minimal conflicts are normally computed using an *Assumption Based Truth Maintenance System* (ATMS) [Forbus and de Kleer 1993]. A minimal diagnosis corresponds to the conjunction of a minimal set of negated health variables that, if conjoined with the minimal conflicts, satisfies (i.e. sets to true) every clause. The common method of computing the minimal diagnoses is to compute minimal *hitting sets* [Lin and Jiang 2003] of the minimal conflicts, where a hitting set is a set that intersects with every clause in the minimal conflicts. A hitting set is minimal if no proper subset of it is a hitting set.

Minimal conflicts are a special compilation language, which suffers from the disadvantage that can make it practically infeasible. Specifically, the number of minimal conflicts tends to be exponential in the number of system variables in the worst case. In our experience with the GDE system, for example (see Section 3.4.1), diagnosis using this approach often ran out of space except on very small systems.

1.1.2.2 Diagnosis by Structure-based Compilation

Fortunately, system models can be compiled into a tractable propositional language such as DNNF, which not only represents the functionality of a system compactly, thanks to exploitation of system structure, but also allows common diagnostic tasks to be performed efficiently [Darwiche 2001; Darwiche and Marquis 2002]. For example, once a system is compiled into DNNF, consistency-based diagnoses, as well as *minimum-cardinality diagnoses*, can be computed in time polynomial in the size of the DNNF [Darwiche 2001]. A minimal diagnosis is of minimum-cardinality if it contains the smallest number of faults.

DNNF can give significant advantage over minimal conflicts as it can often be very compact even though the number of minimal conflicts is too large. This is due to the fact that the size of the DNNF is worst case exponential only in the *treewidth* of the system structure, which is often much smaller than the number of system components. Also, contrary to minimal conflicts, structure can be exploited during compilation to make the process more efficient. Specifically, the compilation is driven by an efficient structural decomposition of the system model.

However, we note that despite the scalability of DNNF-based diagnosis approach, when it is applied to large systems, compilation of system models into DNNF can become a bottleneck due to large numbers of health variables.

1.1.3 Sequential Diagnosis

We now turn to the problem of finding the actual set of faults in a system. *Sequential diagnosis* takes a sequence of measurements of system variables, and checks them against the knowledge base of system's behavior until the actual faults can be inferred [de Kleer and Williams 1987]. In the above example, suppose that the knowledge base consists of a set of minimum-cardinality diagnoses of the system: $\{\neg okX\}$, $\{\neg okY\}$. Suppose that C is measured to be false, which means that X must be faulty, then the diagnosis $\{\neg okY\}$ can be safely removed. The goal in sequential diagnosis is to reduce the *diagnostic cost*. As in [de Kleer and Williams 1987], we assume that each measurement has a constant cost of 1 and define *diagnostic cost* as the number of measurements.

An optimal solution to sequential diagnosis would be a *policy*, that is, a plan of measurements conditioned on previous measurement outcomes, where each path in the plan leads to a diagnosis of the system [Heckerman et al. 1995]. Since computing an optimal policy is intractable in general, traditionally, heuristic methods are employed to propose good measurement points, which try to approximate the optimal policy. For this purpose, a probabilistic model of system can be used to guide the heuristic where probabilities are associated with system variables and/or diagnoses. For example, GDE (general diagnosis engine) computes the Shannon's entropy of the probability distribution over the set of all diagnoses, represented somewhat compactly by the set of minimal diagnoses [de Kleer and Williams 1987; de Kleer et al. 1992; de Kleer 2006]. Entropy is a concept from *information theory*, which reflects the uncertainty over the probability distribution [Pearl 1979]. GDE tries to minimize this uncertainty by proposing to measure a variable which causes the highest reduction in entropy of diagnoses on average. This approach generally has good performance in terms of diagnostic cost, but can fail to diagnose large systems when the set of minimal diagnoses is too large.

1.1.4 Most Probable Explanations

Finally, we discuss a common query in diagnostic reasoning using *Bayesian networks* [Pearl 1988] known as *most probable explanation* (MPE). A Bayesian network is used to represent joint probability distribution over the variables of a system compactly, and is a commonly used technique in many applications that use probabilistic reasoning.

In Bayesian networks, an MPE is a most likely instantiation of all network variables given a piece of evidence. Solving (the decision version of) an MPE query is NP-hard [Shimony 1994]. Exact algorithms for MPE based on *inference* include variable elimination, jointree, and, more recently, compilation [Chavira and Darwiche 2005]. While variable elimination and jointree algorithms have a complexity exponential in the treewidth of the network and are hence impractical for networks of large treewidth, compilation is known to exploit *local structure* so that treewidth is no longer necessarily a limiting factor [Chavira and Darwiche 2005].

When networks continue to grow in size and complexity, however, all these methods can fail, particularly by running out of memory, and one resorts instead to *search* algorithms. Most recently, Choi et al. [2007] proposed a branch-and-bound search framework for finding exact MPE solutions where bounds are computed by solving MPE queries on a relaxed network. The latter is obtained by *splitting* nodes of the network in such a way that (i) its treewidth decreases, making the MPE easier to solve, and (ii) the MPE probability of the relaxed network is no less than that of the original.

However, search time can be exponential in the number of split variables, in the worst case, and is sensitive to the order in which variables are chosen and assigned values.

1.2 Contributions of This Thesis

We now summarize the contributions of this thesis.

1.2.1 Structural Abstraction and Hierarchical Diagnosis

The solution we propose to the problem of diagnosing larger system is based on *hierarchical diagnosis*. We start with an abstraction of the system where certain regions of the circuit, called *cones*, are “carved out” based on a structural analysis. The abstract model, being generally much simpler, allows larger systems to be compiled and their diagnoses computed. The cones are diagnosed only when they are identified by the top-level diagnosis as possibly faulty. We discuss the intricacies involved in properly diagnosing the cones so that redundancy is avoided and results combined from the hierarchical diagnosis sessions are sound and complete with respect to minimum-cardinality diagnoses.

1.2.2 Heuristic for Sequential Diagnosis

For sequential diagnosis, we propose a new heuristic that does not require computing the entropy of diagnoses and scales to much larger systems. Instead we consider the entropies of the system variables to be measured as well as the posterior probabilities of component failures. The idea is to select a component that has the highest posterior probability of failure [Heckerman et al. 1995] and from the variables of that component, measure the one that has the highest entropy. To compute probabilities, we exploit system structure so that a joint probability distribution over the faults and system variables is represented compactly as a Bayesian network, which is then compiled into *deterministic* DNNF (d-DNNF) [Darwiche 2001; Darwiche and Marquis 2002]. All the required posterior probabilities can be exactly computed by evaluating and differentiating the d-DNNF in time linear in the d-DNNF size [Darwiche 2003]. Our heuristic remains effective in hierarchical settings allowing it to be combined with abstraction, resulting in hierarchical sequential diagnosis which scales to larger benchmarks.

1.2.3 Reducing Abstraction Size with Cloning

When the abstraction of a system is still too large to be compiled and diagnosed, we use a novel structure based technique called *cloning*, which systematically modifies the structure of a given system C to obtain a new system C' that has a smaller abstraction and whose diagnoses form a superset of those of the original system; the new system can then be diagnosed and the result mapped back to the original system. The idea is to select a system component G that is not part of a cone and hence cannot be abstracted away in hierarchical diagnosis, create one or more clones of G , and distribute G 's parents (from a graph point of view) among the clones, in such a way that G and its clones now become parts of cones and disappear from the abstraction. Repeated applications of this operation can allow an otherwise unmanageable system to have a small enough abstraction for diagnosis to succeed. This approach allows measurement points to be computed and diagnosis performed on the largest benchmarks. We note here that this technique is different from that of node splitting described in [Pipatsrisawat and Darwiche 2007; Choi et al. 2007], and will discuss the difference further in Section 4.2.

1.2.4 Variable and Value Ordering for MPE Search

Finally, we study the impact of variable and value ordering on the efficiency of branch-and-bound search for MPE. Specifically, we study heuristics based on the entropies of variables and on the notion of *nogoods*. The idea is to start the search with a high probability solution computed from the entropies of variables, and then use dynamically computed scores for variables to favor *nogood* assignments, which tend to cause early backtracking. Compared with the “neutral” heuristic used in [Choi et al. 2007], we show that our new heuristics further improve efficiency significantly, extending the reach of exact algorithms to networks that cannot be solved by other known methods.

1.3 Thesis Structure

The thesis is structured as follows: The relevant background and previous work are discussed in each individual chapter. Abstraction is described in detail in Chapter 2, where a new hierarchical algorithm for computing minimum-cardinality diagnoses is also presented. In Chapter 3, we discuss the probabilistic modeling of the systems required for sequential diagnosis and also present the new heuristic for computing measurement points. We apply the hierarchical approach and cloning to sequential diagnosis in Chapter 4. Computation of MPE is described in Chapter 5. Chapter 6 differentiates our techniques from related work. We draw conclusions and propose future work in Chapter 7.

From here on, we shall use combinational circuits as a concrete example of the type of systems we wish to diagnose, except for MPE where we consider Bayesian networks. The techniques, however, are not limited to circuits.

Hierarchical Diagnosis

This chapter is based upon work published in [Siddiqi and Huang 2007]. Here we formalize the notion of abstraction and present a new hierarchical algorithm for computing the set of minimum-cardinality diagnoses of an abnormal system. It is structured as follows: In Section 2.1 we review the history of relevant diagnosis approaches and formally define the problem of diagnosis, whereas a recent structure-based method that forms the basis of our algorithm is given in Section 2.2. Abstraction is formally defined in Section 2.3, followed by a detailed description of the new hierarchical algorithm in Section 2.4, where we also give analytical proof of the soundness and completeness of the new algorithm (Section 2.4.6). We give empirical results showing that the new algorithm significantly enhances the scalability and efficiency of the existing approach in Section 2.5, and then draw conclusions in Section 2.6.

2.1 Model-based Diagnosis

Reiter developed a general theory of diagnosis [Reiter 1987], which maintains a model (description) of the system to be diagnosed consisting of first order formulas that represent the behavior and the connections between various components of the system. A consistency-based diagnosis of the system is then defined as:

Definition 2.1.1 (Consistency-based Diagnosis)

A system to be diagnosed is defined by a set of components \mathbf{C} , a system description Δ (a propositional formula describing the system), and a set of observations β (a set of propositions). A **consistency-based diagnosis** for $(\Delta, \mathbf{C}, \beta)$ is defined to be a set $\mathbf{D} \subseteq \mathbf{C}$ such that $\Delta \cup \beta \cup \{ok_C | C \in \mathbf{C} \setminus \mathbf{D}\} \cup \{\neg ok_C | C \in \mathbf{D}\}$ is consistent, where ok_C is a binary variable indicating the health of a component C . A diagnosis may also be represented as the set $\{\neg ok_C | C \in \mathbf{D}\}$ of negated health variables indicating faulty components, where the components whose health variables do not appear in the set

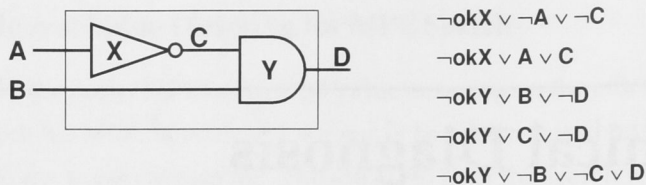


Figure 2.1: A circuit and its CNF encoding.

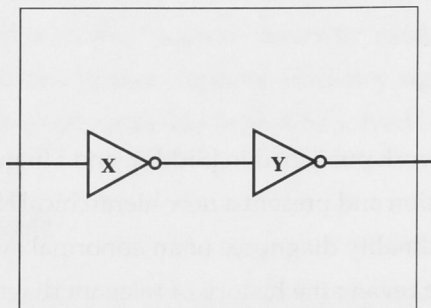


Figure 2.2: Cascaded inverters.

are assumed healthy.

For example, in Figure 2.1 the circuit description is a propositional theory describing the normal behavior of the system.

We denote the set of all health variables as **H**. Where convenient, we shall also use the variables **H** to refer to components themselves.

In general, the number of consistency-based diagnoses can be exponential in the number of system components. However, Reiter is interested in a characterization of the set of all diagnoses which includes only the minimal sets of failing components.

Definition 2.1.2 (Minimal Diagnosis)

A consistency-based diagnosis **D** of (Δ, C, β) is a **minimal diagnosis** iff no proper subset of **D** is a valid diagnosis.

For example, given the abnormal observation $\neg A \wedge B \wedge \neg D$ for the circuit in Figure 2.1, there are three possible diagnoses $\mathbf{d}_1 = \{\neg okX\}$, $\mathbf{d}_2 = \{\neg okY\}$, $\mathbf{d}_3 = \{\neg okX, \neg okY\}$, out of which \mathbf{d}_1 and \mathbf{d}_2 are minimal diagnoses.

When the system model Δ describes only the normal behavior of components, the minimal diagnoses characterize the set of all diagnoses of the system, as every superset of a minimal diagnosis is also a diagnosis. For example, in Figure 2.1, the system model only describes the correct behavior of the components and assumes an arbitrary behavior when they are faulty, and the diagnosis \mathbf{d}_3 is the su-

perset of both \mathbf{d}_1 and \mathbf{d}_2 . A *fault model* is a faulty behavior of a system component [Struss and Dressler 1989]. When fault models are introduced into the system behavior minimal diagnoses fail to characterize the set of all diagnoses as some superset of a minimal diagnosis may not be a diagnosis anymore. For example, the circuit in Figure 2.2 contains two cascaded inverters X and Y, which has the same three diagnoses \mathbf{d}_1 , \mathbf{d}_2 and \mathbf{d}_3 (given above) if no fault models are assumed. However, if we assume that an inverter outputs the same value as that of its input when it fails then \mathbf{d}_3 can no longer be a diagnosis, because if both of them are assumed faulty then the output of the circuit will not be abnormal. The minimal diagnoses in this case are \mathbf{d}_1 and \mathbf{d}_2 but do not characterize the set of all diagnoses, as their superset \mathbf{d}_3 is not a diagnosis.

To remove the above mentioned drawback with the minimal diagnoses the set of all diagnoses are characterized by a more general set of *kernel diagnoses* [de Kleer et al. 1992], which are based upon the idea of *partial diagnoses*. A partial diagnosis is formally represented as a conjunction of health literals and not as a set of failing components. Suppose that we are dealing with two diagnoses $\neg\text{ok}X \wedge \neg\text{ok}Y \wedge \neg\text{ok}Z$ and $\neg\text{ok}X \wedge \neg\text{ok}Y \wedge \text{ok}Z$, then we are certain that both X and Y are faulty, while we are not certain about the health of Z. Hence $\neg\text{ok}X \wedge \neg\text{ok}Y$ is a partial diagnosis. A partial diagnosis is minimal if no proper subset of it is a partial diagnosis. The minimal partial diagnoses are called the *kernel diagnoses*. Kernel diagnoses characterize the set of all diagnoses as follows: Let γ (a conjunction of health literals) be a kernel diagnosis, then γ has the property that if $\text{ok}C$ does not appear in γ then both $\gamma \wedge \text{ok}C$ and $\gamma \wedge \neg\text{ok}C$ are diagnoses, for any $\text{ok}C \in \mathbf{H}$. In the absence of fault models the kernel diagnoses precisely correspond to the minimal diagnoses. In our example, there are two kernel diagnoses: $\neg\text{ok}X \wedge \text{ok}Y$, $\text{ok}X \wedge \neg\text{ok}Y$ corresponding to \mathbf{d}_1 , \mathbf{d}_2 .

Reasoning with propositional theories is intractable in general, e.g. it is generally hard to compute the set of minimal/kernel diagnoses from a propositional model. In principle, system models can be compiled into a tractable language, which supports efficient algorithms for a set of reasoning tasks according to the nature of that representation. The idea is to split the reasoning into two phases: an offline phase that does the hard work of compilation, and an on-line phase that answers the queries regarding reasoning using that compilation instead of the original theory. As an example, the system model can be compiled into a language called *minimal conflicts* [de Kleer 1976] from where the minimal/kernel diagnoses can be computed, which is explained as

follows.

Definition 2.1.3 (Implicate)

A clause (disjunction of literals) α is an **implicate** of a propositional theory Δ iff $\Delta \models \alpha$.

An implicate is prime if no proper subset of it is an implicate of Δ .

Definition 2.1.4 (Implicant)

A term (conjunction of literals) β is an **implicant** of a propositional theory Δ iff $\beta \models \Delta$.

An implicant is prime if no proper subset of it is an implicant of Δ .

Definition 2.1.5 (Conflict)

An ok-literal is either okC or $\neg okC$ for some $C \in \mathbf{C}$. An ok-clause is a disjunction of ok-literals containing no complementary pair of ok-literals. A **conflict** of $(\Delta, \mathbf{C}, \beta)$ is an ok-clause entailed by $\Delta \cup \beta$. Thus a conflict is an implicate of $\Delta \cup \beta$. A negative conflict is a conflict all of whose literals are negative. A conflict is minimal if no proper subset of it is a conflict.

A conflict asserts that the set of health variables appearing in it cannot be assigned values such that the disjunction evaluates to false, as such an assignment would conflict with the model and the observation. Hence, a negative conflict asserts that not all components whose health variables appear in the conflict may be functioning correctly.

Minimal as well as kernel diagnoses can be defined from minimal conflicts as follows:

Definition 2.1.6 (Minimal Diagnoses)

The **minimal diagnoses** of $(\Delta, \mathbf{C}, \beta)$ are all the prime implicants of the set of negative conflicts of $(\Delta, \mathbf{C}, \beta)$.

Definition 2.1.7 (Kernel Diagnoses)

The **kernel diagnoses** of $(\Delta, \mathbf{C}, \beta)$ are all the prime implicants of the set of conflicts of $(\Delta, \mathbf{C}, \beta)$.

Note that in the case of minimal diagnoses only negative conflicts are considered whereas in the case of kernel diagnoses the condition of negativity is relaxed.

The common method of computing prime implicants of a theory Δ is to compute minimal *hitting sets* [Lin and Jiang 2003] of the minimal conflicts of Δ .

Definition 2.1.8 (Hitting Set)

A **hitting set** of a collection of sets \mathbf{S} is a set \mathbf{T} that intersects with every set in the collection \mathbf{S} . A hitting set is minimal if no proper subset of it is a hitting set.

Every such minimal hitting set corresponds to a prime implicant of the (minimal conflicts) of Δ .

The number of minimal conflicts of a theory tends to be exponential in the number of system variables in the worst case, which can make it practically infeasible to compute them. We will demonstrate with experiments, later in Section 3.4, that computing minimal diagnoses can be infeasible even for simple diagnosis problems. The main disadvantage of them is that the structure of the system cannot be exploited in computing them or in representing them compactly.

Before we go into the structure based approach towards diagnosis, we are also interested in another characterization of diagnoses called *minimum-cardinality diagnoses*.

Definition 2.1.9 (Minimum-Cardinality Diagnoses)

*The cardinality of a diagnosis is the number of failing components mentioned in it. A diagnosis γ of $(\Delta, \mathbf{C}, \beta)$ is a **minimum-cardinality diagnosis** if no other diagnosis of $(\Delta, \mathbf{C}, \beta)$ exists whose cardinality is less than that of γ .*

In our example, \mathbf{d}_1 and \mathbf{d}_2 are also minimum-cardinality diagnoses. In general, the number of minimum-cardinality diagnoses of a system can also be exponential in the number of system components.

2.2 Diagnosis by Structure-based Compilation

We now turn to the diagnosis approach that exploits system structure to often avoid the exponential blowup of space caused by computing a set of minimal or kernel diagnoses. Thanks to recent developments, system behavior can be compiled into a tractable form, such as *decomposable negation normal form* (DNNF), which exploits system structure to compactly represent the functionality of the system, and supports efficient computation of common diagnostic queries. DNNF is a graph-based representation for propositional theories. Specifically, each DNNF theory is a DAG (directed acyclic graph) with a single root where all leaves are labeled with literals and all other nodes are labeled with either AND or OR; in addition the *decomposability* property must be satisfied: children of any AND-node must not share variables.

Once a system is compiled into DNNF, consistency-based diagnoses, as well as minimum-cardinality diagnoses, can be computed in time polynomial in the size of the DNNF [Darwiche 2001]. The key is that decomposability allows nonobservables to be projected out in linear time, and allows diagnoses computed for children to be

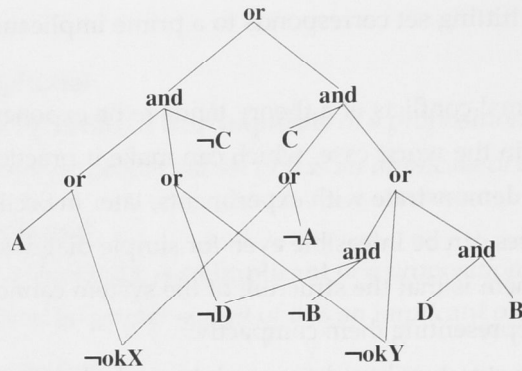


Figure 2.3: DNNF compilation of the circuit in Figure 2.1.

Algorithm 2.2.1 DNNF : Compiles a CNF to DNNF

function DNNF (t, α)

inputs: $\{t : \text{node in decomposition tree}\}, \{\alpha : \text{instantiation}\}$

local variables: $\{\gamma : \text{DNNF}\}, \{\beta : \text{instantiation}\}, \{t_l, t_r : \text{node in decomposition tree}\}$

1: **if** t is a leaf node and $\Delta(t) = \{\phi\}$ **then**

2: $\gamma \leftarrow \phi | \alpha$

3: **else**

4: $\gamma \leftarrow \bigvee_{\beta} \text{dnnf}(t_l, \alpha \wedge \beta) \wedge \text{dnnf}(t_r, \alpha \wedge \beta) \wedge \beta$, where β ranges over all instantiations of $\text{atoms}(t_l) \cap \text{atoms}(t_r) - \text{atoms}(\alpha)$

5: **end if**

return γ

combined at a parent AND-node simply by cross-concatenation (note that diagnoses computed for children can naturally be unioned at a parent OR-node).

In this section, we review this approach in more detail as presented in [Darwiche 2001; Huang and Darwiche 2005]. We explain the technique with the help of the example circuit in Figure 2.1, whose CNF encoding is described in Section 1.1.1 and whose DNNF compilation is shown in Figure 2.3. This example is reproduced from [Huang and Darwiche 2005].

2.2.1 Structure-based Compilation

Given a propositional model of a system, one challenge in computing the set of diagnoses is to project the model over health variables. Projection in propositional theories is generally not easy due to the fact that the variable being projected out may be shared between the conjuncts of a conjunction. However, it becomes easy if the decomposability property holds for every conjunct in the theory. Decomposability can be obtained by performing case analysis on the variables shared between the conjuncts, which is the basis of DNNF compilation. Here we explain this process and

show how system structure can be exploited to do it efficiently. The given algorithms and definitions are the same as in [Darwiche 2001].

The compilation process requires the propositional theory to be in *Conjunctive Normal Form* (CNF). A theory in CNF is a conjunction of clauses, which generally does not satisfy the decomposability property. Each individual clause in a CNF, however, is a DNNF. Let $\text{atoms}(\Delta)$ be the set of variables over which a propositional theory Δ is defined. If β is an instantiation of some variables in $\text{atoms}(\Delta)$, let $\Delta|\beta$ be a simplified version of Δ after it is restricted to β . The following theorem provides the basis for compilation.

Theorem 2.2.1

Let Δ_1 and Δ_2 be two DNNFs and $\mathbf{X} = \text{atoms}(\Delta_1) \cap \text{atoms}(\Delta_2)$. Let Δ be a sentence of the form $\bigvee_{\beta} (\Delta_1|\beta) \wedge (\Delta_2|\beta) \wedge \beta$, where β is an instantiation of variables \mathbf{X} . Then Δ is a DNNF and is equivalent to $\Delta_1 \wedge \Delta_2$.

Based upon the above theorem, the following algorithm converts a CNF Δ to a DNNF.

- (1) If Δ contains a single clause α , then $\text{DNNF}(\Delta) \leftarrow \alpha$.
- (2) Otherwise, $\text{DNNF}(\Delta) \leftarrow \bigvee_{\beta} \text{DNNF}(\Delta_1|\beta) \wedge \text{DNNF}(\Delta_2|\beta) \wedge \beta$, where Δ_1 and Δ_2 are two partitions of clauses in Δ and β is an instantiation of atoms shared between Δ_1 and Δ_2 .

The size of the resulting DNNF can, however, be large and is sensitive to how the partitioning of the CNF is performed. A good partitioning may lead to a smaller DNNF. This is the point where the structure of the problem is exploited to get a good partitioning. Specifically, a *decomposition tree* of the CNF is constructed and its *width* is used to measure the quality of the decomposition.

Definition 2.2.1 (Decomposition Tree)

A **decomposition tree** T for a CNF Δ is a binary tree whose leaves correspond to the clauses in Δ . If t is the leaf node in T corresponding to clause α in Δ , then $\Delta(t) = \{\alpha\}$. For every internal node t : t_l and t_r denote the left and right children of t , respectively, and $\Delta(t) = \Delta(t_l) \cup \Delta(t_r)$. $\text{atoms}(t)$ is defined to be the set of atoms (variables) appearing in clauses $\Delta(t)$. $\text{atoms}^{\uparrow}(t)$ is defined to be the set of atoms associated with leaf nodes that are not in the subtree rooted at t .

For example, Figure 2.4 shows a decomposition tree for the CNF theory in Figure 2.1, where $\text{atoms}(t_2) = \{\text{okX}, A, C\}$ and $\text{atoms}^{\uparrow}(t_2) = \{\text{okY}, B, C, D\}$.

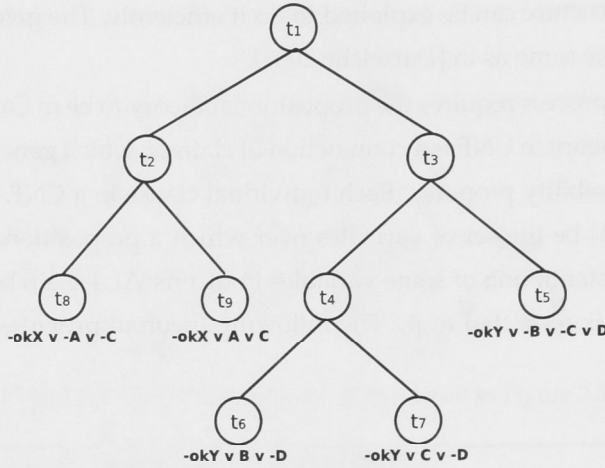


Figure 2.4: A decomposition tree for the CNF in Figure 2.1.

The decomposition tree provides the measure of the complexity of CNF to DNNF compilation as follows:

Definition 2.2.2 (Width of a Decomposition Tree)

Let t be a node in a decomposition tree T . The cluster of node t is defined as follows: if t is a leaf node, then its cluster is $atoms(t)$; if t is an internal node, then its cluster is $(atoms(t) \cap atoms^{\uparrow}(t)) \cup (atoms(t_1) \cap atoms(t_r))$. The **width of a decomposition tree** is the size of its maximal cluster minus one.

The complexity of CNF to DNNF compilation may be exponential only in the width of the decomposition tree. If a good decomposition tree can be obtained, then its width is often much smaller than the number of system components and can provide us with the advantage over computing the set of minimal or kernel diagnoses that may be exponential in the number of system components. Efficient decomposition trees can be constructed by two graph-based techniques, one using an *elimination order* of variables in the network graph [Darwiche 2001], and the other using the *hypergraph partitioning* of the network [Darwiche and Hopkins 2001].

Algorithm 2.2.1 is a basic algorithm to compile a CNF to DNNF by taking into account a decomposition tree of the CNF. This algorithm can be improved with computation reuse by caching the results of the DNNF at each node. Specifically, if two instantiations α and α' generated during case analysis agree on $atoms(t)$, then $dnnf(\Delta(t)|\alpha)$ is equivalent to $dnnf(\Delta(t)|\alpha')$. The compilation process has been made much more efficient thanks to the use of further advanced techniques described in [Darwiche 2004], which we omit.

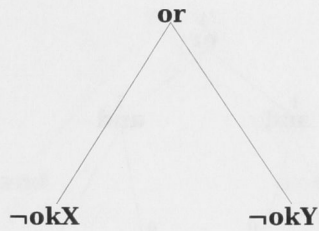


Figure 2.5: Simplified DNNF after restricting the DNNF in Figure 2.3 to $\neg A \wedge B \wedge \neg D$ and projecting out C .

Once a DNNF compilation of the system model has been obtained, the set of minimum-cardinality diagnoses of the system can be computed by performing a series of linear time operations on the DNNF, which first restrict the DNNF to the given observation, followed by projecting it over health variables, followed by minimization and then enumeration of diagnoses, explained as follows.

2.2.2 Restricting the Compilation to the Given Observation

First of all, the DNNF is pruned of all those assignments to variables that are inconsistent with the observation by the following linear time procedure, which *restricts* the DNNF according to the given observation.

(i) Assign Boolean constants (true = 1 or false = 0) to all the variables in the observation according to the observation. (ii) Evaluate the DNNF bottom-up as a Boolean function. (iii) Traverse the DNNF such that parents of a node are visited before the node, during which every true child of every and-node and every false child of every or-node are removed. (iv) Finally, for every node N if N has a single child, collapse N with the the only child of N .

Note that restricting the DNNF to the observation removes all those variables that appear in the observation.

2.2.3 Projecting the Compilation over Health Variables

The DNNF may still contain variables other than health variables whose values are not determined by the observation, which include the non-observables. Thanks to the decomposability property, projection can be done in linear time by replacing every literal of the variables being projected out with true and then simplifying the DNNF as described in steps (ii), (iii) and (iv) in Section 2.2.2. Figure 2.5 shows the simplified DNNF obtained by restricting to observation and projecting out the variable C in DNNF shown in Figure 2.3.

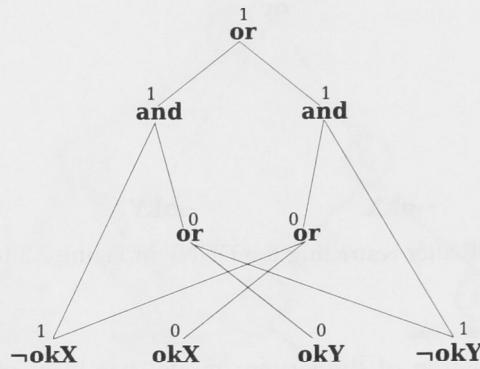


Figure 2.6: Smoothed version of the DNNF in Figure 2.5 and computing minimum cardinality

Once the DNNF has been projected over health variables, the set of all minimum-cardinality diagnoses can be computed by first performing a linear time minimization operation on it, which prunes away all the diagnoses that are not of minimum-cardinality, and then enumerating the remaining diagnoses. However, minimization requires the DNNF to be *smoothed* first.

2.2.4 Smoothing

A smooth DNNF (s-DNNF) is a subset of DNNF with the property that children of every or-node be defined over the same set of variables. For example, the DNNF in Figure 2.5 is not smooth, as the left child of the only or-node is defined over the variable $\{okX\}$ and is missing the variable okY . Similarly, the the right child is defined over the variable $\{okY\}$ and is missing the variable okX . Smoothing can be done by a linear time procedure that adds the terms of type $okZ \vee \neg okZ$ to every child of an or-node that is missing variable okZ . The DNNF in Figure 2.6 shows the smoothed version of DNNF in Figure 2.5.

2.2.5 Minimization

Once smoothed the DNNF can now be minimized. For this purpose, first the minimum-cardinality $mc(N)$ for each node N of the DNNF is computed by following linear time procedure: (i) Traverse the graph so that children of a node are visited before the node and: (a) for every leaf node N , set $mc(N) = 1$ if N represents a negative literal otherwise set $mc(N) = 0$, (b) for every or-node N , set $mc(N)$ to the smallest of the mc s of its children, (c) for every and-node N , set $mc(N)$ to the sum of the mc s of its children. (ii) Now traverse the DNNF so that parents of a node are visited before

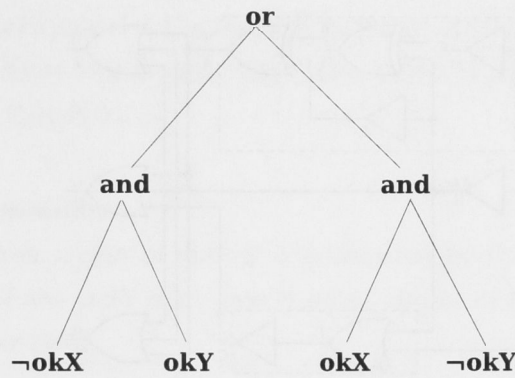


Figure 2.7: Representing minimum-cardinality diagnoses with minimized version of the DNNF in Figure 2.6

the node and remove every child V of every or-node N if $mc(V) > mc(N)$. (iii) Finally, for every node N if N has a single child, collapse N with the the only child of N .

The DNNF resulting from minimization only contains all the minimum-cardinality diagnoses. Figure 2.6 shows the computation of minimum-cardinality. The minimized and simplified DNNF is shown in Figure 2.7, which does not contain the non-minimum-cardinality diagnosis $\{\neg okX, \neg okY\}$.

2.2.6 Enumerating Diagnoses

In order to enumerate the set of all diagnoses, the DNNF has to be smooth. Minimization does not affect the property of smoothness of a DNNF; therefore once minimized, the set of all minimum-cardinality diagnoses can be enumerated from the DNNF by the following procedure.

Associate a set $diagnoses(N)$ with every node N , which represents the set of diagnoses of the DNNF under N . Then traverse the DNNF nodes such that children are visited before parents and: (i) For every leaf node N if it is a positive literal set $diagnoses(N) = \phi$, (ii) for every leaf node N if it is a negative literal set $diagnoses(N) = \{literal(N)\}$, (iii) for every and-node N $diagnoses(N)$ is the cross concatenation of the the diagnoses of its children, (iv) for every or-node N $diagnoses(N)$ is the union of the diagnoses of its children. The diagnoses of the root node is the set of diagnoses represented by the DNNF.

The complexity of enumerating diagnoses of a smooth DNNF is linear in its size and quadratic in the number of diagnoses, i.e. $O(mn)$, where m is the size of the DNNF and $n = |diagnoses(DNNF)|^2$ [Darwiche 1998].

This concludes the review of the structure-based diagnosis approach. Despite the

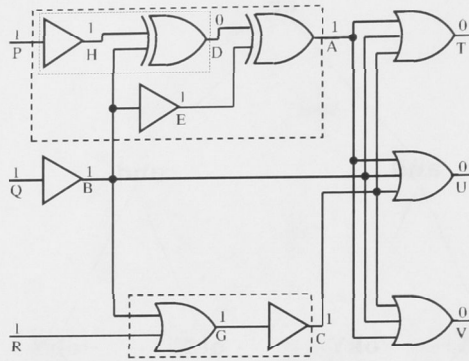


Figure 2.8: A circuit with cones.

scalability of this method, when it is applied to large systems, compilation of system models into DNNF can become a bottleneck due to large number of health variables. We address this problem using an abstraction based hierarchical approach described in the following sections.

2.3 Notation and Definitions

Before moving on to our new hierarchical algorithm we give some notation and definitions that will be used in the thesis including the formal definition of our novel technique of abstraction.

2.3.1 Circuits, Dominators, and Cones

We use C to denote the circuit as well as the set of gates of the circuit including the inputs (as trivial gates). We identify a gate with its output signal. The set of inputs of the circuit is denoted I_C and the set of outputs O_C . For example, $I_C = \{P, Q, R\}$ and $O_C = \{T, U, V\}$ for the circuit in Figure 2.8. We say that a gate Y is a PARENT of a gate G if G directly fans into Y , and thus G is called a CHILD of Y . We assume that the outputs of the circuit form the ‘top’ of the circuit while the inputs form its ‘bottom’.

One may observe that certain regions of this circuit have only limited connectivity with the rest of the circuit. For example, the dotted box containing gates $\{A, D, E, H, P\}$ is a sub-circuit that contributes a single signal (A) to the rest of the circuit. The box containing gates $\{D, H, P\}$ is another such example. We refer to such a sub-circuit as a *cone* (also known as *fan-out free formula* [Lu et al. 2003; Lu et al. 2003]), which we now formally define.

The *fan-in* region of a gate $G \in \mathbf{C}$ is the set of all those gates that have a path passing through G going to some output gate. The fan-in region of gate A in Figure 2.8, for example, is $\{A, B, D, E, H, P, Q\}$.

Definition 2.3.1 (Dominator)

A gate X in the fan-in region of gate G is dominated by G , and conversely G is a dominator of X , if any path from gate X to an output of the circuit contains G [Kirkland and Mercer 1987].

The notion of *cone* then corresponds precisely to the set of gates dominated by some gate G , which we denote by \mathbf{D}_G . For example, the dotted box mentioned above corresponds to $\mathbf{D}_A = \{A, D, E, H, P\}$. From here on, when the meaning is clear, we will simply use G to refer to the cone rooted at G .

2.3.2 Abstraction of Circuit

A circuit can be abstracted by treating all maximal cones in it as black boxes (a maximal cone is one that is either contained in no other cone or contained in exactly one other cone which is the whole circuit). For example, cone A can be treated as a virtual gate with two inputs $\{P, B\}$ and the output A . Similarly, cone A itself can be abstracted by treating cone D as a virtual gate. An abstraction of a circuit can hence be defined as the original circuit minus all non-root gates of maximal cones, or more formally:

Definition 2.3.2 (Abstraction of Circuit)

Given a circuit \mathbf{C} , let $\mathbf{C}' = \mathbf{C}$ if \mathbf{C} has a single output; otherwise let \mathbf{C}' be \mathbf{C} augmented with a dummy gate collecting all outputs of \mathbf{C} . The abstraction $\mathbf{A}_\mathbf{C}$ of circuit \mathbf{C} is then the set of gates $X \in \mathbf{C}$ such that there is a path from X to the output O of \mathbf{C}' that does not contain any dominator of X other than X and O .

For example, $\mathbf{A}_\mathbf{C} = \{T, U, V, A, B, C\}$. $E \notin \mathbf{A}_\mathbf{C}$ as E cannot reach any output without passing through A , which is a dominator of E . Similarly, $\mathbf{A}_\mathbf{A} = \{A, D, E\}$. $H \notin \mathbf{A}_\mathbf{A}$ as its only path to A contains D , which is a dominator of H .

Finally, in this chapter only, we assume weak fault models for gates. Establishing soundness and completeness of the techniques for stronger fault models is a subject of our future work.

2.4 Hierarchical Diagnosis Algorithm

We now describe the new hierarchical algorithm to compute all minimum cardinality diagnoses of an abnormal circuit. The key idea behind the algorithm is to start by obtaining the abstraction \mathbf{A}_C of a circuit C as defined in Section 2.3, and then diagnose C pretending that only gates in \mathbf{A}_C could be faulty. This is the basic technique that will significantly reduce the number of health variables required in the system model, allowing us to compile and diagnose larger circuits. Once this top-level diagnosis session finishes, if a gate appearing in a diagnosis is the root of a cone, which has been abstracted out, then we attempt to diagnose the cone, in a similar hierarchical fashion.

Two things are worth noting here before we go into details. First, cones are single-output circuits and hence the diagnosis of cones will always produce diagnoses of cardinality one. Second, the diagnosis of a cone is *not* performed simply with a recursive call as one may be tempted to expect. Indeed the later diagnosis sessions are very distinct from the initial top-level session. The reason has to do with avoiding redundant computation, which we will discuss later in the section.

Pseudocode of the new hierarchical diagnosis algorithm, which we will refer to as HDIAG, is given in Algorithm 2.4.3. Note that HDIAG is implemented on top of HD05 [Huang and Darwiche 2005] discussed in Section 2.2. Hence the basic method of DNNF compilation and enumeration of diagnoses remains the same; however, it is done in a hierarchical fashion by HDIAG.

2.4.1 Step 1 (dominators)

HDIAG starts by identifying the nontrivial dominator gates in the circuit (line 3 of Algorithm 2.4.3) (a trivial dominator is one that dominates only itself). First the dominators of every gate are obtained. The dominators of a gate are the gate itself union the intersection of the dominators of its parents [Kirkland and Mercer 1987], which can be found by a simple breadth-first traversal of the circuit starting from the outputs. During this process the nontrivial dominators can be identified.

Algorithm 2.4.1 gives a pseudo code of the procedure for identifying dominators, which traverses the circuit in breadth-first manner. Every gate is associated with an integer COUNTER, initialized to 0 (line 2), which keeps a count of how many parents of a gate have been processed at any time. A set of DOMINATORS is associated with every gate, which initially contains only the gate itself (line 3), noting that every gate dominates at least itself. During the breadth-first traversal (lines 5-17), when a gate

Algorithm 2.4.1 FINDDOMINATORS : Finds Dominator Gates in a Circuit**procedure** FINDDOMINATORS (C)**inputs:** { C : set of gates }**local variables:** { Q : set of gates }, { X, Y, Z, G : gate }

```

1: for all  $G \in C$  do
2:   COUNTER( $G$ )  $\leftarrow 0$ 
3:   DOMINATORS ( $G$ )  $\leftarrow \{G\}$ 
4: end for
5:  $Q \leftarrow \text{OUTPUTS}(C)$ 
6: while  $\neg \text{ISEMPTY}(Q)$  do
7:    $X \leftarrow \text{POP}(Q)$ 
8:   for all  $Y \in \text{PARENTS}(X)$  do
9:     DOMINATORS( $X$ )  $\leftarrow \text{DOMINATORS}(X) \cap \text{DOMINATORS}(Y)$ 
10:    for all  $Z \in \text{CHILDREN}(X)$  do
11:      COUNTER( $Z$ )  $\leftarrow \text{COUNTER}(Z) + 1$ 
12:      if COUNTER( $Z$ ) == NUMPARENTS ( $Z$ ) then
13:         $Q \leftarrow Q \cup \{Z\}$ 
14:      end if
15:    end for
16:  end for
17: end while

```

X is visited its dominators are computed (line 9), the counter for every child Z of X is incremented (line 11). When all the parents of a gate have been visited the gate itself is then queued for processing (lines 11-12).

Once the dominators of every gate have been computed the non-trivial dominators can be easily identified. Specifically, a gate is a non-trivial dominator if it is mentioned in the DOMINATORS set of more than one gate.

In our example, the dominator sets for T, U, V, A, B, C are $\{T\}, \{U\}, \{V\}, \{A\}, \{B\}, \{C\}$, respectively; the dominator set for D is $\{D, A\}$ and for H is $\{H, D, A\}$. It can be easily seen that the gates T, U , and V are trivial dominators whereas D and A are nontrivial dominators.

2.4.2 Step 2 (cones and their inputs)

Each nontrivial dominator defines a cone that can be abstracted out. Once the set of dominators of each gate has been computed, the set of gates D_G dominated by a dominator (or equivalently the set of gates contained in a cone) G can be easily computed. The set D_G for a dominator G is the set of all gates X such that $G \in \text{DOMINATORS}(X)$, where all such gates D_G always lie in the fan-in region of G .

Now we identify the inputs of these cones by a depth-first traversal of the circuit. The inputs I_G of a cone G can be found by traversing the fan-in region of G so that if we reach either an input of the circuit or a gate X that does not belong to D_G (or

Algorithm 2.4.2 FINDCONES : Finds Cones and their Inputs

```

procedure FINDCONES ( C )
inputs: { C : set of gates }
local variables: { Q, IC, IG: set of gates }, { X, G : gate }
1: IC ← INPUTS(C)
2: for all G ∈ C do
3:   if ISDOMINATOR(G) then
4:     IG ← φ
5:     Q ← {G}
6:     while ¬ISEMPTY(Q) do
7:       X ← POP(Q)
8:       if X ∈ IC || G ∉ DOMINATORS(X) then
9:         IG ← IG ∪ {X}
10:      else
11:        Q ← QUCHILDREN(X)
12:      end if
13:    end while
14:    SETINPUTS(G, IG)
15:  end if
16: end for

```

equivalently a gate X such that $G \notin \text{DOMINATORS}(X)$), we add it to I_G and backtrack. FINDCONES (line 3 of Algorithm 2.4.3) implements this procedure whose pseudo code is given in Algorithm 2.4.2.

For cone D in our example, we traverse the fan-in region of D in the order D, H, P, B . Gates P and B are added to the inputs of cone D . We backtrack from B as $B \notin D_D$. The inputs of cone D are thus $\{P, B\}$.

2.4.3 Step 3 (top-level diagnosis)

The rest of the algorithm proceeds in two phases. In the first phase we have the (abnormal) observation for the whole circuit. We first propagate the values of the inputs bottom-up, setting the (expected) value of each internal gate of the circuit. These values are saved for reference later. The observed outputs of the circuit are then set which may be abnormal. PROPAGATEINPUTS, SAVEVALUES, and SETOBSOUTPUTS on lines 4 and 5 of Algorithm 2.4.3 implement these procedures.

The health of the abstraction of the circuit, A_C , is then diagnosed. This is achieved by associating a health variable with every gate in A_C . A_C contains all the dominators of the top-level hierarchy plus all the non-dominators that sit between those dominators and the outputs O_C of the circuit. TRAVERSE A_C traverses the top-level hierarchy of the circuit. SINGLEOUTPUT on line 1 of Algorithm 2.4.4 implements the attachment of a dummy gate described in Definition 2.3.2. TRAVERSE A_C is implemented so that as soon as it encounters an in-

Algorithm 2.4.3 HDIAG : Hierarchical Diagnosis Algorithm

```

function HDIAG ( C, obs )
inputs: {C: circuit/set of gates}, {obs: set of <gate,bool>}
output: {set of sets of gates (minimum-cardinality diagnoses)}
local variables: {OC, IC, S, T : set of gates}, {V, D:set of sets of gates}, {Ω :<CNF, set of gates>}
1: OC ← OUTPUTS( C )
2: IC ← INPUTS( C )
3: FINDDOMINATORS( OC ), FINDCONES( OC )
4: PROPAGATEINPUTS ( C, IC, obs ), SAVEVALUES ( C )
5: SETOBSOUTPUTS ( OC, obs )
6: T ← TRAVERSE_AC ( OC, IC ), ATTACHOKS( T )
7: Ω ← GENMODEL( C, OC ∪ IC )
8: V ← CALLHD05( Ω, obs ), ORDERBYDEPTH( V )
9: RESTOREVALUES ( )
10: D ← φ
11: for all S ∈ V do
12:   D ← D ∪ FINDEQDIAGNOSES( S )
13: end for
14: return D

```

Algorithm 2.4.4 TRAVERSE_*A_C* : Traverses Top Level

```

function TRAVERSE_AC ( OC, IC )
inputs: {OC, IC : set of gates}
output: {set of gates}
local variables: {Q, K, T: set of gates}, {Y, O : gate}
1: O ← SINGLEOUTPUT( OC )
2: Q ← {O}, T ← φ
3: while ¬ISEMPTY( Q ) do
4:   Y ← POP( Q )
5:   if Y ∉ IC then
6:     T ← T ∪ {Y}
7:   end if
8:   if ¬ISDOMINATOR( Y ) || Y == O then
9:     K ← CHILDREN( Y ), Q ← Q ∪ K
10:  end if
11: end while
12: return T

```

put of the circuit or a nontrivial dominator (other than root, see line 7 of Algorithm 2.4.4), it backtracks (the inputs are excluded as they are assumed to be healthy). ATTACHOKS associates health variables with each gate in this hierarchy.

Now we create an abstract system model similar to the full model described in Section 2.2. The abstract model contains a system description, in CNF, over the set of observables ($I_C \cup O_C$) and nonobservables ($C \setminus (I_C \cup O_C)$). The model is generated by the function GENMODEL and returned as Ω . Note that only the clauses representing the gates marked by ATTACHOKS will have additional health variables. For example, consider the circuit and its CNF encoding in Figure 2.1, if the given circuit was a cone in a larger circuit, which dominates the gate X , then its clauses in the abstract model

of the circuit would not contain the health variable okX .

HD05 is then called with Ω and the observation to perform the diagnosis. The diagnoses thus obtained will be referred to as *top-level diagnoses*. If a nontrivial dominator $G \in \mathbf{A}_C$ is reported to be possibly faulty, we then enter the second phase by diagnosing the cone under G .

For example, in Figure 2.8, given the inputs, all the outputs of the circuit should be 1, but, all of them are 0 in our observation. We introduce health variables for the set of gates $\mathbf{A}_C \setminus \mathbf{I}_C = \{T, U, V, A, B, C\}$, create a model of the abstraction of the circuit, and pass the model along with the observation $\neg T \wedge \neg U \wedge \neg V \wedge P \wedge Q \wedge R$ to HD05. One of the diagnoses returned by HD05 is $\neg okA \wedge \neg okB \wedge \neg okC$ or, to use an alternative notation, $\{A, B, C\}$ (in the rest of the chapter we will use this latter notation expressing a diagnosis as a set of faulty gates). Since A is a cone, we enter the second phase.

2.4.4 Step 4 (diagnosis of cones)

Suppose that $\mathbf{B} = \{X, G_1, \dots, G_n\}$ is a diagnosis found in the top-level phase and that $X \in \mathbf{B}$ is (the root of) a cone. It should be clear that given the faulty output b of X , if there is a gate $Y \in \mathbf{D}_X$ that, when assumed faulty, permits the same value b at X , then replacing X with Y will yield a valid global diagnosis. This way we try to expand our top-level diagnoses in the procedure `FINDEQDIAGNOSES` (line 12 of Algorithm 2.4.3), given in Algorithm 2.4.5.

Mainly, for each cone X in each top-level diagnosis \mathbf{B} , we find a diagnosis for X hierarchically. We identify X as a sub-circuit with the single output X and the inputs \mathbf{I}_X . The minimum cardinality of diagnoses for X is always one as we mentioned earlier. We then replace X with its singleton diagnoses, in \mathbf{B} , one by one, to produce new global diagnoses. This process is iterated until we reach a base case (no cones left to be diagnosed). We take care that a cone is not diagnosed multiple times by caching on the observation on a cone and saving the corresponding diagnoses (lines 10 and 16), which is explained later in Section 2.4.5.

Cone X could be diagnosed as is done for the whole circuit but there is more to be done in order to find correct diagnoses. Specifically, we need a set of values for the inputs and output of cone X as an observation to pass to HD05. First of all we restore expected values of the gates in the circuit (line 9 of Algorithm 2.4.3) before we call `FINDEQDIAGNOSES`.

We want to diagnose cone X under the assumption that all of the gates $G_i \in \mathbf{B}$ are also faulty, so we need to propagate the fault effect of all G_i

Algorithm 2.4.5 FINDEQDIAGNOSES : Diagnosis of Cones**function** FINDEQDIAGNOSES (**B**)**inputs:** {**B**, set of gates}**output:** {set of sets of gates}**local variables:** {**E**, **V**: set of sets of gates}, {**P**, **R**, **C**, **I_X**, **Q**, **T**, **S**: set of gates}, {**X**, **Y**: gate}, {**Ω** :<CNF, set of gates>}, {**obs**: set of <gate,bool>}

```

1: E ← {B}
2: D ← {B}
3: while ¬ISEMPTY(E) do
4:   P ← POP(E)
5:   for all X ∈ P such that X is a cone and is not the whole circuit do
6:     IX ← INPUTS(X)
7:     PROPAGATEFAULT(P)
8:     obs ← VALUATION({X} ∪ IX)
9:     RESTOREVALUES()
10:    if ¬ALREADYDIAGNOSED(X, obs) then
11:      C ← TRAVERSECONE(X)
12:      T ← TRAVERSE_AC({X}, IX)
13:      ATTACHOKS(T)
14:      Ω ← GENMODEL(C, {X} ∪ IX)
15:      V ← CALLHDD05(Ω, obs)
16:      SAVEDIAGNOSES(X, obs, V)
17:    else
18:      V ← DIAGNOSES(X, obs)
19:    end if
20:    for all R ∈ V do
21:      Y ← POP(R)
22:      if X ≠ Y then
23:        S ← SUBSTITUTE(X, Y, P)
24:        E ← E ∪ {S}
25:        D ← D ∪ {S}
26:      end if
27:    end for
28:  end for
29: end while
30: return D

```

into the sub-circuit. Since faults in circuits propagate bottom-up, we put the set of gates in each diagnosis in decreasing order of their depth in the circuit (ORDERBYDEPTH on line 8 of Algorithm 2.4.3). A fault is processed simply by flipping the output of the faulty gate and propagating its effect. This is done for each $G \in \mathbf{B}$ separately in the order in which they appear in \mathbf{B} . This has the desired effect of setting an observation across cone X ($\mathbf{I}_X \cup \{X\}$) that is consistent with the observation of the overall circuit. Cone X is now ready for diagnosis.

Note that, again, only the abstraction \mathbf{A}_X of cone X is diagnosed. As we mentioned earlier, however, this is not simply a recursive call to Algorithm 2.4.3, but needs to be handled differently (Algorithm 2.4.5). TRAVERSECONE (line 11) identifies and returns the set of gates of the cone (i.e., \mathbf{D}_X).

VALUATION returns a set of $\langle \text{gate}, \text{bool} \rangle$ pairs which represents the currently assigned values to the set of gates $\{X\} \cup \mathbf{I}_X$ passed as input (line 8). The returned set serves as an observation across the cone. Given the cone, its observables and nonobservables, the health of its abstraction is diagnosed using HD05 and the set of singleton diagnoses \mathbf{V} found is saved (line 16). These diagnoses can be retrieved by DIAGNOSES (line 18). SUBSTITUTE (X, Y, \mathbf{P}) generates a new diagnosis by replacing X with Y in \mathbf{P} (line 23). Finally, we return from the function with the expanded set of diagnoses.

The diagnosis of cones is similar to that of the overall circuit given in Algorithm 2.4.3. There are, however, a few details worth mentioning: (i) Every cone is diagnosed hierarchically, i.e., for a cone X we diagnose only the top-level hierarchy \mathbf{A}_X of X . (ii) A cone may be subject to diagnosis multiple times; therefore results of their diagnosis are saved and reused later under certain circumstances (lines 10 and 16), which are discussed later. (iii) Once a new diagnosis is generated by substitution, it is added to the set \mathbf{E} for further processing (line 24 of Algorithm 2.4.5). (iv) The fault effect of a diagnosis is propagated before the diagnosis is processed, and undone afterwards (lines 7 and 9). (v) As substitutions are made, the gates in a newly generated diagnosis may not have the depth order discussed above; however, we do not re-order them for reasons that will be clear soon.

Continuing with our example, we reorder $\{A, B, C\}$ as $\{B, A, C\}$. We flip B to 0, propagate the effect that flips E to 0 and D to 1. A remains 1 at this stage; we then flip A to 0 (gate C is irrelevant to the diagnosis of cone A). Note that the correct output of cone A should be 1 given its inputs $P(1)$ and $B(0)$. We place health variables at the gates $\mathbf{A}_A \setminus \mathbf{I}_A = \{A, D, E\}$, generate a model for cone A , and pass it to HD05 with the observation $\neg A \wedge P \wedge \neg B$, which returns three diagnoses of cardinality one: $\{A\}, \{D\}, \{E\}$. Note that $\{A\}$ is a trivial diagnosis. Substituting D and E for A in the top-level diagnosis $\{B, A, C\}$, we get two new diagnoses: $\{B, D, C\}, \{B, E, C\}$.

2.4.5 Avoiding Redundancy

Since cones can be shared among different diagnoses, a cone is potentially visited more than once before all final diagnoses are computed. It may or may not be redundant to diagnose one cone multiple times. There are three cases to consider in this regard but all of them can be handled with a single technique, described below:

- (i) Suppose that in a top-level diagnosis, a cone S appears in two diagnoses $\{S, T\}$ and $\{S, U\}$, shown in Figure 2.9a. It can be seen that neither T nor U

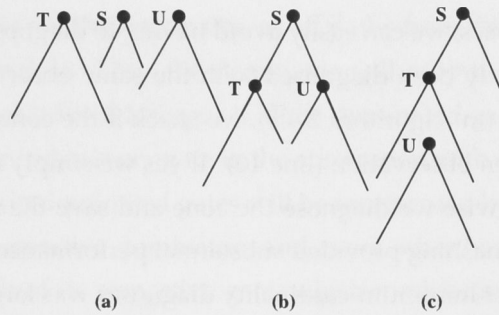


Figure 2.9: Redundancy scenarios.

lies in the fan-in region of cone S . Thus no fault at T or U can affect values at the inputs or output of cone S . This means that diagnosis of S can be obtained once and can be used in the processing of both of the top-level diagnoses.

- (ii) In Figure 2.9b, both T and U lie in the fan-in region of the cone S . Thus faults at T or U can affect values at the inputs and output of cone S . However, if we consider a fault at T and its effect across S (at the inputs and output of S), it may not be same as a fault at U and its effect across S . Thus the diagnosis of cone S during the processing of $\{S, T\}$ may be completely different from that during the processing of $\{S, U\}$. Hence cone S has to be diagnosed twice in this case.
- (iii) In Figure 2.9c, we assume that only $\{S, T\}$ is a top-level diagnosis whereas $\{S, U\}$ is a diagnosis obtained during the processing of $\{S, T\}$ such that $\{U\}$ is a cardinality one diagnosis for cone T . Hence the value seen at the output of T due to a fault at T would be the same as if seen due to a fault at U . Therefore, the effect of both faults, individually, across S would be the same and hence S needs to be diagnosed only once for both diagnoses. This idea extends to any level of substitution when diagnosing cone U further yields new diagnoses. For this reason we do not reorder the gates in the newly generated diagnoses (in `FINDEQDIAGNOSES`) according to their depth.

The third case can be easily implemented in the `FINDEQDIAGNOSES` procedure by making sure that a cone is diagnosed only once and results are cached during a single call to `FINDEQDIAGNOSES`, whereas the first and second cases are harder to implement. However, if we simply maintain a hash of observations on a cone and also save

the corresponding diagnoses, we can easily avoid having to diagnose a cone a second time if the cone has already been diagnosed with the same observation. Therefore, before diagnosing a cone (in Algorithm 2.4.5), we check if the cone has already been diagnosed under the given observation (line 10). If yes we simply retrieve the saved diagnoses (line 18); otherwise we diagnose the cone and save the results of diagnosis (line 16). This kind of hashing provided substantial performance improvement on cases where the number of minimum-cardinality diagnoses was large.

2.4.6 Soundness and Completeness

If some gates in a circuit are encoded without a health variable, while others are encoded with health variables (as in HDIAG), it is assumed that gates with no health variables cannot be faulty and will not appear in any diagnosis. If such an encoding of a circuit is given to HD05 for computing minimum cardinality diagnoses, there could be three outcomes:

- (i) The DNNF compilation produces a contradiction in the model, i.e., certain gates without health variables cannot be healthy under the given observation;
- (ii) The reported diagnoses have cardinality greater than that of the diagnoses of full encoding (when all gates are encoded with a health variable); or
- (iii) The reported set of diagnoses is either the same as the set of diagnoses of the full encoding or is a proper subset of it.

Therefore, it is necessary to establish that every diagnosis found by HDIAG is a minimum-cardinality diagnosis (soundness) and every minimum-cardinality diagnosis is found by HDIAG (completeness). We now give intuition why HDIAG should be sound and complete and discuss each of the above mentioned three possibilities one by one.

- (i) The abstraction of a circuit excludes all those gates that are (non-trivially) dominated by another gate, whereas it includes gates that are not dominated by any gate other than themselves. Hence, the abstraction includes the *highest dominators* of the circuit, where we say that a dominator is *highest* if it is not being dominated by any other gate. When a set of gates is dominated by a single dominator, the effect (if any) of a set of faults in

them does appear at the dominator, as all paths from them to circuit outputs pass through it. Thus the dominator will output a value which is different to what it would output if all its dominated gates were healthy. Since HDIAG encodes every gate in the abstraction with a health variable, the fault effects in dominated gates will be represented by the health variables of their respective dominators and every other health variable represents the fault of its respective gate, which models the health of every gate in the circuit. Hence a satisfiable assignment to health variables can be obtained under an (abnormal) observation and the hierarchical model never contradicts with it, which removes the first possibility.

- (ii) It is also easy to note that the reported diagnoses cannot be of cardinality greater than that of the diagnoses of full encoding. Since a fault at the root of a cone represents a set of faults inside the cone and the smallest cardinality of faults in a cone is 1; therefore, having health variables at the dominated gates could not have led to a smaller cardinality, which removes the second possibility.
- (iii) It leads us to the conclusion that the diagnoses reported by HD05 are of minimum cardinality and that they are subset of the minimum-cardinality diagnoses of the full encoding. If none of the diagnoses mention the possibility of a dominator to be faulty, then the reported set of diagnoses are complete. Otherwise, we note that a cone mentioned in a top-level diagnosis represents a set of possible single faults in it. Hence we diagnose such a cone after getting an observation at its inputs and output to find those single faults, and generate a set of global minimum-cardinality diagnoses that are guaranteed to be valid, thanks to the correctly computed observation for a cone. Doing the same for every cone in every top-level diagnosis and those generated from them gives us the complete set of minimum-cardinality diagnoses.

We now formally prove the following theorem, relying on the fact that the baseline diagnoser HD05 has the same property.

Theorem 2.4.1

HDIAG is sound and complete with respect to minimum-cardinality diagnoses.

Proof. Let \mathbf{C} be the circuit in question.

circuit	gates	cones	health vars for HDIAG	cases	cases solved				time on common cases	
					HDIAG	time	HD05	time	HDIAG	HD05
c432	160	64	59	700	700	0.4	700	7.8	0.4	7.8
c499	202	90	58	800	800	0.2	800	0.1	0.2	0.1
c880	383	177	77	800	799	0.4	786	0.5	0.4	0.5
c1355	546	162	58	800	800	0.4	792	1.5	0.4	1.5
c1908	880	374	160	400	388	685	94	4883	401	4883
c2670	1193	580	167	400	392	396	0	n.a	n.a	n.a

Table 2.1: Comparing HDIAG and HD05 on ISCAS-85 circuits.

Soundness: It should be clear that all diagnoses found by HDIAG are in fact valid. Moreover, they all have the same cardinality because (1) all top-level diagnoses found in Step 3 have the same cardinality by virtue of HD05 and (2) substitutions in Step 4 do not alter the cardinality as cones whose roots are in \mathbf{A}_C cannot overlap (i.e., two gates in a top-level diagnosis cannot be substituted by the same gate in Step 4).

Hence it remains to show that the cardinality of these diagnoses, call it d , is indeed the smallest possible. Suppose, on the contrary, that there exists a diagnosis \mathbf{S} of a smaller cardinality: $|\mathbf{S}| < d$. Now, let each gate in \mathbf{S} be replaced with its highest dominator in the circuit to produce \mathbf{S}' . Clearly, (i) \mathbf{S}' remains a valid diagnosis, (ii) $\mathbf{S}' \subseteq \mathbf{A}_C$, and (iii) $|\mathbf{S}'| \leq |\mathbf{S}|$. (i) and (ii) imply that \mathbf{S}' is a diagnosis for the abstraction \mathbf{A}_C . (iii) implies $|\mathbf{S}'| < d$. This means that the top-level diagnoses for \mathbf{A}_C found in Step 3 are not of minimum cardinality, contradicting the soundness of the baseline diagnoser HD05.

Completeness: Let \mathbf{S} be a diagnosis of minimum cardinality d . Let each gate in \mathbf{S} be replaced with its highest dominator in the circuit to produce \mathbf{S}' . Again, we have (i) and (ii) as above, and moreover $|\mathbf{S}'| = d$. This implies that \mathbf{S}' will be found by HDIAG in Step 3 by virtue of the completeness of HD05 in diagnosing \mathbf{A}_C . \mathbf{S} itself will then be found by substitutions in Step 4 by virtue of the completeness of the diagnosis of cones (which can be easily established by induction as all diagnoses for cones are of cardinality one). ■

2.5 Experimental Results

In order to compare the efficiency and scalability of our approach with that of [Huang and Darwiche 2005], we ran both systems on a set of ISCAS-85 circuits using randomly generated diagnostic cases. For each circuit, we randomly generated a set of input/output vectors accordingly to the correct behavior of the circuit. We then randomly flipped k outputs, with k ranging from 1 to 8, in each input/output vector

to get an (abnormal) observation (the minimum cardinality of the diagnoses was often close to the number of flipped outputs), except for c432 for which the range is from 1 to 7 as it only has only 7 outputs. Up to the circuit c1355, we generated 100 cases for each value of k , whereas for the remaining circuits only 50 cases were generated for each value of k . We observed that flipping more outputs to abnormal values tends to increase the number of diagnoses as well as the complexity of compilation.

All experiments were conducted, using ISCAS-85 benchmark circuits, on a cluster of 32 computers consisting of two types of (comparable) CPUs, Intel Core Duo 2.4 GHz and AMD Athlon 64 X2 Dual Core Processor 4600+, both with 4 GB of RAM running Linux. A time limit of 2 hour and a memory limit of 1.5 GB was imposed on each test case.

The results are given in Table 2.1. The fourth column shows the number of health variables used by HDIAG for the top-level diagnosis. Recall that the baseline approach HD05 requires a health variable for every gate. It is clear that the proposed technique significantly reduces the number of health variables required. We also observe that overall HDIAG was able to solve significantly more number of cases than HD05 while also solving those cases that HD05 can solve. In particular, HD05 could not solve any of the cases for c2670 and only solved 94 cases for c1908, whereas HDIAG failed on only a few cases for these two circuits. In the last two columns of the table, we compare the running times (in seconds) of the two systems on cases they both solved. The new approach clearly results in better efficiency.

Finally, we note that both programs reported exactly the same set of diagnoses for each case they both solved, as we expect given Theorem 2.4.1.

2.6 Conclusions

From the model compilation viewpoint, the first benefit we gain is that we significantly reduce the number of health variables needed in the model, allowing the diagnosis to scale to larger circuits. As a related benefit, some parts of the circuits may never be analyzed since a cone is only analyzed if it is part of a diagnosis computed in the higher level. As a third benefit we can now produce the complete set of minimum-cardinality diagnoses in a compact form, as a set of top-level diagnoses, each representing a class of diagnoses that can be obtained by substitutions. Overall, our approach results in improved efficiency and scalability as demonstrated against a recent diagnosis tool on ISCAS-85 circuits.

to get information about the system's behavior. The first step is to identify the symptoms and the conditions under which they occur. This is done by asking the user to describe the problem in detail. The next step is to identify the possible causes of the problem. This is done by asking the user to describe the conditions under which the problem occurs. The final step is to identify the most likely cause of the problem. This is done by asking the user to describe the conditions under which the problem occurs.

All experts were trained to use the hierarchical diagnosis system. The system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system.

The hierarchical diagnosis system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system. The hierarchical diagnosis system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system.

The hierarchical diagnosis system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system.

2.6. Conclusion

From the results reported in this paper, it can be seen that the hierarchical diagnosis system is a useful tool for diagnosing faults in a power plant. The hierarchical diagnosis system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system. The hierarchical diagnosis system was used to diagnose a number of faults in a power plant. The results of the diagnosis were compared with the results of a traditional diagnosis system. The results showed that the hierarchical diagnosis system was more accurate and faster than the traditional diagnosis system.

Adding Probabilities for Sequential Diagnosis

This chapter is based upon work published in [Siddiqi and Huang 2008]. Here, we add probabilities to our diagnostic framework and extend the compilation based diagnosis approach to sequential diagnosis, and also present a new scalable probabilistic heuristic for proposing measurements. The structure of this chapter is as follows: In Section 3.1 we discuss the probabilistic framework that underlies our algorithm. In Section 3.2 we formally define sequential diagnosis and review the previous GDE approach. In Section 3.3 we give details of the new techniques, and give empirical evidence for their effectiveness in Section 3.4. Finally, we draw conclusions in Section 3.5.

3.1 Probabilistic Framework

In this section we provide knowledge of the probabilistic framework that forms the basis of our sequential diagnosis algorithms, including *joint probability distributions*, Bayesian networks, and computation of probabilities by compilation.

3.1.1 Joint Probability Distributions

Consider a causality diagram between three events S (smoke), T (theft) and A (alarm) in Figure 3.1, which shows that alarm will be caused by either smoke, or theft, or both smoke and theft. We say that S and T are parents of A . Suppose that alarm is observed ($A = 1$), one may be interested in knowing the likelihood of smoke or theft in order to take appropriate actions.

A joint probability distribution is a way of assigning likelihood to events. Figure 3.2 shows a joint probability distribution between S , T and A . The probability

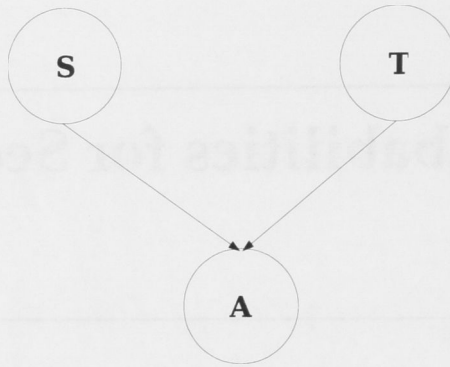


Figure 3.1: A causality diagram between three events S (smoke), T (theft) and A (alarm).

S	T	A	Pr(S, T, A)
1	1	1	0.24
1	1	0	0
1	0	1	0.36
1	0	0	0
0	1	1	0.16
0	1	0	0
0	0	1	0
0	0	0	0.24

Figure 3.2: A joint probability distribution between three events S (smoke), T (theft) and A (alarm).

of $A = 1, S = 1$, for example, can be obtained by summing up all the entries in the table which are consistent with $A = 1 \wedge S = 1$, i.e. entries 1 and 3. Hence $\Pr(S = 1, A = 1) = 0.60$. Similarly $\Pr(T = 1, A = 1) = 0.40$. The probabilities of individual events can be computed in the same way, e.g. $\Pr(S = 1)$ can be computed by summing up the first four entries of the table.

Definition 3.1.1 (Joint Probability Distribution)

A **joint probability distribution** is a function over a set of variables \mathbf{X} , mapping each instantiation \mathbf{x} of these variables to a number, denoted as $f(\mathbf{x})$, such that $0 \leq f(\mathbf{x}) \leq 1$ and $\sum_{\mathbf{x}} f(\mathbf{x}) = 1$.

Note that representing a joint probability distribution as a single table requires space exponential in the number of variables and hence will not be practical for large systems. A Bayesian network provides the solution to this problem.

S	θ_S	T	θ_T
1	0.6	1	0.4
0	0.4	0	0.6

S	T	A	$\theta_{A S,T}$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	1

Figure 3.3: CPTs for nodes S, T, and A. for the Bayesian network in Figure 3.1

3.1.2 Bayesian Network

A Bayesian network exploits the *Markov property* regarding the independence amongst the events to compactly represent the joint probability distribution [Pearl 1988]. Suppose that the node S has a parent X, and suppose that $S = 1$ is already known, then it is simple to note that the value of A cannot be affected by X, i.e. A becomes independent of X if S is known. Markov property captures this independence.

Definition 3.1.2 (Bayesian network)

A Bayesian network is a directed acyclic graph (DAG) together with a conditional probability table (CPT) for every node of the graph, where a node represents a variable and a CPT specifies the conditional probabilities of various instantiations of the corresponding variable given instantiations of the parent variables.

For example, Figure 3.1 shows a Bayesian network, and Figure 3.3 shows the CPTs of each of its nodes. The number $\theta_{x|u}$ in the rightmost column of a row is called a *network parameter* and represents the conditional probability of an instantiation of X given an instantiation of parent variables U. For example, the table at the bottom of Figure 3.3 gives the conditional probabilities ($\theta_{a|s,t}$) of instantiations of A given instantiations of S and T. The two tables at the top are special cases representing nodes having no parents (root nodes), which show numbers called *prior probabilities*.

Definition 3.1.3 (Markov Property)

Given an instantiation of its parents, the probability of a network node is independent of any node which is not in the fan-out of the node.

Markov property allows us to specify the probabilistic relationship of every variable with its parents only and the joint probabilities over all network variables can be obtained by realizing that the nodes of the network satisfy this property.

The joint probabilities can be computed from a Bayesian network by the *chain rule*, which says that the probability of an instantiation \mathbf{x} of all network variables \mathbf{X} is simply the product of all network parameters $\theta_{x|u}$, where xu is consistent with \mathbf{x} :

$$\Pr(\mathbf{x}) = \prod_{xu \sim \mathbf{x}} \theta_{x|u}. \quad (3.1)$$

For example, the probability $\Pr(S = 1, T = 1, A = 1)$ can be obtained by multiplying the first entry in the A 's CPT (1) with the first entry in S 's CPT (0.6) and the first entry in T 's CPT (0.4), i.e., $1 \times 0.6 \times 0.4 = 0.24$.

The joint probability distribution encoded in the Bayesian network provides the basis for computing any *posterior probabilities* $\Pr(x|\mathbf{y})$ (probability of an instantiation of variable X given an instantiation of variables \mathbf{Y}); however, it does not provide an efficient way of doing so. Computing a posterior probability $\Pr(x|\mathbf{y})$ involves the following equation:

$$\Pr(x|\mathbf{y}) = \frac{\Pr(x\mathbf{y})}{\Pr(\mathbf{y})}. \quad (3.2)$$

Computing $\Pr(x\mathbf{y})$ in turn involves summing out all variables other than X and \mathbf{Y} , which has a complexity exponential in the number of such variables if done naively; computing $\Pr(\mathbf{y})$ is similar.

3.1.3 Computing Posteriors by Compilation

Fortunately, [Darwiche 2002] shows that the CPTs of a Bayesian network can be encoded into a CNF formula, and after compiling the formula into *deterministic DNNF* (d-DNNF) all posterior probabilities can be computed by traversing the d-DNNF in linear time. Deterministic DNNF is a subset of DNNF with the property that children of every or-node are mutually logically inconsistent. In this section, we go through this technique as presented in [Darwiche 2002; Darwiche 2003].

3.1.3.1 Bayesian Network as Multi-Linear Functions

In order to answer probabilistic queries on a Bayesian network it is viewed as a multi-linear function having two types of variables (1) for every value x of a variable X in a

network, there is a variable λ_x , called *evidence indicator*, (2) for every instantiation xu of every variable X and its parents U in the network, there is a variable $\theta_{x|u}$, called *network parameter*. The multi-linear function has a term for each instantiation of network variables obtained by multiplying all evidence indicators and network parameters that are consistent with that instantiation. For example, for the 3 variable network in Figure 3.1 there would be 8 terms corresponding to 8 instantiations of S , T and A , as follows:

$$\begin{aligned}
 f = & \lambda_s \lambda_t \lambda_a \theta_s \theta_t \theta_{a|st} + \\
 & \lambda_s \lambda_t \lambda_{\bar{a}} \theta_s \theta_t \theta_{\bar{a}|st} + \\
 & \lambda_s \lambda_{\bar{t}} \lambda_a \theta_s \theta_{\bar{t}} \theta_{a|s\bar{t}} + \\
 & \lambda_s \lambda_{\bar{t}} \lambda_{\bar{a}} \theta_s \theta_{\bar{t}} \theta_{\bar{a}|s\bar{t}} + \\
 & \lambda_{\bar{s}} \lambda_t \lambda_a \theta_{\bar{s}} \theta_t \theta_{a|\bar{s}t} + \\
 & \lambda_{\bar{s}} \lambda_t \lambda_{\bar{a}} \theta_{\bar{s}} \theta_t \theta_{\bar{a}|\bar{s}t} + \\
 & \lambda_{\bar{s}} \lambda_{\bar{t}} \lambda_a \theta_{\bar{s}} \theta_{\bar{t}} \theta_{a|\bar{s}\bar{t}} + \\
 & \lambda_{\bar{s}} \lambda_{\bar{t}} \lambda_{\bar{a}} \theta_{\bar{s}} \theta_{\bar{t}} \theta_{\bar{a}|\bar{s}\bar{t}}
 \end{aligned} \tag{3.3}$$

We can compute any probability given the above multi-linear function. The probability $\Pr(\mathbf{e})$ of a piece of evidence \mathbf{e} can be computed by evaluating the multi-linear function after setting every indicator to 0 if it is inconsistent with \mathbf{e} , and to 1 otherwise. For example if $\mathbf{e} = (S = 1, A = 1)$, we set $\lambda_s = 1, \lambda_{\bar{s}} = 0, \lambda_t = 1, \lambda_{\bar{t}} = 1, \lambda_a = 1, \lambda_{\bar{a}} = 0$ in the above equation and evaluate it to get: $\Pr(\mathbf{e}) = \theta_s \theta_t \theta_{a|st} + \theta_s \theta_{\bar{t}} \theta_{a|s\bar{t}} = 0.6 \times 0.4 \times 1 + 0.6 \times 0.6 \times 1 = 0.24 + 0.36 = 0.60$.

According to differential semantics of inference in a Bayesian network, any posterior probability $\Pr(x|\mathbf{e})$ can be computed by first taking a partial derivative $\partial f / \partial \lambda_x$ and then evaluating the result under the evidence \mathbf{e} , i.e. by computing $\partial f / \partial \lambda_x(\mathbf{e})$, which gives us the probability $\Pr(x\mathbf{e})$. $\Pr(x\mathbf{e})$ is then divided by $\Pr(\mathbf{e})$ to get $\Pr(x|\mathbf{e})$ according to equation 3.2. The partial derivative $\partial f / \partial \lambda_x$ can be computed by setting λ_x to 1 and $\lambda_{\bar{x}}$ to 0 and simplifying the resulting equation. For example $\partial f / \partial \lambda_s$ is given as:

$$\begin{aligned}
& \neg\lambda_s \vee \neg\lambda_{\bar{s}}, \lambda_s \vee \lambda_{\bar{s}} \\
& \neg\lambda_t \vee \neg\lambda_{\bar{t}}, \lambda_t \vee \lambda_{\bar{t}} \\
& \neg\lambda_a \vee \neg\lambda_{\bar{a}}, \lambda_a \vee \lambda_{\bar{a}} \\
& \neg\lambda_s \vee \neg\theta_s, \lambda_s \vee \theta_s \\
& \neg\lambda_{\bar{s}} \vee \neg\theta_{\bar{s}}, \lambda_{\bar{s}} \vee \theta_{\bar{s}} \\
& \neg\lambda_t \vee \neg\theta_t, \lambda_t \vee \theta_t \\
& \neg\lambda_{\bar{t}} \vee \neg\theta_{\bar{t}}, \lambda_{\bar{t}} \vee \theta_{\bar{t}} \\
& \quad \neg\lambda_s \vee \neg\lambda_a \\
& \quad \neg\lambda_t \vee \neg\lambda_{\bar{a}} \\
& \quad \neg\lambda_{\bar{s}} \vee \neg\lambda_{\bar{t}} \vee \neg\lambda_a
\end{aligned}$$

Figure 3.4: CNF encoding of the Bayesian network in Figure 3.1, which exploits logical constraints and context-specific independence.

$$\begin{aligned}
\partial f / \partial \lambda_s &= \lambda_t \lambda_a \theta_s \theta_t \theta_{a|st} + \\
& \lambda_t \lambda_{\bar{a}} \theta_s \theta_t \theta_{\bar{a}|st} + \\
& \lambda_{\bar{t}} \lambda_a \theta_s \theta_{\bar{t}} \theta_{a|\bar{s}\bar{t}} + \\
& \lambda_{\bar{t}} \lambda_{\bar{a}} \theta_s \theta_{\bar{t}} \theta_{\bar{a}|\bar{s}\bar{t}}
\end{aligned} \tag{3.4}$$

3.1.3.2 Converting the Bayesian Network into Arithmetic Circuit

Similar to the joint probability distribution in Figure 3.2, the multi-linear function is exponential in the number of network variables. However, the function can be represented compactly as an *arithmetic circuit* (AC), which also allows partial differentiation to be computed, simultaneously, for all the variables, under a given evidence, in time linear in the size of the circuit. For this purpose, the CPTs of the Bayesian network are encoded as CNF and then compiled to d-DNNF, which can be easily transformed into an arithmetic circuit (described below). In the following, we allow ourselves the flexibility to interchangeably use the terms d-DNNF and arithmetic circuit.

We discuss one of the two types of CNF encodings of a CPT given in [Chavira and Darwiche 2008]. The encoding contains two types of clauses. First, for each network variable X with values x^1, x^2, \dots, x^k , the encoding Δ contains the

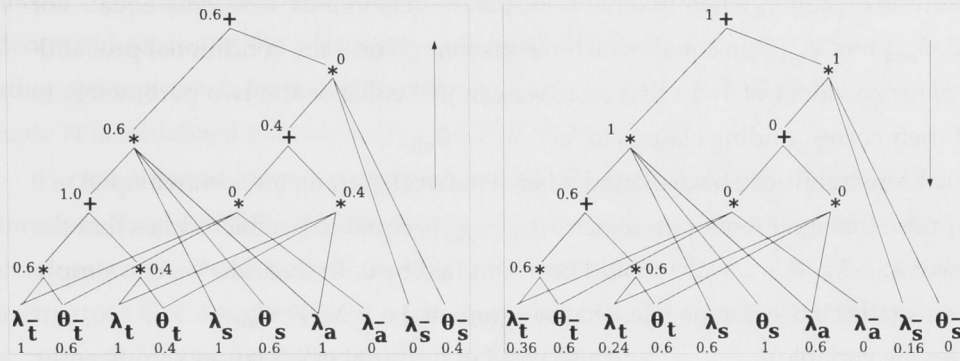


Figure 3.5: Arithmetic circuit obtained from the d-DNNF compilation of the CNF in Figure 3.4 and its evaluation (left) and differentiation (right) under observation $\mathbf{e} = \{A = 1, S = 1\}$.

clauses:

$$\lambda_{x_1} \vee \dots \vee \lambda_{x_k} \tag{3.5}$$

$$\neg \lambda_{x_i} \vee \neg \lambda_{x_j}, i \neq j \tag{3.6}$$

Second, for each network parameter $\theta_{x|u_1, \dots, u_m}$, the encoding Δ contains the clauses:

$$\lambda_x \wedge \lambda_{u_1} \wedge \dots \wedge \lambda_{u_m} \leftrightarrow \theta_{x|u_1, \dots, u_m} \tag{3.7}$$

The propositional theory Δ containing the above clauses for every CPT encodes the multi-linear function of the Bayesian network.

The CNF encoding of the Bayesian network can be compiled to d-DNNF. Thanks to structure exploitation, the d-DNNF is often very compact compared to the size of the multi-linear function, and may be exponential only in the treewidth of the network. A d-DNNF can be transformed into an arithmetic circuit by first smoothing it and then replacing each conjunction in d-DNNF by multiplication, and each disjunction by addition, and each negative literal by 1.

The encoding of the Bayesian network in Figure 3.1 contains 6 indicator variables, 12 network parameters and 38 clauses. However a much more efficient encoding results if some local structure in the form of *logical constraints* and *context-specific independence (CSI)* can be exploited, which can further reduce the complexity of compilation.

CSI can be exploited when two network parameters in a CPT become equal. For example, $\theta_{a|st}$ and $\theta_{a|s\bar{t}}$ are equal, which means that given s the conditional probability of a is independent of T . In that case, we can just collapse the two parameters to $\theta_{a|s}$ and their corresponding clauses to $\lambda_a \wedge \lambda_s \leftrightarrow \theta_{a|s}$.

Logical constraints can be exploited when a network parameter is either equal to 0 or 1. Suppose that a network parameter $\theta_{x|u_1, \dots, u_m}$ is equal to 0, which states that the assignment $\lambda_x \wedge \lambda_{u_1} \wedge \dots \wedge \lambda_{u_m}$ should be multiplied by 0. In that case we can simply generate a single clause that negates that assignment, i.e. $\neg \lambda_x \vee \neg \lambda_{u_1} \vee \dots \vee \neg \lambda_{u_m}$. It eliminates the variable $\theta_{x|u_1, \dots, u_m}$ and also reduces the number of clauses significantly. Similarly, if $\theta_{x|u_1, \dots, u_m}$ is equal to 1, which states that the assignment $\lambda_x \wedge \lambda_{u_1} \wedge \dots \wedge \lambda_{u_m}$ should be multiplied by 1, in which case we do not need to generate any clause.

For example, we exploit CSI on the CPT of node A in Figure 3.3: The second and fourth parameters are merged and their corresponding clauses collapse to $\lambda_s \wedge \lambda_{\bar{a}} \leftrightarrow 0$. Similarly, second and sixth parameters are merged and their corresponding clauses collapse to $\lambda_t \wedge \lambda_{\bar{a}} \leftrightarrow 0$. The seventh parameter can not be merged with any other and its clauses are $\lambda_s \wedge \lambda_{\bar{t}} \wedge \lambda_a \leftrightarrow 0$. We then use logical constraints to generate only three clauses for A given at the bottom of Figure 3.4.

As a consequence, the CNF encoding of the Bayesian network in Figure 3.1 contains 6 indicator variables and only 4 network parameters and 17 clauses, which is shown in Figure 3.4. The arithmetic circuit resulting from this encoding is shown in Figure 3.5.

3.1.3.3 Computing Probability of Evidence

After the compilation, the probability $\Pr(\mathbf{e})$ for an instantiation \mathbf{e} of any set of variables E can be obtained by the following linear-time procedure: (i) Set all evidence indicators E to Boolean constants (1 or 0) according to the instantiation \mathbf{e} , (ii) set all other evidence indicators to 1, (iii) set all network parameters to the corresponding numbers, (iv) evaluate the d-DNNF bottom-up.

The number at the root will be $\Pr(\mathbf{e})$. For example, Figure 3.5 (left) computes the probability of evidence $\mathbf{e} = \{A = 1, S = 1\}$, which is equal to 0.6.

3.1.3.4 Computing Posteriors

After the probability of evidence has been computed, all posteriors can be computed by a single downward traversal on the d-DNNF that performs partial differentiation. This is done by associating two registers with each node of the d-DNNE, namely ev

and dr . The registers ev are assigned values while computing the probability of evidence and represent the arithmetic evaluation of respective nodes. Before differentiating, the register dr for root is initialized to 1, i.e. $dr(\text{root}) = 1$, and dr for every other node N is initialized to 0, i.e. $dr(N) = 0, N \neq \text{root}$.

Partial differentiation is performed by the following linear time procedure, which traverses the d -DNNF such that the parents of a node are visited before node: (i) For every or-parent P of a node N , update $dr(N)$ as $dr(N) = dr(N) + dr(P)$, (ii) for every and-parent P of a node N let $d = dr(P)$, for every child V of P other than N update $d = d \times ev(V)$, finally update $dr(N) = dr(N) + d$.

After the differentiation the value in the dr register of a literal x (a leaf node) equals the probability $\Pr(xe)$. This value when divided by the value in the ev register of the root node (which equals $\Pr(\mathbf{e})$), gives the posterior probability $\Pr(x|\mathbf{e})$ according to Equation 3.2.

For example, Figure 3.5 (right) performs the partial differentiation on the arithmetic circuit. The numbers at the leaf nodes when divided by $\Pr(\mathbf{e}) = 0.6$ give the respective posterior probabilities. The posterior probability $\Pr(T = 0 | S = 1, A = 1)$ is thus $0.36/0.60 = 0.60$.

3.2 Previous Work

Having discussed the essentials of probabilities and Bayesian networks, we now formally define the problem of sequential diagnosis and review the previous GDE framework in more detail.

3.2.1 Sequential Diagnosis

Suppose that the system to be diagnosed is formally modeled by a joint probability distribution $\Pr(\mathbf{X} \cup \mathbf{H})$ over a set of variables partitioned into \mathbf{X} and \mathbf{H} . Variables \mathbf{X} are those whose values can be either observed or measured, and variables \mathbf{H} are the *health* variables, one for each component describing its health mode.

Diagnosis starts in the initial (belief) state

$$I_0 = \Pr(\mathbf{X} \cup \mathbf{H} \mid \mathbf{X}_0 = \mathbf{x}_0) \quad (3.8)$$

where values \mathbf{x}_0 of some variables $\mathbf{X}_0 \subseteq \mathbf{X}$ (we are using boldface uppercase letters to mean both sets and vectors) are given by the observation, and we wish to reach a goal

state

$$I_n = \Pr(\mathbf{X} \cup \mathbf{H} \mid \mathbf{X}_o = \mathbf{x}_o, \mathbf{X}_m = \mathbf{x}_m) \quad (3.9)$$

after measuring the values \mathbf{x}_m of some variables $\mathbf{X}_m \subseteq \mathbf{X} \setminus \mathbf{X}_o$, $|\mathbf{X}_m| = n$, one at a time, such that (the boldface $\mathbf{0}$ and $\mathbf{1}$ denote vectors of 0's and 1's):

$$\exists \mathbf{H}_f \subseteq \mathbf{H}, \Pr(\mathbf{H}_f = \mathbf{0} \mid \mathbf{X}_o = \mathbf{x}_o, \mathbf{X}_m = \mathbf{x}_m) = 1 \text{ and}$$

$$\Pr(\mathbf{H}_f = \mathbf{0}, \mathbf{H} \setminus \mathbf{H}_f = \mathbf{1} \mid \mathbf{X}_o = \mathbf{x}_o, \mathbf{X}_m = \mathbf{x}_m) > 0.$$

That is, in a goal state a set of components \mathbf{H}_f are known to be faulty with certainty and no logical inconsistency arises if all other components are assumed to be healthy. Other types of goal conditions are possible. For example, if the health states of all gates are to be determined with certainty, the condition will be that $\Pr(\mathbf{H} = \mathbf{0} \mid \mathbf{X}_o = \mathbf{x}_o, \mathbf{X}_m = \mathbf{x}_m)$ is 0 or 1 for all $H \in \mathbf{H}$ (such goals are only possible to reach if strong fault models are given).

Two special cases are worth mentioning:

- (i) If the initial state I_0 satisfies the goal condition with $\mathbf{H}_f = \emptyset$ then the observation is normal and no diagnosis is required.
- (ii) If the initial state I_0 satisfies the goal condition with some $\mathbf{H}_f \neq \emptyset$, then the observation is abnormal but the diagnosis is already completed (assuming that we are able to check probabilities as necessary); in other words, a sequence of length 0 solves the problem.

As in [de Kleer and Williams 1987], we assume that all measurements have unit cost. Hence the objective is to reach a goal state in the fewest measurements possible.

3.2.2 GDE Framework

We start with a definition of Shannon's entropy ξ , which is defined with respect to a probability distribution of a discrete random variable X ranging over values x_1, x_2, \dots, x_k . Formally:

$$\xi(X) = - \sum_{i=1}^k \Pr(X = x_i) \log \Pr(X = x_i) \quad (3.10)$$

Entropy measures the amount of uncertainty over the value of the random variable. It is maximal when all probabilities $\Pr(X = x_i)$ are equal, and minimal when one

of the probabilities is 1, corresponding nicely to our intuitive notion of the degree of uncertainty.

3.2.2.1 Minimizing Entropy

The classical GDE framework [de Kleer and Williams 1987], on receiving an abnormal observation $\mathbf{X}_0 = \mathbf{x}_0$, considers the Shannon's entropy of the probability distribution over the set of all diagnoses. The idea is that the probability distribution over the diagnoses reflects the uncertainty over the actual faults, and the entropy captures the amount of this uncertainty. Therefore, GDE uses a one-step lookahead strategy based on entropy and proposes to measure a variable X that will give the least value of the entropy of diagnoses resulting from measuring X , on average. The average expected entropy $\xi_e(X)$ over the diagnoses that result from measuring X can be computed from the following equation:

$$\xi_e(X) = \sum_{i=1}^k \Pr(X = x_i) \xi(X = x_i) \quad (3.11)$$

where $\xi(X = x_i)$ is the entropy of diagnoses resulting from taking the hypothesized measurement $X = x_i$.

Hence the best next variable to measure is the one that has the least value of $\xi_e(X)$.

3.2.2.2 Computing Probabilities of Variables

GDE does not have an efficient and exact method of computing the probabilities $\Pr(X = x_i)$ of values of variables; hence it resorts to estimating them from the current set of diagnoses as follows: If every diagnosis predicts that $X \neq x_i$, then $\Pr(X = x_i) = 0$; otherwise, the set of diagnoses are divided into two sets namely \mathbf{S}_{x_i} and \mathbf{U}_X . The set \mathbf{S}_{x_i} are those diagnoses that predict the value of X to be x_i , whereas \mathbf{U}_X are those that do not predict the value of X . Every diagnosis in $\mathbf{S}_{x_i} \cup \mathbf{U}_X$ contributes to the probability $\Pr(X = x_i)$ such that $\Pr(X = x_i)$ is the sum of the contributions from each diagnosis. The contribution from a diagnosis in the set \mathbf{S}_{x_i} is the probability of that diagnosis, whereas the contribution from a diagnosis in the set \mathbf{U}_X is only approximated, with an error, ϵ_{x_i} , such that every value of X is assumed to be equally likely. Formally:

$$\Pr(X = x_i) = \sum_{\mathbf{d} \in \mathbf{S}_{x_i}} \Pr(\mathbf{d}) + \epsilon_{x_i} \quad (3.12)$$

where

$$\epsilon_{x_i} = \frac{1}{m} \sum_{\mathbf{d} \in \mathbf{U}_X} \Pr(\mathbf{d}) \quad (3.13)$$

and m is the domain size of X .

Note that if \mathbf{S}_{x_i} is empty then the values of X essentially become equally likely.

3.2.2.3 Computing Expected Entropy

In practice, the computation of average expected entropy $\xi_e(X)$ of diagnoses resulting from hypothesized measurements of X can be done from quantities predicted from the current set of diagnoses and does not require computing the new set of diagnoses resulting from measuring X . Suppose that first n of the m possible values of X are predicted, then:

$$\xi_e(X) = \sum_{i=1}^n (\Pr(X = x_i) + \epsilon_{x_i}) \xi(X = x_i) + \sum_{i=n+1}^m \epsilon_{x_i} \xi_{\mathbf{U}_X} \quad (3.14)$$

where $\xi_{\mathbf{U}_X}$ is the expected entropy if X is measured to have an unpredicted value (i.e. all but the diagnoses \mathbf{U}_X are eliminated):

$$\xi_{\mathbf{U}_X} = - \sum_{\mathbf{d} \in \mathbf{U}_X} \Pr(\mathbf{d}) \log(\Pr(\mathbf{d})) \quad (3.15)$$

Since the term $\epsilon_{x_i} \xi_{\mathbf{U}_X}$ is independent of the value measured, thus rewriting Equation 3.14 we obtain:

$$\xi_e(X) = \sum_{i=1}^n (\Pr(X = x_i) + \epsilon_{x_i}) \xi(X = x_i) + (m - n) \epsilon_{x_i} \xi_{\mathbf{U}_X} \quad (3.16)$$

Substituting and simplifying gives:

$$\xi_e(X) = \xi + \Delta \xi_e(X) \quad (3.17)$$

where ξ is the current entropy and $\Delta \xi_e(X)$ is:

$$\Delta \xi_e(X) = \sum_{i=1}^n \Pr(X = x_i) \log(\Pr(X = x_i)) + \Pr(\mathbf{U}_X) \log(\Pr(\mathbf{U}_X)) - \frac{n \Pr(\mathbf{U}_X)}{m} \log\left(\frac{n \Pr(\mathbf{U}_X)}{m}\right) \quad (3.18)$$

where

$$\Pr(\mathbf{U}_X) = \sum_{\mathbf{d} \in \mathbf{U}_X} \Pr(\mathbf{d})$$

Hence the best next measurement is the one that minimizes $\Delta \xi_c(X)$, which does not require computing the diagnoses resulting from measuring X .

3.2.2.4 Computing Probabilities of Diagnoses

The initial probabilities of diagnoses are computed from prior failure probabilities of components. Let \mathbf{d} be a diagnosis, then the initial probability $\Pr(\mathbf{d})$ is given as:

$$\Pr(\mathbf{d}) = \prod_{\text{okC} \in \mathbf{D}} \Pr(\neg \text{okC}) \prod_{\text{okC} \in \mathbf{H} \setminus \mathbf{D}} 1 - \Pr(\neg \text{okC}) \quad (3.19)$$

After a measurement $X = x_i$ is taken the entropy is updated by updating the posterior probabilities of the diagnoses, potentially reducing some of them to 0 at which point they are discarded. The posterior probabilities $\Pr(\mathbf{d}|X = x_i)$ after the measurement $X = x_i$ are computed using Bayes' rule as:

$$\Pr(\mathbf{d}|X = x_i) = \frac{\Pr(X = x_i|\mathbf{d})\Pr(\mathbf{d})}{\Pr(X = x_i)} \quad (3.20)$$

The computation of probability $\Pr(X = x_i|\mathbf{d})$ again involves estimation, as follows: If \mathbf{d} predicts that $X \neq x_i$ then $\Pr(X = x_i|\mathbf{d}) = 0$; If \mathbf{d} predicts that $X = x_i$ then $\Pr(X = x_i|\mathbf{d}) = 1$; Otherwise, if \mathbf{d} does not predict any value of X then the conditional probabilities $\Pr(X = x_i|\mathbf{d})$ of values of X are considered to be equally likely, i.e. $\Pr(X = x_i|\mathbf{d}) = 1/m$.

3.2.2.5 Drawbacks

The results in [de Kleer et al. 1992] involving single-fault cases for ISCAS-85 circuits indicate that this method leads to measurement costs close to those of optimal policies. However, a major drawback of GDE is that it can be impractical when the number of diagnoses is large [de Kleer and Williams 1987; de Kleer 2006], as it requires going through the set of diagnoses for the computation of entropy. In practice, though, one does not need to perform the computationally prohibitive task of computing the set of all diagnoses, and can only work with a much smaller set of preferable diagnoses, e.g., the *minimal diagnoses*, as the available public version of GDE [Forbus and de Kleer 1993] does, or the even smaller set of *most probable diag-*

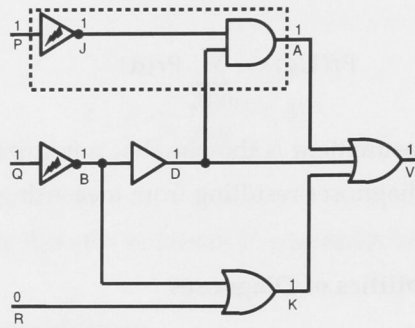


Figure 3.6: A faulty circuit.

nodes [de Kleer 1992]. However, the set of minimal diagnoses can also be exponentially large. In addition, working with a set of most probable diagnoses can compromise the quality of the heuristic in terms of diagnostic cost [de Kleer 1992], as the computed probabilities may be inaccurate. In the next section we will present the new techniques we have introduced to address the drawbacks of GDE and significantly enhance the scalability and efficiency of sequential diagnosis.

3.3 The New Method

We model the circuit as a propositional formula in CNF, and compile it, together with an abnormal observation, into d-DNNF, where we associate real numbers with the propositional variables, which represent the prior probabilities of gate failures given as part of the input to the diagnosis task. Because of the independence condition that exists among the gates, our d-DNNF compilation amounts to a compilation of a Bayesian network that compactly encodes the joint probability distribution over all the wires and faults of the circuit.

To select a measurement point, we use heuristics that do not require the enumeration of all diagnoses. Instead we only require the entropies of the wires together with the posterior probabilities of component failures. All probabilities required can be exactly computed simultaneously in a single traversal of the d-DNNF which performs partial differentiation [Darwiche 2003].

We start by presenting in the following section the system modeling and compilation method that underlies our new diagnostic system.

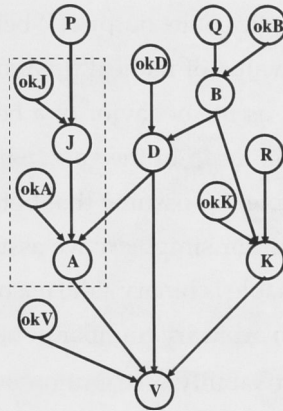


Figure 3.7: Bayesian network for the circuit in Figure 3.6.

P	θ_P	okJ	θ_{okJ}
1	0.5	1	0.9
0	0.5	0	0.1

P	okJ	J	$\theta_{J P,okJ}$
1	1	1	0
1	1	0	1
1	0	1	0.5
1	0	0	0.5
0	1	1	1
0	1	0	0
0	0	1	0.5
0	0	0	0.5

Figure 3.8: CPTs for nodes P, J, and okJ. for the Bayesian network in Figure 3.7

3.3.1 System Modeling and Compilation

In order to define a joint probability distribution $\Pr(\mathbf{X} \cup \mathbf{H})$ over the circuit behavior, we first assume that the prior probability of failure $\Pr(H = 0)$ is given for each gate $H \in \mathbf{H}$, as part of the input to the diagnosis task [de Kleer and Williams 1987]. For example, the small table with two entries on the top-right of Figure 3.8 gives the prior probability of failure for gate J as 0.1.

3.3.1.1 Conditional Probability Tables

Prior fault probabilities alone do not define the joint probability distribution $\Pr(\mathbf{X} \cup \mathbf{H})$. In addition, we need to specify for each gate how its output is related to its inputs and health mode. A conditional probability table (CPT) for each gate does this job.

The CPT shown on the bottom of Figure 3.8, for example, defines the behavior of

gate J : Each entry gives the probability of its output (J) being a particular value given the value of its input (P) and the value of its health variable (ok_J). In case $ok_J = 1$, the probabilities are always 0 or 1 as the behavior of a healthy gate is deterministic. The case of $ok_J = 0$ defines the *fault model* of the gate, which is also part of the input to the diagnosis task. In our example, we assume that both output values have probability 0.5 when the gate is broken. For simplicity we assume that all gates have two health modes (i.e., each health variable is binary); the encoding and compilation to be described later, however, allows an arbitrary number of health modes.

Given these tables, the joint probability distribution over the circuit behavior can be obtained by realizing that the gates of a circuit satisfy the Markov property: Given its inputs and health mode, the output of a gate is independent of any wire which is not in the fan-out of the gate. This means that the circuit can be effectively treated as a Bayesian network in the straightforward way, by having a node for each wire and each health variable, and having an edge going from each input of a gate to its output, and also from the health variable of a gate to its output. Figure 3.7 shows the result of this translation for the circuit in Figure 3.6.

The joint probability distribution encoded in the Bayesian network provides the basis for computing any posterior probabilities that we may need when proposing measurement points (by the *chain rule*). However, as said earlier, it does not provide an efficient way of doing so. Specifically, computing a posterior $\Pr(X = x \mid \mathbf{Y} = \mathbf{y})$ given the values \mathbf{y} of all the wires \mathbf{Y} with known values involves summing out all variables other than X and \mathbf{Y} , which has a complexity exponential in the number of such variables if done naively.

3.3.1.2 Propositional Modeling

As we have discussed a Bayesian network can be encoded into a logical formula and compiled into d-DNNE, which, if successful, allows posterior probabilities of all variables to be computed efficiently [Darwiche 2003]. For the purposes of sequential diagnosis, we encode the Bayesian network as follows.

Consider the subcircuit in the dotted box in Figure 3.6 as an example, which can be modeled as the following formula, as discussed in Section 1.1.1:

$$ok_J \rightarrow (J \leftrightarrow \neg P), \quad ok_A \rightarrow (A \leftrightarrow (J \wedge D))$$

Note that the above formula fails to encode half of the CPT entries, where $ok_J =$

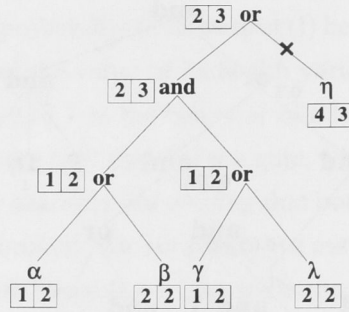


Figure 3.10: Pruning d-DNNF to improve heuristic accuracy.

to be measured.

3.3.2.1 Heuristic Based on Entropy of Wire

Since a wire X only has two values, its entropy can be written as:

$$\xi(X) = -(p_x \log p_x + p_{\bar{x}} \log p_{\bar{x}}) \quad (3.21)$$

where $p_x = \Pr(X = 1 \mid \mathbf{Y} = \mathbf{y})$ and $p_{\bar{x}} = \Pr(X = 0 \mid \mathbf{Y} = \mathbf{y})$ are the posterior probabilities of X having values 1 and 0, respectively, given the values \mathbf{y} of wires \mathbf{Y} whose values are known.

While $\xi(X)$ captures the uncertainty over the value of the wire, we can also interpret it as the average amount of information gain provided by measuring the wire. Hence as a first idea we consider selecting a wire with maximal entropy for measurement at each step.

3.3.2.2 Improving Heuristic Accuracy

This idea alone, however, did not work very well in our initial experiments as the diagnostic cost still remained quite high in many cases. As would be confirmed by subsequent experiments, this is largely due to that the (implicit) space of all diagnoses is generally very large and can include a large number of unlikely diagnoses, which tends to compromise the accuracy of the information gain provided by the entropy. The experiments to confirm this explanation are as follows.

When the d-DNNF compilation is produced, and before it is used to compute probabilities, we prune the d-DNNF graph so that models (satisfying variable assignments) corresponding to diagnoses with more than k broken gates are removed. A complete pruning is not easy; however, an approximation can be achieved in time

linear in the d-DNNF size, by a variant of the minimization procedure described in [Darwiche 2001], which is explained later.

It requires associating two integer registers with each node n of the d-DNNF, namely ur and dr , and a two-pass traversal through the d-DNNF described below. The ur register is updated during upward traversal and represents the minimum-cardinality of diagnoses under a node, whereas the dr register is updated during downward traversal and represents the upper bound on the fault-cardinality for a node which is used to prune branches emanating from the node whose ur exceeds the bound. The two passes of the procedure are as follows: Initialize $ur(n)$ to 0 and $dr(n)$ to $-\infty$ (least possible value) for all n . Traverse the d-DNNF so that children are visited before parents and for every leaf node, set $ur(n)$ to 1 if n is a negated health variable and 0 otherwise; for every or node, set $ur(n)$ to the minimum of the values of ur of its children; for every and node set $ur(n)$ to the sum of the values of ur of its children. Now traverse the d-DNNF so that parents are visited before children and set dr for the root node to the value k ; for every or node, remove every child p of n for which $ur(p) > dr(n)$ and for every remaining child v set $dr(v)$ to $dr(n)$ if $dr(n) > dr(v)$; for every child p of every and node, let t_p be the sum of the values of ur of all the other children and set $dr(p)$ to the value t_p if $t_p > dr(p)$.

An example is shown in Figure 3.10. The ur and dr registers are shown for each node. The branches labeled α , β , γ , and η are sub-graphs associated with hypothetical values for ur registers. The figure shows that the minimum-cardinality for every node (ur) is less than or equal to the bound (dr) except for the branch labeled η , which gets pruned accordingly.

We set the initial k to the number of actual faults in the experiments, and observed that a significant reduction of diagnostic cost resulted in almost all cases. This improved performance is apparently due to that the pruning updates the posterior probabilities of all variables, making them more accurate since many unlikely diagnoses have been eliminated.

In practice, however, the number of faults is not known beforehand and choosing an appropriate k for the pruning can be nontrivial (note that k need not be exactly the same as the number of actual faults for the pruning to help). Interestingly, the following heuristic, which is the one we will actually use, appears to achieve a similar performance gain in an automatic way: We select a component that has the highest posterior probability of failure (an idea from [Heckerman et al. 1995]), and then from the variables of that component, measure the one that has the highest entropy. This

Algorithm 3.3.1 Probabilistic sequential diagnosis**function** PSD(C, Δ, D, y, k)**inputs:** $\{C$: circuit $\}, \{\Delta$: d-DNNF $\}, \{y$: measurements $\}, \{k$: fault cardinality $\}, \{D$: ordered set of known faults $\}$ **output:** $\langle D, y \rangle$

- 1: REDUCE ($\Delta, D, k - |D|$) if D has changed
- 2: Given y on variables Y , EVALUATE (Δ, y) to obtain $\Pr(y)$
- 3: DIFFERENTIATE (Δ) to obtain $\Pr(X = 1, y) \forall$ wires X
- 4: Deduce fault as $D = D \cup \{X : \Pr(okX = 1, y) = 0\}$
- 5: **if** D has changed && MEETSCRITERIA(Δ, D, y) **then**
- 6: **return** $\langle D, y \rangle$
- 7: **end if**
- 8: Measure wire X which is the best under a given heuristic
- 9: Add the measured value x of X to y , and go back to line 1

heuristic does not require the above pruning of the d-DNNF, and appears to improve the diagnostic cost to a similar extent by focusing the measurement selection on the component most likely to be broken (empirical results to this effect are given and discussed in Section 3.4).

3.3.3 The Algorithm

We start by encoding the circuit as a logical formula as described in Section 3.3.1, where a subset of the variables are associated with numbers representing the prior fault probabilities and probabilities involved in the fault models of the gates, which is then compiled into d-DNNF Δ .

The overall sequential diagnosis process we propose is summarized in Algorithm 3.3.1. The inputs are a circuit C , its d-DNNF compilation Δ , the set of faults D (which is empty but will be used in the hierarchical approach), a set of known values y of wires, and an integer k specifying the fault cardinality bound (this is for running the model pruning experiments described in Section 3.3.2.2, and is not required for diagnosis using our final heuristic). We reduce Δ by pruning some models (line 1) when the fault cardinality bound k is given. We then evaluate (line 2) and differentiate (line 3) Δ , select a measurement point and take the measurement (line 8), and repeat the process (line 9) until the stopping criteria are met (line 5).

The stopping criteria on Line 5 are given earlier in Section 3.2 as the goal condition, i.e., we stop when the abnormal observation is explained by all the faulty gates D already identified assuming that other gates are healthy. A faulty gate X is identified when $\Pr(okX = 1, y) = 0$ where y are the values of wires that are already known, and as mentioned earlier these probabilities are obtained for all gates simultaneously

in the d-DNNF differentiation process. Finally, the condition that the current set of faulty gates, with health modes \mathbf{H}_f , explains the observation is satisfied when $\Pr(\mathbf{H}_f = \mathbf{0}, \mathbf{H} \setminus \mathbf{H}_f = \mathbf{1}, \mathbf{y}) > 0$, which is checked by a single evaluation of the original d-DNNF. The algorithm returns the actual faults together with the new set of known values of wires (line 6).

We proceed now to describe a thorough empirical evaluation of the new sequential diagnosis algorithms.

3.4 Experimental Results

This section provides an empirical evaluation of our new diagnostic system, referred to as SDC (sequential diagnosis by compilation), that implements the approaches described in Section 3.3.2. All experiments were conducted using the same hardware and with the same memory and time limits described in Section 2.5.

We generated test cases for single- and multiple-fault scenarios using ISCAS-85 benchmark circuits. For single faults, we simulated the equal prior probability of faults by generating n fault scenarios for each circuit, where n equals the number of gates in the circuit: Each scenario contains a different faulty gate. We then randomly generated 5 test cases for each of these n scenarios. Doing the same for multiple-fault scenarios would not be practical due to the large number of combinations, so for each circuit we simply generated 500 (for circuits up to c1355) or 100 (for remaining circuits) random scenarios with the given fault cardinality and a random test case for each scenario.

Each test case is a faulty circuit where some gates give incorrect outputs. The inputs and outputs of the circuit are observed, and actual values of its wires are used to simulate the results of taking measurements. We use $\Pr(\text{ok}X = 1) = 0.9$ for all gates X of the circuit. Note that such cases, where all gates fail with equal probability, are conceivably harder to solve as the diagnoses will tend to be less differentiable. Then, for each gate, the two output values are given equal probability when the gate is faulty. Again, this will tend to make the cases harder to solve due to the high degree of uncertainty. For each circuit and fault cardinality, we report the cost (number of measurements taken) and time (in seconds) to locate the faults, averaged over all test cases solved.

We start with a comparison of SDC with GDE and show that SDC scales to much larger circuits, hence illustrating the effectiveness of our new heuristic (along with the

size	system	single-fault		double-fault		triple-fault	
		cost	time	cost	time	cost	time
13	GDE	3.6	2.0	3.8	1.81	4.0	1.9
	SDC	3.6	0.01	3.4	0.01	2.8	0.01
14	GDE	3.5	6.66	3.3	15.1	3.0	14
	SDC	4.2	0.01	2.9	0.01	2.9	0.01
15	GDE	3.4	111	3.5	88	4.3	299
	SDC	3.9	0.01	3.4	0.01	3.7	0.01
16	GDE	3.3	398	3.5	556	3.2	509
	SDC	3.5	0.01	3.3	0.01	2.8	0.01
17	GDE	3.7	2876	4.6	4103	4.5	2067
	SDC	3.8	0.01	4.2	0.01	4.2	0.01

Table 3.1: Comparison with GDE.

new way to compute probabilities).

3.4.1 Comparison with GDE

We used the publicly available version of GDE [Forbus and de Kleer 1993] for the comparison, downloadable from <http://www.qrg.northwestern.edu/BPS/readme.html>. GDE uses ATCON, a constraint language developed using the LISP programming language, to represent diagnostic problem cases. A detailed account of this language is available in [Forbus and de Kleer 1993]. Further, it employs an interactive user interface that proposes measurement points with their respective costs and lets the user enter outcomes of measurements. For the purpose of comparison we translated our problem descriptions to the language accepted by GDE, and also modified GDE to automatically read in the measurement outcomes from the input problem description. We also compiled the LISP code to machine dependent binary code using the native C compiler to improve run-time performance.

GDE quickly ran out of memory even on the smallest circuit (c432) in ISCAS-85. We observed that the *Assumption Based Truth Maintenance System* [Forbus and de Kleer 1993], used by GDE as a reasoning system to generate minimal diagnoses, actually generated exponentially many sets of *assumptions* (about values of system variables, called *environments*) before running out of memory. Note that results in [de Kleer et al. 1992] were obtained by using a simple reasoning system which assumes a single fault, in which case the number of diagnoses is bounded by the number of gates. We also note that GDE has to recompute the minimal diagnoses after each measurement [de Kleer and Williams 1987], while in our approach the circuit (or its

circuit	system	pruning	single-fault		double-fault		five-fault	
			cost	time	cost	time	cost	time
c432 (160 gates)	RAND	no	92.3	20.7	97.7	23.2	117.8	26.5
		yes	4.5	11.4	36.8	12.4	99.7	17.2
	SDC(ew)	no	42.0	16.6	42.5	21.3	68.4	25.5
		yes	3.7	11.1	8.6	12.0	33.8	12.8
	SDC(fp)	no	6.7	11.7	6.4	12.5	9.4	13.0
		yes	4.3	11.0	5.0	12.3	9.1	12.6
c499 (202 gates)	RAND	no	109.6	0.8	120.6	1.2	150.0	1.4
		yes	5.5	0.2	20.1	0.2	104.9	0.7
	SDC(ew)	no	58.1	0.7	54.0	0.5	95.8	0.8
		yes	3.6	0.2	3.7	0.2	35.7	0.3
	SDC(fp)	no	6.5	0.2	4.3	0.2	7.2	0.2
		yes	4.8	0.2	3.0	0.2	7.1	0.2
c880 (383 gates)	RAND	no	221.0	1.9	251.3	1.9	306.4	2.3
		yes	5.4	0.2	47.3	0.3	205.7	1.3
	SDC(ew)	no	26.8	0.3	32.8	0.4	79.0	0.7
		yes	4.0	0.2	6.8	0.2	30.5	0.4
	SDC(fp)	no	10.8	0.2	9.2	0.2	15.8	0.3
		yes	5.6	0.2	6.7	0.2	14.0	0.3
c1355 (546 gates)	RAND	no	327.2	4.3	365.7	5.7	437.4	5.6
		yes	7.4	0.4	59.0	1.0	328.6	3.5
	SDC(ew)	no	82.6	1.3	91.2	1.5	203.9	3.4
		yes	4.9	0.4	5.5	0.4	65.9	1.1
	SDC(fp)	no	34.1	0.8	14.8	0.5	19.3	0.8
		yes	8.0	0.4	9.4	0.6	18.4	0.6

Table 3.2: Effectiveness of heuristic.

abstraction) is compiled only once and the same compilation used to compute all the probabilities required during the whole diagnosis session.

To enable a useful comparison, we extracted a set of small subcircuits from the ISCAS-85 circuits: 50 circuits of size 13, 14, 15 and 16, and 10 circuits of size 17. For each circuit we randomly generated 5 single-fault, 5 double-fault, and 5 triple-fault scenarios, and one test case (input/output vector) for each fault scenario.

Table 3.1 summarizes the comparison between GDE and SDC (baseline) on these benchmarks, where the first column shows the size the circuit. It is clear that the running time of GDE increases by roughly an order of magnitude with an increase of just one gate in the circuit size. SDC performs as well as GDE in terms of diagnostic cost, and solves every case instantly while GDE takes up to more than an hour.

3.4.2 Larger Benchmarks

To evaluate the performance of SDC on the larger ISCAS-85 circuits, we have again conducted three sets of experiments, this time involving single, double, and five

faults, respectively. As the version of GDE available to us is unable to handle these circuits, in order to provide a systematic reference point for comparison we have implemented a random strategy where a random order of measurement points is generated for each circuit and used for all the test cases. This strategy also uses the d-DNNF to check whether the stopping criteria have been met.

Table 3.2 shows the comparison between the random strategy and SDC using the baseline approach with two different heuristics, one based on entropies of wires alone (ew) and the other based also on failure probabilities (fp). For each of the three systems we ran the same set of experiments with and without pruning the d-DNNF (using the known fault cardinality as described in Section 3.3.2.2), indicated in the third column of the table. Only the test cases for the first four circuits could be solved. For other circuits the failure occurred during the compilation phase, and hence affected both the random strategy and SDC.

It is clear that the diagnostic cost is significantly lower with both heuristics of SDC than with the random strategy whether or not pruning has been used. It is also interesting to note that pruning significantly reduces the diagnostic cost for the random and SDC-ew strategies, but has much less effect on SDC-fp except in a few cases (c1355 single-fault).

Comparing the diagnostic cost of the two heuristics of SDC shows that SDC-fp dominates SDC-ew across the board, both with and without pruning. With few exceptions (e.g., c1355 and c880, single-fault), SDC-fp without pruning exhibits an overall performance comparable to or better than that of SDC-ew with pruning. As mentioned earlier, this indicates that the combination of failure probabilities and wire entropies appears to achieve an effect similar to that of pruning; in other words, it achieves an automatic pruning effect without requiring a fault cardinality bound.

3.5 Conclusions

We have presented a new system, called SDC, for sequential diagnosis that significantly advances the state of the art by solving, for the first time, a set of nontrivial multiple-fault diagnostic cases on large benchmark circuits, where no restriction on the cardinality of faults is assumed before hand. On the small benchmarks that can be solved by the previous GDE system, SDC also exhibits much higher efficiency and comparable performance in terms of diagnostic cost.

Hierarchical Sequential Diagnosis

This chapter combines the hierarchical diagnosis approach of Chapter 2 with the sequential diagnosis method of Chapter 3 to scale sequential diagnosis to larger systems. Some of the material presented in Section 4.1 was published in [Siddiqi and Huang 2008]. Here we further extend this work with a novel technique of *gate cloning* to reduce the abstraction size of a system that scales sequential diagnosis to some of the largest benchmark systems. We present our techniques in Sections 4.1 and 4.2, describing the *hierarchical sequential diagnosis* and *gate cloning* respectively, whereas empirical evidence for their effectiveness is given in Section 4.3. Conclusions are drawn in Section 4.4.

4.1 Hierarchical Approach

As discussed in Section 2.4, the basic idea of hierarchical diagnosis is that the compilation of the system model into d-DNNF will be more efficient and scalable when the number of system components is reduced. This can be achieved by abstraction, where subsystems, known as cones, are treated as single components. An example of a cone is depicted in Figure 3.6. The objective here is to use a single health variable and failure probability for the entire cone, hence significantly reducing the size of the encoding and the difficulty of compilation. Once a cone is identified as faulty in the top-level diagnosis, it can then be compiled and diagnosed, in a recursive fashion.

In Chapter 2, we only dealt with the task of computing diagnoses, which does not involve probabilities or measurement selection. In the context of sequential diagnosis, several additional techniques have been introduced, particularly in the computation of prior failure probabilities for the cones and the way measurement points are selected, detailed below.

4.1.1 Propositional Encoding

We start with a discussion of the hierarchical encoding for probabilistic reasoning, which is similar to the hierarchical encoding presented in Chapter 2. Specifically, for the diagnosis of the abstraction \mathbf{A}_C of the given circuit C , only the nodes corresponding to the gates $\mathbf{A}_C \setminus \mathbf{I}_C$ will have extra health nodes in the corresponding Bayesian network, which are the nodes $\{A, B, D, K, V\}$ in our example (\mathbf{I}_C stands for the set of inputs of the circuit C). The node okJ is removed from the Bayesian network in Figure 3.7, as J is a wire internal to the cone rooted at A .

In addition, we assume that a cone always outputs a wrong value when it fails (which is explained in Section 4.1.2), due to which no extra network parameter is needed to complete the CPT of the cone, but extra clauses modeling the abnormal behavior of the cone are required. For example, the encoding given in Section 3.3.1.2 for cone A in Figure 3.6 (in the dotted box) is modified as follows:

$$J \leftrightarrow \neg P, \quad okA \rightarrow (A \leftrightarrow (J \wedge D)), \quad \neg okA \rightarrow (A \not\leftrightarrow (J \wedge D))$$

The first part of the formula encodes the normal behavior of gate J (without a health variable); the next encodes the normal behavior of the cone; the last encodes that the cone outputs a wrong value when it fails. Other gates (that are not roots of cones) in the abstraction \mathbf{A}_C are encoded as described in Section 3.3.1.2.

The formulas for all the gates in a cone together encode a single CPT for the whole cone, which provides the conditional probability of the cone's output given the health and inputs of the cone, instead of the health and inputs of the gate at the root of the cone. For example, the above encoding is meant to provide the conditional probability of A given P, D , and okA (instead of J, D , and okA), where okA represents the health mode of the whole cone and is associated with its prior failure probability, which is initially unknown to us and has to be computed for all cones (explained below). Such an encoding of the whole circuit provides a joint probability distribution over the variables $\mathbf{A}_C \cup \mathbf{I}_C \cup \mathbf{H}$, where $\mathbf{H} = \{okX \mid X \in \mathbf{A}_C \setminus \mathbf{I}_C\}$.

4.1.2 Prior Failure Probabilities for Cones

When a cone is treated as a single gate, its prior probability of failure as a whole can be computed given the prior probabilities of gates and cones inside it. Specially we have to compute the prior probabilities of variable okX for every cone X . We do this by creating two copies Δ_h and Δ_f of the cone, where Δ_h models only the healthy

behavior of the cone (without health variables), and Δ_f includes the faulty behavior as well (i.e., the full encoding described in Section 3.3.1.2). The outputs of both Δ_h and Δ_f are collected into an XOR-gate X . When the output of XOR-gate X equals to 1 it enforces that both of its inputs must be different in value. We then compute the probability $\Pr(X = 1)$, which gives the probability of the outputs of Δ_h and Δ_f being different. We compute this probability by compiling the encoding of the cone into d-DNNF and evaluating it under $X = 1$.

Note that this procedure itself is also abstraction based and hierarchical, performed bottom-up with the probabilities for the inner cones computed before those for the outer ones. Also note that it is performed only once per circuit as a preprocessing step.

4.1.3 Measurement Point Selection and Stopping Criteria

In principle, the heuristic to select variables for measurement and the stopping criteria are the same as in the baseline approach; however, a couple of details are worth mentioning.

First, when diagnosing the abstraction of a given circuit (or cone) C , the measurement candidates are restricted to variables $\mathbf{A}_C \cup \mathbf{I}_C$, ignoring the internal wires of the maximal cones—those are only measured if a cone as a whole has been found faulty.

Second, when diagnosing a cone, it is generally important to have full knowledge of the values of its inputs before a final diagnosis is concluded. A diagnosis of a cone concluded with only partial knowledge of its inputs may never be part of any valid global diagnosis. The reason is that the diagnosis of the cone assumes that the unknown inputs can take either value, while in reality their values may become fixed when wires in other parts of the circuit are measured. To avoid this situation while retaining the effectiveness of the heuristic, we modify the measurement point selection as follows when diagnosing a cone. After selecting a gate with the highest probability of failure, we consider the wires of that gate *plus* the inputs of the cone, and measure the one with the highest entropy. We do not conclude a diagnosis for the cone until values of all its inputs become known (through measurement or deduction), except when the health of all the gates in the cone has been determined without knowing all the inputs to the cone (it is possible to identify a faulty gate, and with strong fault models also a healthy gate, without knowing all its inputs). Generally, the cost increase due to this is insignificant because when a cone is concluded as faulty in the abstraction of a circuit, the values of a significant number, if not all, of its inputs are

Algorithm 4.1.1 Hierarchical probabilistic sequential diagnosis

```

function HPSD( $C, u_C, k$ )
inputs:  $\{C : \text{circuit}\}, \{u_C : \text{obs. across circuit}\} \{k : \text{fault cardinality}\}$ 
local variables:  $\{B, D, T : \text{set of gates}\} \{y, z, u_G : \text{set of measurements}\} \{i, k' : \text{integer}\}$ 
output:  $\{\text{pair} \langle D, u_C \rangle\}$ 
1:  $\Delta \leftarrow \text{COMPILE2DDNNF}(A_C, u_C)$ 
2:  $i \leftarrow 0, D \leftarrow \phi, y \leftarrow u_C$ 
3:  $\langle B, y \rangle \leftarrow \text{PSD}(C, \Delta, B, y, k)$ 
4: for  $\{; i < |B|; i ++\}$  do
5:    $G \leftarrow \text{ELEMENT}(B, i)$ 
6:   if  $G$  is a cone then
7:      $z \leftarrow y \cup \text{IMPLICATIONS}(\Delta, y)$ 
8:      $u_G \leftarrow \{x : x \in z, X \in I_G \cup O_G\}$ 
9:      $k' \leftarrow k - |D| - |B| + i + 2$ 
10:     $\langle T, u_G \rangle \leftarrow \text{HPSD}(D_G \cup I_G, u_G, k')$ 
11:     $y \leftarrow y \cup u_G, D \leftarrow D \cup T$ 
12:     $\text{EVALUATE}(\Delta, y), \text{DIFFERENTIATE}(\Delta)$ 
13:   else
14:      $D \leftarrow D \cup \{G\}$ 
15:   end if
16: end for
17:  $z \leftarrow y \cup \text{IMPLICATIONS}(\Delta, y)$ 
18:  $u_C \leftarrow u_C \cup \{x : x \in z, X \in I_C \cup O_C\}$ 
19: if  $\text{MEETSCRITERIA}(C, D, y)$  then
20:   return  $\langle D, u_C \rangle$ 
21: else
22:   goto line 3
23: end if

```

often already known.

4.1.4 The Algorithm

Pseudocode for the hierarchical approach is given in Algorithm 4.1.1 as a recursive function. The inputs are a circuit C , a set of known values u_C of wires at the inputs I_C and outputs O_C of the circuit, and again the optional integer k specifying the fault cardinality bound for the purpose of experimenting with the effect of model pruning. We start with the d-DNNF compilation of the abstraction of the given circuit (line 1) and then use the function PSD from Algorithm 3.3.1 to get a diagnosis B of the abstraction (line 3), assuming that the measurement point selection and stopping criteria in Algorithm 3.3.1 have been modified according to what is described in Section 4.1.3. The abstract diagnosis B is then used to get a concrete diagnosis D in a loop (lines 4–14). Specifically, if a gate $G \in B$ is not the root of a cone, then it is added to D (line 14); otherwise cone G is recursively diagnosed (line 10) and the result of it added to D

(line 11).

Before recursively diagnosing a cone G , we compute an abnormal observation \mathbf{u}_G at the inputs and the output ($\mathbf{I}_G \cup \{G\}$) of the cone G . The values of some of G 's inputs and output will have been either measured or deduced from the current set of measurements. The value of a gate X is implied to be x under the measurements \mathbf{y} if $\Pr(X = \neg x, \mathbf{y}) = 0$, which is easy to check once Δ has been differentiated under \mathbf{y} . The function `IMPLICATIONS`(Δ, \mathbf{y}) (lines 7 and 15) implements this operation, which is used to compute the partial abnormal observation \mathbf{u}_G (line 8). A fault cardinality bound k' for the cone G is then inferred (line 9), and the algorithm called recursively to diagnose G , given \mathbf{u}_G and k' .

The recursive call returns the faults \mathbf{T} inside the cone G together with the updated observation \mathbf{u}_G . The observation \mathbf{u}_G may contain some new measurement results regarding the variables $\mathbf{I}_G \cup \{G\}$, which are added to the set of measurements \mathbf{y} of the abstraction (line 11); other measurement results obtained inside the cone are ignored due to reasons explained in Section 4.1.3. The concrete diagnosis \mathbf{D} is augmented with the faults \mathbf{T} found inside the cone (line 11), and Δ is again evaluated and differentiated in light of the new measurements (line 12).

After the loop ends, the variable \mathbf{u}_C is updated with the known values of the inputs \mathbf{I}_C and outputs \mathbf{O}_C of the circuit C (line 16). The stopping criteria are checked for the diagnosis \mathbf{D} (line 17) and if met the function returns the pair $\langle \mathbf{D}, \mathbf{u}_C \rangle$ (line 18); otherwise more measurements are taken until the stopping criteria (line 17) have been met. Note that it is quite possible that the abstract diagnosis \mathbf{B} meets the stopping criteria while the concrete diagnosis \mathbf{D} obtained from \mathbf{B} does not, an example of which is given in Section 4.1.4.1.

Since \mathbf{D} can contain faults from inside the cones, the compilation Δ cannot be used to check the stopping criteria for \mathbf{D} (note the change in the parameters to the function `MEETSCRITERIA` at line 17) as the probabilistic information regarding wires inside cones is not available in Δ . The criteria are checked as follows instead: We first propagate the values of inputs in the circuit, and then propagate the fault effects of gates in \mathbf{D} , one by one, by flipping their values and propagating them towards the circuit outputs in such a way that deeper faults are propagated first (see Chapter 2), and then check the values of circuit outputs obtained for equality with those in the observation (\mathbf{y}).

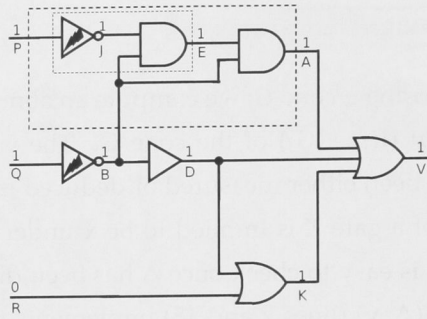


Figure 4.1: A faulty circuit with faults at B and J.

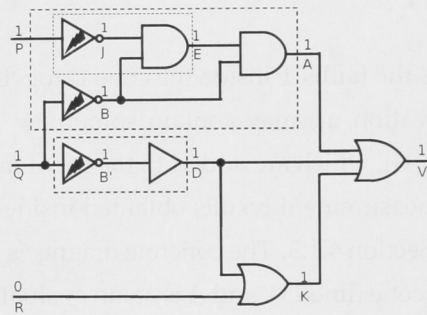


Figure 4.2: Creating a clone B' of B according to D.

4.1.4.1 Example

Suppose that we diagnose the abstraction of the circuit in Figure 3.6, with the observation $\mathbf{u}_C = \{P = 1, Q = 1, R = 0, V = 1\}$, and take the sequence of measurements $\mathbf{y} = \{D = 1, K = 1, A = 1\}$. It is concluded, from the abstract system model, that given the values of P and D, the value 1 at A is abnormal. So the algorithm concludes a fault at A. Note that $Q = 1$ and $D = 1$ suggests the presence of another fault besides A triggering the measurement of gate B, which is also found faulty. The abstract diagnosis $\{A, B\}$ meets the stopping criteria with respect to the abstract circuit. We then enter the diagnosis of cone A by a recursive call with observation $\mathbf{u}_A = \{P = 1, D = 1, A = 1\}$. The only unknown wire J is measured and found faulty, which explains the observation at cone's output A, given its inputs P and D. The recursion terminates and the abstract diagnosis $\mathbf{B} = \{A, B\}$ generates the concrete diagnosis $\mathbf{D} = \{J, B\}$, which meets the stopping criteria and the algorithm terminates.

4.2 Gate Cloning

In the preceding section, we have proposed an abstraction based approach to sequential diagnosis, which reduces the complexity of compilation and diagnosis by reducing the number of system components to be diagnosed. We now take one step further, aiming to handle systems that are so large that they remain intractable even after abstraction, as is the case for the largest circuits in the ISCAS-85 benchmark suite.

Our solution is a novel method that systematically modifies the structure of a circuit to reduce the size of its abstraction. Specifically, we select a gate G with parents \mathbf{P} that is not part of a cone and hence cannot be abstracted away in hierarchical diagnosis, and create a clone G' of it according to some of its parents $\mathbf{P}' \subset \mathbf{P}$ in the sense that G' inherits all the children of G and feeds into \mathbf{P}' while G no longer feeds into \mathbf{P}' (see Figures 4.1 and 4.2 for an example). The idea is to create a sufficient number of clones of G so that G and its clones become part of some cones and hence can be abstracted away. Repeated applications of this operation can allow an otherwise unmanageable system to have a small enough abstraction for compilation and diagnosis to succeed. The hierarchical algorithm is then extended to diagnose the new circuit and the result mapped to the original circuit. We show that we can now solve almost all the benchmark circuits, using this approach.

Before we go into the details of the new method, we differentiate it from a technique known as *node splitting* [Choi et al. 2007], which is used to solve MPE queries on a Bayesian network. Node splitting breaks enough number of edges between nodes from the network such that the MPE query on the resulting network becomes easy to solve. A broken edge is replaced with a root variable with a uniform prior. The resulting network is a relaxation or approximation of the original in that its MPE solution, which may be computed from its compilation, gives an upper bound on the MPE solution of the original network. A depth-first branch and bound search algorithm then searches for an optimal solution using these bounds to prune its search space. A similar approach is also used to solve Weighted Max-SAT problems [Pipatsrisawat and Darwiche 2007]. We shall give a detailed account of this technique in Chapter 5.

This version of node splitting is not directly applicable in the present setting for the following reasons. If edges in a circuit are broken and redirected into new root variables (primary inputs), the resulting circuit represents a different input-output function from that of the original circuit. The abnormal observation on the original circuit may hence become a normal one on the new circuit (if the wires through which

the fault propagates are broken), eliminating the basis for diagnosis. Our technique of gate cloning, which can also be viewed as a version of node splitting, introduces clones of a gate instead of primary inputs and preserves the input-output function of the circuit. Also, the new circuit is a relaxation of the original in that its diagnoses are a superset of those of the original.

We now formally define gate cloning.

Definition 4.2.1 (Gate Cloning)

Let G be a gate in a circuit C with parents P . We say that G is **cloned according to parents** $P' \subset P$ when it results in a circuit C' that is obtained from C as follows:

- The edges going from G to its parents P' are removed.
- A new gate G' functionally equivalent to G is added to the circuit such that G' shares the inputs of G and feeds into each of P' .

Figures 4.1 and 4.2 show an example where creating a clone B' of B according to $\{D\}$ results in a new circuit whose abstraction contains only the gates $\{A, D, K, V\}$, whereas the abstraction of the original circuit contains also gate B .

4.2.1 Choices in Gate Cloning

There are two choices to be made in gate cloning: Which gates do we clone, and for each of them how many clones do we create and how do they split the parents?

Since the goal of cloning is to reduce the abstraction size, it is clear that we only wish to clone those gates that lie in the abstraction (i.e., not within cones). Among these, cloning of the root of a cone cannot reduce the abstraction size as it will destroy the existing cone, reintroducing some of the gates inside the cone into the abstraction. For example, cloning D according to K in Figure 4.2 will produce a circuit where D and its clone can be abstracted away but B' is no longer dominated by D and hence is reintroduced into the abstraction. Therefore, the final candidates for cloning are precisely those gates in the abstract circuit that are not roots of cones. Note that the order in which these candidates are processed is unimportant in that each when cloned will produce an equal reduction, namely a reduction of precisely 1 in the abstraction size, if any.

It then remains to determine for each candidate how many clones to create and how to connect them to the parents. To understand our final method, it helps to consider a naive method that simply creates $|P| - 1$ clones (where P is the set of parents)

and has each clone, as well as the original, feed into exactly one parent. This way every parent of the gate becomes the root of a cone and the gate itself and all its clones are abstracted away. In Figure 4.1, for example, B has three parents $\{E, A, D\}$, and this naive method would create two clones of B for a total of three instances of the gate to split the three parents, which would result in the same abstraction as in Figure 4.2.

The trick now is that the number of clones can be reduced by knowing that some parents of the gate may lie in the same cone and a single clone of the gate according to those parents will be sufficient for that clone to be abstracted away. In the example of Figure 4.1, again, the parents E, A of B lie in the same cone A and it would suffice to create a single clone of B according to $\{E, A\}$, resulting in the same, more efficient cloning as in Figure 4.2.

More formally, we partition the parents of a gate G into subsets P_1, P_2, \dots, P_q such that those parents of G that lie in the same cone are placed in the same subset and the rest in separate ones. We then create $q - 1$ clones of G according to any $q - 1$ of these subsets, resulting in G and all its clones being abstracted away. This process is repeated for each candidate gate until the abstraction size is small enough or no further reduction is possible.

4.2.2 Diagnosis with Gate Cloning

The new circuit is functionally equivalent to the original and has a smaller abstraction, but is not equivalent to the original for diagnostic purposes. As the new model allows a gate and its clones to fail independently of each other, it is a relaxation of the original model in that the diagnoses of the new circuit form a superset of those of the original. Specifically, each diagnosis of the new circuit that assigns the same health state to a gate and its clones for all gates corresponds to a diagnosis of the original circuit; other diagnoses are spurious and are to be ignored.

The core diagnosis process given in Algorithm 4.1.1 continues to be applicable on the new circuit, with only two minor modifications necessary. First, the spurious diagnoses are (implicitly) filtered out by assuming the same health state for all clones (including the original) of a gate as soon as the health state of any one of them is known. Second, whenever measurement of a clone of a gate is proposed, the actual measurement is taken on the original gate in the original circuit, for obvious reasons (in other words, the new circuit is used for reasoning and the original for measurements).

In principle, the presence of spurious diagnoses in the model can potentially skew

circuit	pruning	single-fault		double-fault		five-fault	
		cost	time	cost	time	cost	time
c432 (64 cones)	no	15.4	0.4	15.8	0.5	22.2	0.5
	yes	4.9	0.3	10.4	0.4	21.5	0.4
c499 (90 cones)	no	7.3	0.1	5.8	0.1	10.5	0.2
	yes	4.5	0.1	3.9	0.1	9.6	0.2
c880 (177 cones)	no	9.5	0.1	10.2	0.1	17.4	0.2
	yes	5.6	0.1	7.6	0.1	16.3	0.2
c1355 (162 cones)	no	9.3	0.3	8.2	0.2	14.0	0.3
	yes	5.8	0.2	6.3	0.2	14.4	0.3
c1908 (374 cones)	no	11.0	222	17.1	587	34.9	505
	yes	3.0	214	8.5	463	32.4	383
c2670 (580 cones)	no	16.3	213	19.2	172	25.4	58
	yes	6.5	196	13.3	90	24.3	45

Table 4.1: Effectiveness of abstraction.

the measurement point selection heuristic (at least in the early stages of diagnosis, before the spurious diagnoses are gradually filtered out). However, by using smaller benchmarks that could be diagnosed both with and without cloning, we conducted an empirical analysis which indicates, interestingly, that the overall diagnostic cost is only slightly affected. We discuss this in more detail in Section 4.3.2.

4.3 Experimental Results

We present the experiments in two subsections demonstrating the effectiveness of the two techniques proposed in this chapter, namely the hierarchical sequential diagnosis, and gate cloning. The experiments are conducted on test cases used in Section 3.4 (of the last chapter) for diagnosing ISCAS-85 benchmark circuits using our baseline approach.

4.3.1 Effectiveness of Abstraction

We now report, in Table 4.1, the results of repeating the same experiments with SDC-fp (failure probabilities based heuristic) using the hierarchical approach and compare them with the corresponding results (of our baseline approach) in Table 3.2.

Most notably, the running time generally reduces for all cases and we are now able to handle two more circuits, namely c1908 and c2670, solving 139 of 300 cases for c1908 and 298 of 300 cases for c2670 in the three categories combined. In terms

circuit	total gates	abstraction size	cloning time	total clones	abstraction size after cloning
c432	160	59	0.03	27	39
c499	202	58	0.02	0	58
c880	383	77	0.1	24	57
c1355	58	58	0.05	0	58
c1908	880	160	0.74	237	70
c2670	1193	167	0.77	110	116
c3540	1669	353	5.64	489	165
c5315	2307	385	3.6	358	266
c6288	2416	1456	0.16	0	1456
c7552	3512	545	6.68	562	378

Table 4.2: Results of preprocessing step of cloning.

circuit	single-fault		double-fault		five-fault	
	cost	time	cost	time	cost	time
c432	7.2	10.3	6.6	7.8	9.6	9.7
c880	11.2	0.2	9.3	0.2	16.2	0.3

Table 4.3: Effect of gate cloning on diagnostic performance.

of diagnostic cost, in most cases the hierarchical approach is comparable to the baseline approach. On c432, the baseline approach consistently performs better than the hierarchical in each fault cardinality, while the reverse is true on c1355.

The results indicate that the main advantage of hierarchical approach is that larger circuits can be solved. For circuits that can also be solved by the baseline approach, hierarchical approach may help reduce the diagnostic cost by quickly finding faulty portions of the circuit, represented by a set of faulty cones, and then directing the measurements inside them, which can result in more useful measurements. On the other hand, it may suffer in cases where it has to needlessly go through hierarchies to locate the actual faults, while the baseline version can find them more directly and efficiently. Finally, we note that pruning helps further reduce the diagnostic cost to various degrees as with the baseline approach.

4.3.2 Effectiveness of Gate Cloning

In this subsection we discuss the experiments with gate cloning. We show that cloning does not significantly affect diagnostic cost and allows us to solve much larger circuits, in particular, nearly all the circuits in the ISCAS-85 suite.

Table 4.2 shows the result of the preprocessing step of cloning on each circuit. The columns give the name of the circuit, the total number of gates in that circuit, the size of the abstraction of the circuit before cloning, the time spent on cloning, the total

circuit	single-fault		double-fault		five-fault	
	cost	time	cost	time	cost	time
c432	15.2	0.1	14.8	0.1	20.2	0.1
c880	8.8	0.1	9.3	0.1	15.8	0.2
c1908	13.6	2.8	18.3	5.0	35.4	5.1
c2670	13.5	4.5	15.3	0.7	20.1	2.3
c3540	27.8	382	30.5	72.5	36.1	108.6
c5315	7.2	2.5	21.1	5.9	24.4	6.6
c7552	70.6	1056	43.1	129.0	104.8	1108

Table 4.4: Hierarchical sequential diagnosis with gate cloning (c499 and c1355 omitted as they are already easy to diagnose and cloning does not lead to reduced abstraction).

number of clones created in the circuit, and the abstraction size of the circuit obtained after cloning. On all circuits except c499, c1355, and c6288, a significant reduction in the abstraction size has been achieved. c6288 appears to be an extreme case with a very large abstraction that lacks hierarchy, while gates in the abstractions of c499 and c1355 are all roots of cones, affording no opportunities for further reduction (note that these two circuits are already very simple and easy to diagnose).

We now investigate the effect of gate cloning on diagnostic performance. To isolate the effect of gate cloning we use the baseline version of SDC (i.e., without abstraction). Table 4.3 summarizes the performance of baseline SDC with cloning on the circuits c432 and c880. Comparing these results with the corresponding entries in Table 3.2 shows that the overall diagnostic cost is only slightly affected by cloning. We further observed that in a significant number of cases, the proposed measurement sequence did not change after cloning, while in most of the other cases it changed only insubstantially. Moreover, in a number of cases, although a substantially different sequence of measurements was proposed, the actual diagnostic cost did not change much. Finally, note that the diagnosis time in the case of c432 has reduced after cloning, which can be ascribed to the general reduction in the complexity of compilation due to a smaller abstraction.

Our final set of experimental results, summarized in Table 4.4, illustrates the performance of hierarchical sequential diagnosis with gate cloning—the most scalable version of SDC. All the test cases for circuits c1908 and 2670 were now solved, and the largest circuits in the benchmark suite could now be handled: All the cases for c5315, 165 of the 300 cases for c3540, and 157 of the 300 cases for c7552 were solved. In terms of diagnostic cost, cloning generally resulted in a slight improvement. In terms of time, the difference is insignificant for c432 and c880, and for the larger circuits (c1908 and c2670) diagnosis with cloning was clearly more than an order of magnitude faster.

4.4 Conclusions

We have presented two new techniques that scale the sequential diagnosis to the largest benchmark systems. Specifically, we first apply the abstraction based approach to our sequential diagnosis method of Chapter 3. We then go a step further to reduce the abstraction size of a system by a novel approach of *gate cloning*. The new techniques allow us to diagnose for the first time almost all the circuits in the ISCAS-85 benchmark suite with good performance in terms of diagnostic cost.

4.3 Conclusions

The first part of the paper is devoted to the general theory of hierarchical sequential diagnosis. In the second part, the theory is applied to the diagnosis of a system with a hierarchical structure. The third part is devoted to the application of the theory to the diagnosis of a system with a hierarchical structure. The fourth part is devoted to the application of the theory to the diagnosis of a system with a hierarchical structure. The fifth part is devoted to the application of the theory to the diagnosis of a system with a hierarchical structure.

Computing Most Probable Explanations

This chapter is based upon work published in [Siddiqi and Huang 2009]. Here we present a new heuristic to dynamically order variables and their values in branch-and-bound search for a common diagnostic query known as MPE. The structure of this chapter is as follows: An introduction of the chapter is given in Section 5.1. We formally define the problem of MPE and introduce some notation and definitions in Section 5.2. We then review two types of exact methods for computing MPE in Sections 5.3 and 5.4, based on inference and search, respectively. The new heuristic is discussed in Section 5.5, followed by a comprehensive empirical analysis in Section 5.6. Conclusions are drawn in Section 5.7.

5.1 Introduction

In Bayesian networks, an MPE is a most likely instantiation of all network variables given a piece of evidence. Solving (the decision version of) an MPE query is NP-hard [Shimony 1994].

When networks continue to grow in size and complexity, inference based methods can fail to solve MPE queries, particularly by running out of memory, and one resorts instead to *search* algorithms. A search algorithm systematically goes through every assignment to network variables to find the one that has the highest probability. Search only takes a linear amount of memory for its execution, but takes time exponential in the network size due to exponential size space of assignments. A branch-and-bound search can reduce this complexity by computing a bound on the probability of any extension to a partial assignment, such that, if the bound becomes less than the best solution found so far then the search can backtrack, knowing that the current partial

assignment cannot lead to an optimal solution.

Choi et al. [2007] consider such a search technique that computes upper bounds on the MPE using an approximation of the network obtained through node splitting. Instead of all the network nodes, search is applied on a smaller set of nodes, known as split nodes, and the search time may be exponential only in the number of split nodes. Hence the efficiency of this approach depends on the number of split nodes, the quality of bounds, and the difficulty of computing the bounds. Therefore they have compared the performance of two heuristics for splitting nodes, where one, based on jointrees, tries to minimize the number of split nodes and generate a significantly easier network, which can also compromise the quality of bounds. The other, based on mini-buckets, tries to minimize the relaxation in the bounds, which may not generate significantly easier network. They have shown that jointree-based heuristic leads to orders of magnitude performance improvement on hard networks compared to the one based on mini-buckets.

However, Choi et al. [2007] have not addressed the impact of variable and value ordering on the efficiency of this approach. In this chapter we have taken the initiative to study various heuristics for variable and value ordering in pursuit of enhancing the scalability of this technique. We show that our new heuristics further improve efficiency significantly, extending the reach of exact algorithms to networks that cannot be solved by other known methods.

5.2 Notation and Definitions

We start with a formal definition of MPE. Let N be a Bayesian network with variables X , a *most probable explanation* for some evidence e is defined as:

$$\text{MPE}(N, e) =_{\text{def}} \arg \max_{x \sim e} \Pr_N(x)$$

where $x \sim e$ means that the assignments x and e are consistent, i.e, they agree on every common variable. We will also write $\text{MPE}_p(N, e)$ to denote the probability of the MPE solutions.

MPE queries involve potentially maximizing over all (uninstantiated) network variables, and are known to be NP-hard in general [Shimony 1994].

Before we discuss the various techniques used to solve MPE queries on a Bayesian network we define some helpful terms.

T	A	Pr(T, A)
1	1	0.24
1	0	0
0	1	0.36
0	0	0.24

Figure 5.1: Maximizing out S from the factor in Figure 3.2.

Definition 5.2.1 (Factor)

A factor is a function over a set of variables \mathbf{X} , mapping each instantiation of these variables to a non-negative number, denoted as $f(\mathbf{x})$.

For example every CPT of a Bayesian network is a factor, and so is the joint probability distribution in Figure 3.2. Assuming that a factor is represented in the form of a table its size is exponential in the number of its variables.

We now define two operations applicable on factors.

Definition 5.2.2 (Maximizing out)

Let $f(\mathbf{x})$ be a factor over variables \mathbf{X} and let X be a variable in \mathbf{X} . Maximizing out the variable X from f results in a another factor $\max_X f$ over variables $\mathbf{Y} = \mathbf{X} \setminus \{X\}$, which is defined as follows:

$$(\max_X f)(\mathbf{y}) = \text{def} \max_x f(x\mathbf{y})$$

For example, maximizing out S from the factor in Figure 3.2 results in the factor in Figure 5.1.

Definition 5.2.3 (Multiplying)

Let $f(\mathbf{x})$ and $g(\mathbf{y})$ be two factors over variables \mathbf{X} and \mathbf{Y} , respectively, sharing variables $\mathbf{S} = \mathbf{X} \cap \mathbf{Y}$. Multiplying f and g results in a factor fg over variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ by combining, in the new factor, those instantiations of \mathbf{X} and \mathbf{Y} that agree on variables \mathbf{S} , which map to numbers obtained by multiplying the numbers corresponding to instantiations being combined in their respective factors, i.e.

$$(fg)(\mathbf{z}) = \text{def} f(\mathbf{x})g(\mathbf{y})$$

where \mathbf{x} and \mathbf{y} are compatible with \mathbf{z} , i.e. $\mathbf{x} \sim \mathbf{z}$, $\mathbf{z} \sim \mathbf{y}$.

For example, the factor in Figure 3.2 is the result of multiplying the three factors shown in Figure 3.3.

The complexity of maximizing out is linear in the size of the given factor, whereas that of multiplying is linear in the size of the resulting factor.

Note that the given evidence can be represented as a set of factors each defined over a single variable that appears in the evidence. The instantiation of the variable in the factor that appears in the evidence maps to 1, and every other instantiation maps to 0.

5.3 Computing MPE by Inference

Given a Bayesian network and a piece of evidence, a naive method of computing the MPE for the evidence is to multiply together all the network CPTs and the evidence factors, which gives us a single factor that represents the joint probability distribution over all network variables, under the given evidence. The MPE can then be computed by maximizing over all the variables in the resulting factor. However, such an approach would have complexity exponential in the number of network nodes and will not be feasible in practice.

We discuss two most commonly used methods for exact MPE based on inference that are relevant to the current discussion. They provide a bound on the complexity of solving the MPE query, and make it possible to solve larger networks. In the following discussion we focus on computing the probability of the MPE. However, the actual MPE can also be computed by some extra book-keeping.

5.3.1 Variable Elimination

The key observation behind this approach is the result that if f and g are two factors, and X appears only in g , then

$$\max_X (f g) = f \max_X g.$$

The above result allows the operations of multiplication and maximization on the set of factors to be performed incrementally. Specifically, given a set of factors and the task of maximizing out a variable X , one only needs to multiply those factors that mention X and then maximize out X from the resulting factor. The resulting factor then replaces the factors that participated in the process from the given set, which eliminates X . The same is repeated for all the variables, and when all variables have been eliminated any remaining factors are multiplied together to get a single factor that contains only a single number against an empty instantiation representing the probability of the MPE. This process is called variable elimination.

We have the following complexity result: The complexity of variable elimination is dominated by the largest factor obtained during the elimination process, or alternatively, exponential in the treewidth of the elimination order (introduced in Section 2.2.1).

One particular implementation of variable elimination is *bucket elimination* [Dechter 1996], which provides the convenience of not having to search in the set of factors for those that mention a particular variable X , by placing the factors in an order. Specifically, each variable is associated with a *bucket* and buckets are given the same order as the elimination order. The factors are distributed to buckets such that a factor is placed in the first bucket (according to the order) whose variable is mentioned in the factor. When variable X is processed, the set of factors in its bucket are multiplied and X is maximized out, the resulting factor is then placed in the first proceeding bucket whose variable is mentioned in it.

5.3.2 Compilation

A more recent approach is based upon compilation of a network into arithmetic circuit, which was discussed in Section 3.1.3.2. Once a network has been compiled MPE for any evidence can be computed in time linear its size. Specifically, after the evidence has been set in the arithmetic circuit, it is evaluated bottom-up by treating every addition node as a maximization node. The value at the root node gives the probability of the MPE. The size of the arithmetic circuit may be exponential only in the treewidth of the network. However, we have discussed in Section 3.1.3.2 that compilation is known to exploit *local structure* so that complexity of compilation may be further reduced [Chavira and Darwiche 2005], and is exponential in the treewidth in the worst case only.

When networks continue to grow in size and complexity both the above mentioned methods can fail, particularly by running out of memory, and one resorts instead to either computing an approximate MPE or using algorithms based on *systematic search*. In the following we discuss the search based approach and show how it can be combined with approximation to gain efficiency.

5.4 Computing MPE by Systematic Search

A search method for computing exact MPE would systematically search amongst all the assignments to the network variables to select the most probable assignment. This

can be done by performing a depth-first search in the tree of assignments to network variables where the leaf nodes correspond one-to-one to complete instantiations of variables, and internal nodes correspond (one-to-one) to partial instantiations of variables. The outgoing edges from a tree node correspond to the possible instantiations of a variable. The probability of an instantiation can be computed using the chain rule in time linear in the number of network nodes. However, such a method would have complexity exponential in the network variables.

The complexity of search can be reduced if one is able to find an upper bound on the probability of any extension to the current partial assignment. Specifically, one can always backtrack when the upper bound becomes less than or equal to the current best solution, meaning that such a partial assignment would not lead to a better solution. This kind of search is called branch-and-bound search and is discussed later. First we discuss how such a bound can be computed.

5.4.1 Computing Bounds using Mini-Buckets

When an MPE query with respect to a network cannot be solved in the given memory a well-known approximation strategy known as *mini-buckets* [Dechter and Rish 2003], based on bucket elimination, can ignore certain dependencies amongst variables so that the new network becomes easier to solve and yet its solution can only “go wrong” in one direction, that its solution is always greater than or equal to that for the original network. Specifically, when processing a bucket of a variable X if it is not possible to compute the factor $\max_X(f(\mathbf{x})g(\mathbf{y}))$ (where X appears both in \mathbf{X} and \mathbf{Y}) because of insufficient available memory, then one can push maximization inside the multiplication, i.e. $\max_X f(\mathbf{x}) \cdot \max_X g(\mathbf{y})$. This gives the upper bound on the solution of $\max_X(f(\mathbf{x})g(\mathbf{y}))$, as for any two non-negative functions $\max_X(f(\mathbf{x})g(\mathbf{y})) \leq \max_X f(\mathbf{x}) \cdot \max_X g(\mathbf{y})$.

This approach is called mini-buckets as it virtually splits a bucket into two or more mini-buckets. In the above example the bucket of X gets split into two buckets containing factors $f(\mathbf{x})$ and $g(\mathbf{y})$, respectively. The factors obtained by performing maximization on each mini-bucket are placed in the next proceeding bucket in the same manner as in bucket elimination, and the next bucket is processed. As more mini-buckets are created the efficiency of bucket elimination as well as the quality of solution decreases.

Choi et al. [2007] formulate mini-buckets in a more general setting known as *node splitting*, of which mini-buckets is a special case, and use approximate solutions as upper bounds to prune a branch-and-bound search for an exact MPE. They also show

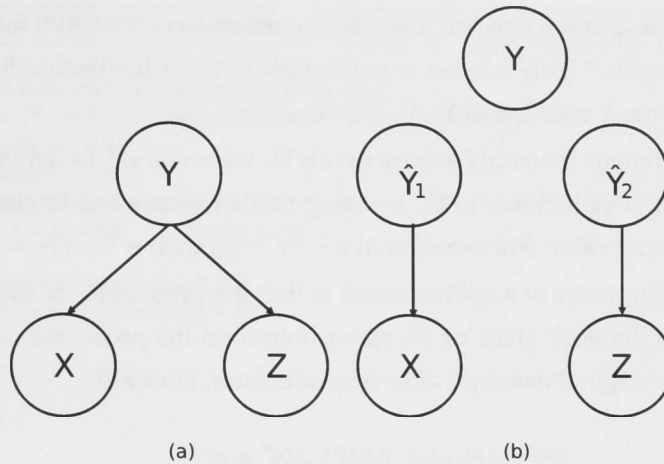


Figure 5.2: An example of node splitting.

that search may only be exponential in a much smaller set of *split variables* in the worst case. While mini-buckets focuses on generating better quality solutions, node splitting focuses on generating good network relaxations that reduce the number of split variables. Moreover, such an approximation can be used in conjunction with any exact inference algorithm—not just variable elimination—to solve MPE. In particular, they have shown that by replacing variable elimination with compilation, it is possible to improve search efficiency by orders of magnitude on hard networks.

In this following we review the search framework based on node splitting proposed in [Choi et al. 2007], which provides the basis for our new contributions.

5.4.2 Computing Bounds using Node Splitting

Node splitting creates a *clone* \hat{X} of a node X such that \hat{X} inherits some of the children of X . Formally:

Definition 5.4.1 (Node Splitting)

Let X be a node in a Bayesian network N with children Y . We say that X is **split according to children** $Z \subseteq Y$ when it results in a network N' that is obtained from N as follows:

- The edges outgoing from X to its children Z are removed.
- A new root node \hat{X} with a uniform prior is added to the network with nodes Z as its children.

A special case of splitting is when X is split according to every child, in which case X is said to be *fully split*. Figure 5.2 shows an example where Y has been split according to both of its children X and Z , and \hat{Y}_1, \hat{Y}_2 are clones of Y .

If \mathbf{e} is an assignment to variables in network N , we write \mathbf{e}^{\rightarrow} to denote the compatible assignment to their clones in N' , meaning that a variable and its clones (if any) are assigned the same value. For example, if $\mathbf{e} = \{Y = y\}$, then $\mathbf{e}^{\rightarrow} = \{\hat{Y}_1 = y, \hat{Y}_2 = y\}$.

An interesting property of a split network is that the probability of the MPE with respect to the split network gives us an upper bound on the probability of the MPE with respect to the original network, after normalization. Formally:

$$\text{MPE}_p(N, \mathbf{e}) \leq \beta \text{MPE}_p(N', \mathbf{e}, \mathbf{e}^{\rightarrow})$$

where β is a constant equal to the total number of instantiations of the clone variables. For example, suppose that in the network of Figure 5.2a, variables have binary domains and $\Pr(Y = y) = 0.6$, $\Pr(Y = \bar{y}) = 0.4$; all the parameters in the CPTs of X and Z are 0.5; and $\mathbf{e} = \{Y = \bar{y}\}$. Recall that both Y_1, Y_2 are given uniform priors. Then $\text{MPE}_p(N, \mathbf{e}) = 0.10$ and $\text{MPE}_p(N', \mathbf{e}, \mathbf{e}^{\rightarrow}) = 0.025$. The value of β is 4 in this case and we have $\text{MPE}_p(N, \mathbf{e}) \leq 4 * 0.025$.

In this example the upper bound equals the exact solution. In general, this is guaranteed if all the split variables have been instantiated in \mathbf{e} (and their clones in \mathbf{e}^{\rightarrow}). Hence to find an exact MPE one need only search in the space of instantiations of the split variables, as opposed to all network variables. At leaves of the search tree, solutions computed on the relaxed network give candidate MPE solutions, and elsewhere they give upper bounds to prune the search.

5.4.2.1 Branch-and-Bound Search

Algorithm 5.4.1 formalizes such a branch-and-bound search (this code only computes the MPE probability; however the actual MPE can be recovered with minor book-keeping). The procedure receives a split network N' and the evidence \mathbf{e} . At each call to `BNB-MPE`, the bound $\beta \text{MPE}(N', \mathbf{e}, \mathbf{e}^{\rightarrow})$ is computed (line 1). If the bound becomes less than or equal to the current solution (line 2), meaning that any further advancement in this part of the search tree is not going to give a better solution, the search backtracks. Otherwise, if the bound is greater than the current solution and the current assignment is complete over split variables, meaning that the bound has become exact, the current solution is replaced with the better one (lines 3–4). If the assignment

Algorithm 5.4.1 BNB-MPE : Computes probability of MPE for evidence \mathbf{e} **procedure** BNB-MPE (N', \mathbf{e})**inputs:** $\{N : \text{split network}\}, \{\mathbf{e} : \text{network instantiation}\}$ **global variables:** $\{Y : \text{split variables}\}, \{\beta : \text{number of possible assignments to clone variables}\}$

```

1: bound =  $\beta \text{MPE}(N', \mathbf{e}, \mathbf{e}^{-\rightarrow})$ 
2: if bound > solution then
3:   if  $\mathbf{e}$  is complete instantiation of variables  $Y$  then
4:     solution = bound /* bound is exact */
5:   else
6:     pick some  $X \in Y$  such that  $X \notin E$ 
7:     for each value  $x_i$  of variable  $X$  do
8:        $\mathbf{e} \leftarrow \mathbf{e} \cup \{X = x_i\}$ 
9:       BNB-MPE ( $N', \mathbf{e}$ )
10:       $\mathbf{e} \leftarrow \mathbf{e} \setminus \{X = x_i\}$ 
11:     end for
12:   end if
13: end if

```

is not complete then an unassigned split variable X is chosen (line 6), \mathbf{e} is appended with the assignment $X = x_i$ (line 7), and BNB-MPE is recursively called (line 8). After the recursive call returns the assignment $X = x_i$ is removed from \mathbf{e} and other values of X are tried in the same way.

5.4.2.2 The Choice of Variables to Split

The choice of which variables to split (variables Y in the pseudocode) has a fundamental impact on the efficiency of this approach. More split variables may make the relaxed network easier to solve, but may loosen the bounds and increase the search space. Choi et. al [2007] studied a strategy based on jointree construction [Dechter 2003]: A variable is chosen and fully split that can cause the highest reduction in the size of the jointree cliques and separators. Once a variable is fully split a jointree of the new network is constructed, and the process repeated until the treewidth of the network drops to a preset target. This heuristic was shown to outperform the mini-buckets heuristic in [Choi et al. 2007], and is therefore used in our present work. Also, given the success reported in [Choi et al. 2007] we will use compilation as the underlying method for computing MPEs of relaxed networks.

5.5 Variable and Value Ordering for MPE Search

An important aspect of perhaps any search algorithm is the quality of variable and value order which can have significant impact on the efficiency of the search. Choi et al. [2007] have used a neutral heuristic and left this aspect unaddressed. Given

the advancements already reported in their work based on combining splitting strategies with new inference methods, one cannot but wonder whether more sophisticated variable and value ordering might not take us even farther.

We have thus undertaken this investigation and now take delight in reporting it here. The first type of heuristic we examined is based on computing the entropies of variables, and the second on a form of learning from nogoods. We will discuss some of our findings, which have eventually led us to combine the two heuristics for use in a single algorithm.

5.5.1 Entropy-based Ordering

We start by revisiting the notion of Shannon's entropy ξ , as given in Section 3.2.2, which is defined with respect to a probability distribution of a discrete random variable X ranging over values x_1, x_2, \dots, x_k . Formally:

$$\xi(X) = - \sum_{i=1}^k \Pr(X = x_i) \log \Pr(X = x_i).$$

Entropy quantifies the amount of uncertainty over the value of the random variable. It is maximal when all probabilities $\Pr(X = x_i)$ are equal, and minimal when one of them is 1.

Hence as a first experiment we consider a heuristic that favors those instantiations of variables that are more likely to be MPEs. The idea is that finding an MPE earlier helps terminate the search earlier. To that end, the heuristic prefers those variables that provide more certainty about their values (i.e., have smaller entropies), and favor those values of the chosen variable that are more likely than others (i.e., have higher probabilities).

This heuristic can be used in either a static or dynamic setting. In a static setting, we order the split variables in increasing order of their entropies, and order the values of each variable in decreasing order of their probabilities, and keep the order fixed throughout the search. In a dynamic setting, we update the probabilities of the split variables at each search node and reorder the variables and values accordingly.

The heuristic requires computing the probabilities of the values of all split variables, which can be obtained conveniently as we have already assumed that the split network will be compiled for the purpose of computing its MPE—the compilation, in the form of arithmetic circuits, admits linear-time procedures for obtaining these probabilities. Since a variable and its clones can now get values independent of each

other, the computed probabilities for a variable may be inaccurate. Suppose that we have computed the probability of a value of a variable and the probabilities of the same value of its clones with respect to the split network. Out of these some will be closer to the probability with respect to the original network. Therefore, in an attempt to improve the accuracy of the probabilities, we take the average of the probabilities of a split variable and its clones (for the same value) when evaluating the heuristic.

While we will present detailed results in Section 5.6, we note here that our initial experiments clearly showed that a static entropy-based ordering immediately led to significantly better performance than the neutral heuristic. On the other hand, the dynamic version, although often effective in reducing the search tree, turned out to be generally too expensive, resulting in worse performance overall.

5.5.2 Nogood-based Ordering

Hence we need to look further if we desire an effective heuristic that remains inexpensive in a dynamic setting. Here the notion of learning from *nogoods* comes to rescue.

5.5.2.1 Nogoods

In the *constraint satisfaction* framework where it was originally studied, a nogood is a partial instantiation of variables that cannot be extended to a complete solution. In our case, we define a nogood to be a partial instantiation of the split variables (i.e., search variables) that results in pruning of the node (i.e., the upper bound being \leq the current best candidate solution).

Note that at the time of pruning, some of the assignments in the nogood \mathbf{g} may contribute more than others to the tightness of the bound (and hence the pruning of the node), and it may be possible to retract some of those less contributing assignments from \mathbf{g} without loosening the bound enough to prevent pruning. That is, an assignment $X = x$ can be removed from \mathbf{g} if after removing it from \mathbf{g} the upper bound remains \leq to the current best candidate solution. These refined nogoods can then be *learned* (i.e., stored) so that any future search branches containing them can be immediately pruned.

Nogood learning in this original form comes with the significant overhead of having to re-compute bounds to determine which assignments can be retracted, which is confirmed by experiments given in Section 5.6. However, it gives us an interesting motivation for an efficient variable and value ordering heuristic described below.

5.5.2.2 Ordering Heuristic

The idea is to favor those variables and values that can quickly make the current assignment a nogood, so that backtracking occurs early during the search. Hence we give scores to the values of variables proportionate to the amounts of reduction that they cause in the bound, and favor those variables and values that have higher scores.

Specifically, every value x_i of a variable X is associated with a score $S(X = x_i)$, which is initialized to 0. The score $S(X)$ of the variable X is the average of the scores of its values. Once a variable X is assigned a value x_i , the amount of reduction in the bound caused by it is added to the score of $X = x_i$. That is, the score $S(X = x_i)$ is updated to be $S(X = x_i) + (cb - nb)$, where cb is the bound before assigning the value x_i to X and nb is the bound after the assignment. The heuristic chooses a variable with the highest score and assigns values to it in decreasing order of the scores of those values.

During initial experiments we observed that over the course of the search the scores can become misleading, as past updates to the scores may have lost their relevance under the current search conditions. To counter this effect, we reinitialize the scores periodically, and have found empirically that a period of 2000 search nodes tends to work well.

Finally, we combine this heuristic with the entropy-based approach by using the entropy-based static order as the initial order of the variables and their values, so that the search may tend to start with a better candidate solution. We now proceed to present an empirical study which shows, among other things, that this combined heuristic gives significantly better performance both in terms of search space and time.

5.6 Experimental Results

In this empirical study we evaluate the different heuristics we have proposed, and in particular show that with the final combined heuristic we achieve significant advances in efficiency and scalability, able to solve problems that are intractable for other known methods.

We conducted our experiments on a cluster of computers consisting of two types of (comparable) CPUs, Intel Core Duo 2.4 GHz and AMD Athlon 64 X2 Dual Core Processor 4600+, both with 4 GB of RAM running Linux. A memory limit of 1 GB was imposed on each test case. Compilation of Bayesian networks was done using the C2D compiler [Darwiche 2004; Darwiche 2005]. We use a trivial seed of 0 as the initial

networks	cases	cases solved									common cases			
		NG-DVO						SC-DVO			NG-DVO		SC-DVO	
		solved	time	space	nogoods	pruned		solved	time	space	time	space	time	space
Ratio 50	90	77	89.5	2337	875	576	90	33.2	3365	89.5	2337	7.7	1218	
Ratio 75	150	108	74.8	3202	1019	1159	139	49.7	14875	74.8	3202	3.4	1648	
Ratio 90	210	135	42.3	6161	873	4413	160	26.2	13719	42.3	6161	1.6	1402	

Table 5.1: Comparing nogood learning with score-based DVO on grid networks.

networks	cases	cases solved									common cases					
		ENT-DVO			ENT-SVO			SC-DVO			ENT-DVO		ENT-SVO		SC-DVO	
		solved	time	space	solved	time	space	solved	time	space	time	space	time	space	time	space
Ratio 50	2250	1890	95.7	6286	2121	51.6	8262	2248	25.9	2616	94.1	6215	24.3	4619	7.3	1082
Ratio 75	3750	2693	60.3	5195	3015	40.4	15547	3624	38.0	10909	57.2	4965	15.3	6722	1.6	712
Ratio 90	5250	3463	23.4	2387	3599	22.2	12919	3995	8.2	4453	20.0	2102	10.0	6740	0.3	275

Table 5.2: Results on grid networks.

MPE solution to start the search. In general, we keep splitting the network variables until treewidth becomes ≤ 10 , unless otherwise stated.

We used a variety of networks: the grid networks introduced in [Sang et al. 2005], a set of randomly generated networks as in [Marinescu et al. 2003], networks for genetic linkage analysis, and a set of 42 networks for decoding error-correcting codes as generated in [Choi et al. 2007]. The last group were trivial to solve by all the three heuristics, so we do not report results for them.

For each heuristic, we report the number of cases solved, search time, and search space, where the search space are time are averages over solved cases. We also compare the performance of heuristics on those cases that were solved by all heuristics. As a reference point, we generated a random static variable and value order for the split variables in each split network instance and compared all the heuristics against it. The comparison showed that the new heuristics generally provide many orders of magnitude savings in search time and space over the random heuristic, and hence we will only include the new heuristics in the presented results. Finally, we will write SVO and DVO as shorthand for static and dynamic variable and value ordering, respectively.

5.6.1 Grid Networks

We first use these networks to show, in Table 5.1, that the score-based dynamic ordering (referred to as SC-DVO) outperforms nogood learning itself. For this purpose, we consider a similar dynamic ordering referred to as NG-DVO in the table. This heuristic starts with the same initial order, uses the same method of variable and value selection, and also reinitializes the scores after the specified period. However, the scores are updated as follows: When a nogood is learned, the score is incremented for each

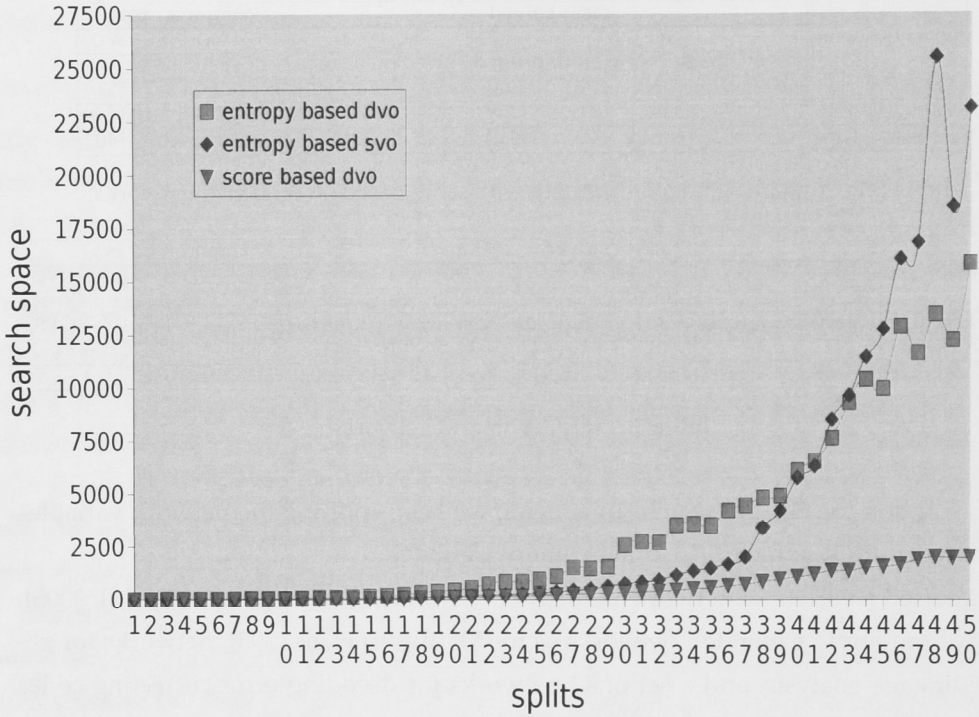


Figure 5.3: Comparing search spaces on grid networks.

literal in the nogood. We compare the performance of both techniques by computing the MPE for the empty evidence on each network. For NG-DVO we also report the average number of nogoods learned and the average number of branches pruned by nogoods. The results clearly show that score-based DVO is significantly faster than nogood learning and results in a much reduced search space.

We then show that the score-based DVO performs consistently better than other heuristics when the number of splits in a network is varied. For this purpose, we consider only 80 instances of these networks, in the range 90-20-1 to 90-30-9, which are some of the harder instances in this suite. The treewidths of these instances range from high twenties up to low forties. However, all of them can be compiled with a single split using the strategy mentioned above.

For each instance we split the network using 1 up to 50 splits, making a total of $50 * 80 = 4000$ cases to solve. For each case we compute the MPE for the empty evidence under a time limit of 15 minutes. The entropy-based DVO solved 3656 cases, entropy-based SVO solved 3968 cases, and score-based DVO solved 3980 cases, where the score-based DVO solved all those cases solved by the others. We plot a graph of search space against the number of splits, and a graph of search time against the

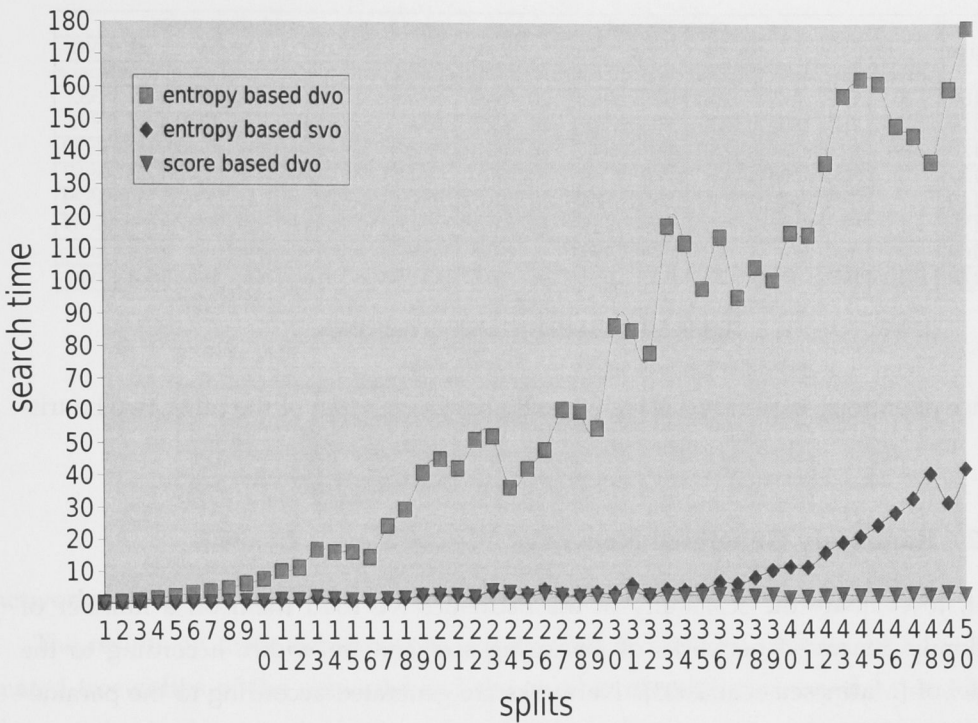


Figure 5.4: Comparing search times on grid networks.

number of splits. Each of data point on the graphs is the average of search spaces/time for those queries solved by all three heuristics.

In Figure 5.3, we observe that, interestingly, entropy-based SVO can often perform significantly better than entropy-based DVO, although its search space starts to grow faster than the DVO when the number of splits increases. This can be ascribed to the fact that DVO can exploit the changing probabilities of variables during search. The most significant result, however, is that for any number of splits the search space of score-based DVO is generally much smaller than that of the other two, and can become orders of magnitude smaller when the number of splits grows.

In Figure 5.4, we observe that entropy-based DVO gives poor performance, apparently because it has to do expensive probability updates at each search node. Again, we note that the score-based DVO is generally much faster than the other two heuristics, and can become orders of magnitude faster when the number of splits grows.

To further assess the performance of heuristics on a wide range of MPE queries, we considered the whole suite of grid networks and randomly generated 25 queries for each network. The results, summarized in Table 5.2, again confirm that the perfor-

size	cases solved									common cases					
	ENT-DVO			ENT-SVO			SC-DVO			ENT-DVO		ENT-SVO		SC-DVO	
	solved	time	space	solved	time	space	solved	time	space	time	space	time	space	time	space
100	20	131	867	20	47	696	20	31	532	131	867	47	696	31	532
110	20	187	1800	20	70	1270	20	67	952	187	1800	70	1270	67	952
120	19	923	9374	20	471	7760	20	210	3061	923	9374	434	7640	188	2950
130	12	1100	10849	15	1100	19755	18	561	7462	1100	10849	729	11168	291	3989
140	7	1715	19621	11	1340	28574	15	1090	10545	1680	20016	741	17965	311	4529
150	1	2505	10800	6	1859	41628	15	1619	25293	2505	10800	809	8727	634	6789
160	0	n.a.	n.a.	1	2379	22428	6	2257	36884	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
170	2	0.1	0	2	0.1	0	3	637	14319	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.

Table 5.3: Results on random networks.

mance of entropy-based DVO is significantly better than that of the other two heuristics.

5.6.2 Randomly Generated Networks

Next, to evaluate the scalability of the heuristics we tried them on a number of randomly generated networks of increasing size and treewidth, according to the model of [Marinescu et al. 2003]. Networks are generated according to the parameters (N, K, C, P) , where N is the number of variables, K is their domain size, C is the number of CPTs, and P is the number of parents in each CPT. The network structure is created by randomly picking C variables out of N and, for each, randomly selecting P parents from the preceding variables, relative to some ordering. Each CPT is generated uniformly at random. We used $C = 90\%N$, $P = 2$, $K = 3$, and N ranging from 100 to 200 at increments of 10. The treewidths of these networks range from 15 to 33, and generally increases with N . For each network size we generated 20 network instances, and for each instance generated a random MPE query, making a total of 20 queries per network size. The time limit to solve each query was set to 1 hour.

The results of these experiments are summarized in Table 5.3. The score-based DVO solved all those cases that the other two could solve, plus more, and scales much better when the network size increases. On cases solved by all three, the performance of score-based DVO is also the best in terms of both search time and space.

5.6.3 Networks for Genetic Linkage Analysis

We now consider even harder cases from genetic linkage analysis (<http://bioinfo.cs.technion.a.c.il/superlink/>). We extracted 8 networks that have high treewidths and cannot be compiled without splitting, and considered 20 randomly generated MPE queries on each of them, for a total of 160 queries. Since these

network	treewidth	splits	SC-DVO		
			solved	time	space
pedigree13	43	83	20	1728	168251
pedigree19	36	59	12	6868	532717
pedigree31	44	84	16	6672	446583
pedigree34	35	79	12	5109	332899
pedigree40	37	72	2	6575	745088
pedigree41	43	84	9	7517	783737
pedigree44	33	54	5	7041	673165
pedigree51	54	75	16	3451	268774

Table 5.4: Results on networks for genetic linkage analysis.

networks	cases	cases solved				time on common cases	
		SamIam		SC-DVO		SamIam	SC-DVO
		solved	time	solved	time		
Ratio 50	90	49	23.8	90	31.1	23.8	1.4
Ratio 75	150	45	14.1	148	185.4	14.1	0.3
Ratio 90	210	48	18.9	169	114.5	18.9	0.1

Table 5.5: Comparison with SamIam on grid networks.

networks are significantly harder, the time limit on each query was set to 4 hours.

The results on these networks are summarized in Table 5.4. We report the estimated treewidths of the networks and the number of splits performed on each of them, and only show results for score-based DVO, as the two entropy-based heuristics could only solve 9 trivial cases when the MPE for the given evidence was already 0 and the search was not performed at all. The score-based DVO, by contrast, solved most of the cases, which further establishes its better scalability.

5.6.4 Comparison with Other Tools

First we compare our MPE solver SC-DVO with an independent Bayesian network inference engine known as SamIam (<http://reasoning.cs.ucla.edu/samiam/>). As we were unable to run SamIam on our cluster, these experiments were conducted on a machine with a 3.2 GHz Intel Pentium IV processor and 2 GB of RAM running Linux. For each network we compute the MPE for the empty evidence under a time limit of 1 hour.

The random and genetic linkage analysis networks proved too hard to be solved by SamIam. For the grid networks, the results of the comparison are summarized in Table 5.5. We observe that SC-DVO solved roughly from 2 to 3 times more instances than SamIam, and on cases solved by both SC-DVO can also be orders of magnitude faster.

We then compare SC-DVO with the best-first search algorithm (SMBBF) of [Marinescu and Dechter 2007]. This tool is only available for Windows, and we used

networks	cases	cases solved				time on common cases	
		SMBBF		SC-DVO		SMBBF	SC-DVO
		solved	time	solved	time		
Ratio 50	9	7	5.39	9	14.33	5.39	4.34
Ratio 75	15	10	16.16	15	199.11	16.16	2.1
Ratio 90	15	10	7.32	15	3.87	7.32	0.429

Table 5.6: Comparison with SMBBF on grid networks.

a machine with an Intel Core 2 Duo 2.33 GHz and 1 GB of memory running Windows. Note that our solver SC-DVO was run on a cluster with comparable CPU speeds. However, since Windows reserves some of the memory for the operating system, we accordingly reduced the memory limit for SC-DVO from 1 GB to 768 MB for these experiments. Again a time limit of 1 hour was imposed for each query, computing the MPE for the empty evidence .

SMBBF requires a parameter i that bounds the mini-bucket size. The results of [Marinescu and Dechter 2007] do not appear to suggest any preferred value for i ; hence we somewhat arbitrarily set $i = 20$ taking into account the relatively high treewidth of our benchmarks.

Due to limited computational resources we used a sample of the grid networks under each ratio category such that each selected network represented a class of networks comparable in difficulty. The comparison of is shown in Table 5.6. In contrast to SC-DVO, SMBBF failed to solve a significant number of the networks; it was also slower on the networks that were solved by both.

Finally, from the random networks we arbitrarily took 4 networks of size 100, 110, 120, and 130, respectively, and from the networks for genetic linkage analysis we took pedigree13, the apparently easiest one. None of these instances could be solved by SMBBF, which quickly ran out of memory and got terminated by the operating system. By contrast, SC-DVO solved all of them.

5.7 Conclusions

We have presented a novel and efficient heuristic for dynamically ordering variables and their values in a branch-and-bound search for MPE. A comprehensive empirical evaluation indicates that significant improvements in search time and space are achieved over less sophisticated heuristics and over an existing Bayesian network inference engine. On the whole, we have extended the reach of exact MPE algorithms to networks that are now solved for the first time, particularly the very high treewidth pedigree networks mentioned in table 5.4.

Related Work

In this Chapter, we differentiate our techniques from related work of two types: structural methods and probabilistic methods.

6.1 Structural Methods

These techniques include [Smith et al. 2004] that exploits structural dominators of the circuit to make the diagnosis more efficient, and [Chittaro and Ranon 2004; Feldman and van Gemund 2006] that use structure-based hierarchical methods.

6.1.1 Diagnosis using Boolean Satisfiability

Smith, Veneris and Viglas [2004] use satisfiability (SAT) solvers as the reasoning system to find faults in ISCAS-85 and ISCAS-89 circuits. They are also able to exploit structural dominators of the circuit to make the diagnosis more efficient, which is similar to our approach of Chapter 2. Below we describe their approach, and then outline the similarities and differences with our work.

The health of a gate G is modeled by a multiplexer M , injected ahead of each gate. Specifically, the output of G is connected to an input of M , whereas the other input of M is not connected, which is considered to be the dummy output of G used when the gate must be faulty, and can take arbitrary value. The output of M is considered to be the output of G , and is connected to all those points where G was originally connected. The selection line S of M selects between the original (healthy) or the dummy (faulty) output of G . The gate whose multiplexer selects the faulty line is considered to be broken. The resulting circuit is encoded into CNF and a SAT solver is used to find a satisfying assignment to all the variables, which when projected over the selection lines of the multiplexers gives a diagnosis. To find multiple diagnoses, an extra clause that prevents the same diagnosis being found again is added to the database of the

SAT solver and the solver is then forced to backtrack. The cardinality of the diagnoses is enforced, before the diagnosis starts, through an additional adder circuit involving the selection lines with a comparator to compare against the desired cardinality, such that as soon as the cardinality of the diagnosis increases the desired cardinality the solver backtracks. To improve the performance, the task is divided into two phases. In the first phase, multi-plexers are injected only at the structural dominators of circuit. In the second pass, the faults are found in their respective fan-in cones.

With respect to exploiting the circuit structure the technique has similarities to our approach. However, the authors report results on only single and double stuck-at faults, which are generally easier to handle. Our approach does not depend on any prior assumptions about the cardinality of fault and can handle faults of arbitrary cardinality.

6.1.2 Hierarchical Methods

Chittaro and Ranon [2004] propose a new formalism for hierarchical diagnosis based on structural abstraction. The hierarchical decomposition of the system is created by collecting subsets of components into single units, called aggregates. An aggregate can be further decomposed into smaller aggregates representing different levels of abstractions. First, a set of diagnoses at the most abstract level are computed, which are then refined hierarchically to the most detailed level.

Similarly, Feldman and van Gemund [2006] develop a two-step hierarchical diagnosis algorithm and test it on reverse engineered ISCAS-85 circuits [Hansen et al. 1999] that are available in high-level form. The idea is to decompose the system into hierarchies in such a way as to minimize the sharing of variables between the aggregates. This can be done for well engineered problems and they have formed hierarchies by hand for ISCAS-85 circuits. The hierarchy is represented by a graph where each node in the graph represents an aggregate. Two aggregates are connected by an edge if their components share variables. The depth of a hierarchy is equal to the number of nodes in the longest path from the root of the hierarchy to a leaf node. For example, a circuit with four cascaded buffer gates ($A \leftrightarrow B \leftrightarrow C \leftrightarrow D$) could be represented by a three node hierarchy having depth 3, containing components $N1 : \{A\}$, $N2 : \{B\}$, $N3 : \{C, D\}$, such that the node $N1$ is the root and is connected to $N2$, whereas $N2$ is connected to $N3$.

When every aggregate is encoded into CNF, the system gets represented by a hierarchical logical formula. In the above example, if we encode every gate without a

health variable then the three aggregates will be encoded as: $N1 : \{(\neg A \vee B) \wedge (A \vee \neg B)\}$, $N2 : \{(\neg B \vee C) \wedge (B \vee \neg C)\}$, $N3 : \{(\neg C \vee D) \wedge (C \vee \neg D)\}$. This representation can be translated to a hierarchical DNF (Disjunctive Normal Form) of an adjustable depth: to a fully hierarchical DNF, a fully flattened DNF, or a partially flattened DNF dictated by a depth parameter. Specifically, each node is compiled into a DNF and then the depth of the hierarchy can be reduced (if required) by multiplying (conjoining) nodes and collapsing them to single nodes. In the above example when every node is compiled to a DNF, we get: $N1 : \{(A \wedge B) \vee (\neg A \wedge \neg B)\}$, $N2 : \{(B \wedge C) \vee (\neg B \wedge \neg C)\}$, $N3 : \{(C \wedge D) \vee (\neg C \wedge \neg D)\}$. If the depth needs to be adjusted to 2 one can multiply $N2$ and $N3$ to get a single node: $N2 \times N3 : \{(\neg B \vee C) \wedge (B \vee \neg C) \wedge (\neg C \vee D) \wedge (C \vee \neg D)\}$.

Thus a fully flattened representation contains a single node only where all the nodes get multiplied together, and a fully hierarchical representation contains a single component at every node. Once a hierarchical DNF has been obtained, a hierarchical best-first search algorithm is employed to find diagnoses. Since a fully flattened representation can be of exponential size, and the search in hierarchical representation may take time exponential in the depth of the hierarchy, the idea is to adjust the depth so that the formula may be compiled in hierarchical form within the given memory limits and search performed in the given time limit.

The hierarchical aspect of these two approaches is similar to that of ours; however, they require a hierarchical decomposition of the system to be either given as part of the input, or obtained by hand, while our approach searches for hierarchies automatically. Another major difference is that they consider only the computation of diagnoses and do not address the problem of sequential diagnosis.

6.2 Probabilistic Methods

These techniques include [Varshney et al. 1982] that combine information theory and heuristic search to generate optimal policies, [de Kleer 1992; de Kleer 2006] that are related to GDE, methods that use a Bayesian network model of the system e.g. [Heckerman et al. 1995; Flesch et al. 2007] as well as techniques to compute MPE.

6.2.1 Optimal Policies using Heuristic Search and Entropy

Varshney et al. [1982] have studied the application of heuristic search and information theory on sequential diagnosis. They have used the heuristic search in AND/OR graphs (e.g. AO* and CF algorithms) to generate optimal policies. The heuristic eval-

uation functions (HEF) used at each search node to guide the search are constructed using the entropy of diagnoses, which ensure that the policy generated by the algorithm is always an optimal policy. On the bright side the technique can significantly reduce the search space of optimal policies by the use of better HEFs. However an HEF based on computation of entropy of diagnoses seems feasible only when the number of diagnoses is small.

6.2.2 Focusing on Probable Diagnoses

To address the problem of computation explosion of GDE, de Kleer [1992] considers focusing the processing of the GDE and its underlying reasoning engine to a smaller set of most probable diagnoses. Hence the reasoning engine responsible for generating the conflicts and a set of diagnoses only considers the probable conflicts and diagnoses, significantly reducing the computational cost. The technique allows the diagnoses to be generated and measurement points computed for the largest circuits in ISCAS-85 benchmark suite. The author does not report any results regarding the diagnostic cost on these circuits. However, he does mention that the quality of the measurements were compromised as the computed probabilities may be inaccurate. By contrast, in our approach the probabilities can be computed exactly. Cloning does affect the probabilities of variables but our experiments show that it often does not affect the diagnostic cost in a significantly adverse manner.

6.2.3 Improving Probability Estimates to Lower Diagnostic Costs

Based on the GDE framework, de Kleer [2006] studies the sensitivity of diagnostic cost to what is called the ϵ -policy (introduced in Section 3.2.2), which is the policy that quantifies how the posterior probabilities of diagnoses are to be estimated when GDE computes its heuristic.

In our case, probabilities of diagnoses are not required at all, and the other probabilities that are required can all be computed exactly by evaluating and differentiating the d-DNNF. Nevertheless, our algorithm can be sensitive to the initial probabilistic model given and sensitivity analysis in this regard may lead to interesting findings.

6.2.4 Decision-theoretic Troubleshooting

Heckerman, Breese and Rommelse [1995] propose a Bayesian network based sequential diagnosis system, which avoids the exponential space problem of generating "op-

timal sequences using decision trees” and attempts to approximate them using fault probabilities that are computed from the Bayesian network model of the system. The idea is to test a component for failure that has the highest probability of failure. If the component is healthy, the probabilities are updated and the next component is tested. If the component is actually faulty and replacing it brings the faulty system to normal state, the process stops. Otherwise more tests are made to find other faults. A prior assumption of a single fault is used; however the system is able to find multiple faults by reiterating after a single fault has been found such that during the next iteration again a single fault is assumed.

Our idea of testing the most likely failing component comes from this work. However, in this technique the testing of a component is considered a unit operation and components are tested in decreasing order of their likelihood of failure, which is computed assuming a single fault (this assumption could compromise the quality of the measurement sequence in multiple-fault cases as the authors pointed out). In our case, by contrast, the testing of each variable of a component is a unit operation, calling for a more complex heuristic in order to minimize the number of tests; also, we do not need to assume a single fault. Our work also goes further in scalability using a series of structure-based techniques: compilation, abstraction, and component cloning.

6.2.5 Adding Uncertainty to Model-based Diagnosis

Recently, Flesch, Lucas and Weide [2007] proposed a new framework to integrate probabilistic reasoning into model-based diagnosis. The framework is based upon the notion of *conflict measure*, which originated as a tool for the detection of conflicts between an observation and a given Bayesian network [Jensen 2001]. When a system is modeled as a Bayesian network for diagnostic reasoning, it is possible to use this conflict measure to differentiate between diagnoses according to their degree of consistency with a given set of observations. This work, however, does not address the problem of sequential diagnosis, i.e., locating actual faults by taking measurements.

6.2.6 Computing Most Probable Explanations

Popular exact techniques are bucket elimination (based on variable elimination), DNNF compilation, systematic search and reduction to *weighted MAX-SAT*.

The complexity of Bucket elimination [Dechter 1996] is exponential in the treewidth of the elimination order. Compilation has similar complexity; however, it is known to exploit *local structure* so that its complexity may be further

reduced [Chavira and Darwiche 2005], and is exponential in the treewidth in the worst case only. Both variable elimination and compilation can run out of memory when the treewidth is large. Search methods can often avoid this memory explosion. These include branch-and-bound search (used in our work) and best-first search [Shimony and Charniak 1991]. A more recent implementation of best-first search [Marinescu and Dechter 2007] is empirically analyzed in Section 5.6 in comparison with our technique.

Another technique reduces the problem of computing MPE to the well known problem of Weighted MAX-SAT (W-MAX-SAT) [Park 2002; Pipatsrisawat and Darwiche 2007]. A weighted CNF is a CNF in which each clause is associated with a number called its weight. W-MAX-SAT finds an instantiation of the variables in the CNF such that the total sum of the weights of the satisfied clauses is maximal, which can be done using a systematic search method [Pipatsrisawat and Darwiche 2007].

Conclusions and Future Work

We have demonstrated that it is possible to exploit the structure of a system to scale diagnosis to those large systems which cannot otherwise be diagnosed. Specifically, we have used a DNNF-based compilation approach that can exploit system structure to compactly represent the functionality of the system and also allows common diagnostic tasks to be performed efficiently.

For a large system that cannot be compiled, we can obtain a structural abstraction of it that can significantly reduce the number of components to be diagnosed, allowing larger systems to be compiled and diagnosed in a hierarchical fashion. We are able to compute a set of minimum-cardinality diagnoses of an abnormal system more efficiently using abstraction.

The compilation also allows us to efficiently compute a novel heuristic to propose measurement points, which enables us to locate actual faults in the system with low cost. The heuristic that is based upon failure probabilities of component failures and the entropies of system variables is scalable and can also be combined with abstraction to diagnose larger systems.

For systems whose abstraction is still too large to compile, we employ a technique of component cloning that modifies the structure of the system so that the size of its abstraction reduces significantly, allowing it to be compiled and diagnosed. We show that cloning allows measurement points to be computed for very large systems without affecting the diagnostic cost in an adverse manner.

Finally, we apply the entropy-based approach to computing most probable explanations in a Bayesian network using a branch-and-bound search algorithm. We propose a heuristic for dynamically ordering search variables and their values that significantly reduces the search space and time, is easy to compute, and enables us to solve many networks for the first time.

- Topics for future work include sensitivity of diagnostic cost to probabilities in the

system model (which may be estimated from data), diagnosis with measurements of varying costs, application of hierarchical approach to MPE search, and application of techniques in this thesis to other probabilistic queries such as MAP (maximum a posteriori hypothesis).

Bibliography

- CHAVIRA, M. AND DARWICHE, A. 2005. Compiling Bayesian networks with local structure. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)* (2005), pp. 1306–1312. (pp. 6, 77, 96)
- CHAVIRA, M. AND DARWICHE, A. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172, 6–7 (April), 772–799. (p. 40)
- CHITTARO, L. AND RANON, R. 2004. Hierarchical model-based diagnosis based on structural abstraction. *Artificial Intelligence* 155, 1-2, 147–182. (pp. 91, 92)
- CHOI, A., CHAVIRA, M., AND DARWICHE, A. 2007. Node splitting: A scheme for generating upper bounds in Bayesian networks. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI)* (2007), pp. 57–66. (pp. 6, 7, 8, 65, 74, 78, 79, 81, 85)
- DARWICHE, A. 1998. Model-based diagnosis using structured system descriptions. *Artificial Intelligence Research* 8, 165–222. (p. 19)
- DARWICHE, A. 2001. Decomposable negation normal form. *Journal of the ACM* 48, 4, 608–647. (pp. 4, 7, 13, 14, 15, 16, 53)
- DARWICHE, A. 2002. A logical approach to factoring belief networks. In *In Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR)* (2002), pp. 409–420. (p. 38)
- DARWICHE, A. 2003. A differential approach to inference in Bayesian networks. *Journal of the ACM* 50, 3, 280–305. (pp. 7, 38, 48, 50, 51)
- DARWICHE, A. 2004. New advances in compiling CNF into decomposable negation normal form. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)* (2004), pp. 328–332. (pp. 16, 84)
- DARWICHE, A. 2005. The C2D compiler user manual. Technical Report D-147, Computer Science Department, UCLA. <http://reasoning.cs.ucla.edu/c2d/>. (p. 84)
- DARWICHE, A. AND HOPKINS, M. 2001. Using recursive decomposition to con-

-
- struct elimination orders, jointrees and dtrees. In *Trends in Artificial Intelligence, Lecture notes in AI* (2001), pp. 180–191. Springer-Verlag. (p. 16)
- DARWICHE, A. AND MARQUIS, P. 2002. A knowledge compilation map. *Artificial Intelligence Research* 17, 229–264. (pp. 3, 4, 7)
- DE KLEER, J. 1976. Local methods for localizing faults in electronic circuits. *MI AI Memo 394, Cambridge, MA.* (p. 11)
- DE KLEER, J. 1992. Focusing on probable diagnosis. In *Readings in model-based diagnosis*, pp. 131–137. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (pp. 48, 93, 94)
- DE KLEER, J. 2006. Improving probability estimates to lower diagnostic costs. In *17th International Workshop on Principles of Diagnosis (DX)* (2006). (pp. 5, 47, 93, 94)
- DE KLEER, J., MACKWORTH, A. K., AND REITER, R. 1992. Characterizing diagnoses and systems. In *Readings in model-based diagnosis*, pp. 54–65. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (pp. 3, 11)
- DE KLEER, J., RAIMAN, O., AND SHIRLEY, M. 1992. One step lookahead is pretty good. In *Readings in model-based diagnosis*, pp. 138–142. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (pp. 5, 47, 56)
- DE KLEER, J. AND WILLIAMS, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32, 1, 97–130. (pp. 5, 44, 45, 47, 49, 56)
- DECHTER, R. 1996. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)* (1996), pp. 211–219. (pp. 77, 95)
- DECHTER, R. 2003. *Constraint Processing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (p. 81)
- DECHTER, R. AND RISH, I. 2003. Mini-buckets: A general scheme for bounded inference. *Journal of the ACM* 50, 2, 107–153. (p. 78)
- FELDMAN, A. AND VAN GEMUND, A. 2006. A two-step hierarchical algorithm for model-based diagnosis. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI)* (2006). (pp. 91, 92)
- FLESCH, I., LUCAS, P., AND VAN DER WEIDE, T. 2007. Conflict-based diagnosis: Adding uncertainty to model-based diagnosis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)* (2007), pp. 380–385. (pp. 93, 95)

-
- FORBUS, K. D. AND DE KLEER, J. 1993. *Building problem solvers*. MIT Press, Cambridge, MA, USA. (pp. 4, 47, 56)
- HANSEN, M. C., YALCIN, H., AND HAYES, J. P. 1999. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design and Test of Computers* 16, 3, 72–80. (p. 92)
- HECKERMAN, D., BREESE, J. S., AND ROMMELSE, K. 1995. Decision-theoretic troubleshooting. *Communications of the ACM* 38, 3, 49–57. (pp. 5, 7, 51, 53, 93, 94)
- HUANG, J. AND DARWICHE, A. 2005. On compiling system models for faster and more scalable diagnosis. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI) (2005)*, pp. 300–306. (pp. 2, 14, 22, 32)
- JENSEN, F. V. 2001. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. (p. 95)
- KIRKLAND, T. AND MERCER, M. R. 1987. A topological search algorithm for ATPG. In *Proceedings of the 24th Conference on Design Automation (DAC) (1987)*, pp. 502–508. (pp. 21, 22)
- LIN, L. AND JIANG, Y. 2003. The computation of hitting sets: review and new algorithms. *Information Processing Letters* 86, 4, 177–184. (pp. 4, 12)
- LU, F., WANG, L.-C., CHENG, K.-T., AND HUANG, R. C.-Y. 2003. A circuit SAT solver with signal correlation guided learning. In *Design, Automation and Test in Europe (DATE) (2003)*, pp. 10892–10897. (p. 20)
- LU, F., WANG, L.-C., CHENG, K.-T. T., MOONDANOS, J., AND HANNA, Z. 2003. A signal correlation guided ATPG solver and its applications for solving difficult industrial cases. In *Proceedings of the 40th Conference on Design automation (DAC) (2003)*, pp. 436–441. (p. 20)
- MARINESCU, R. AND DECHTER, R. 2007. Best-first AND/OR search for most probable explanations. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence (UAI) (2007)*. (pp. 89, 90, 96)
- MARINESCU, R., KASK, K., AND DECHTER, R. 2003. Systematic vs. non-systematic algorithms for solving the MPE task. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI) (2003)*. (pp. 85, 88)
- PARK, J. D. 2002. Using weighted MAX-SAT engines to solve MPE. In *Eighteenth national conference on Artificial intelligence (Menlo Park, CA, USA, 2002)*, pp. 682–687. American Association for Artificial Intelligence. (p. 96)

- PEARL, J. 1979. Entropy, information and rational decisions. *Policy Analysis and Information Systems, Special Issue on Mathematical Foundations* 3, 1, 93–109. (p. 5)
- PEARL, J. 1988. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. (pp. 5, 37)
- PIPATSRISAWAT, K. AND DARWICHE, A. 2007. Clone: Solving weighted Max-SAT in a reduced search space. In *Proceedings of the 20th Australian Conference on Artificial Intelligence (AI)* (2007), pp. 223–233. (pp. 7, 65, 96)
- REITER, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32, 1, 57–95. (pp. 3, 9)
- SANG, T., BEAME, P., AND KAUTZ, H. 2005. Solving Bayesian networks by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, Volume 1 (2005), pp. 475–482. AAAI Press. (p. 85)
- SHIMONY, S. E. 1994. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence* 68, 2, 399–410. (pp. 6, 73, 74)
- SHIMONY, S. E. AND CHARNIAK, E. 1991. A new algorithm for finding MAP assignments to belief networks. In *Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence (UAI)* (New York, NY, USA, 1991), pp. 185–196. Elsevier Science Inc. (p. 96)
- SIDDIQI, S. AND HUANG, J. 2007. Hierarchical diagnosis of multiple faults. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)* (2007), pp. 581–586. (p. 9)
- SIDDIQI, S. AND HUANG, J. 2008. Probabilistic sequential diagnosis by compilation. In *Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics (ISAIM)* (2008). (pp. 35, 59)
- SIDDIQI, S. AND HUANG, J. 2009. Variable and value ordering for MPE search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)* (2009), pp. 1964–1969. (p. 73)
- SMITH, A., VENERIS, A., AND VIGLAS, A. 2004. Design diagnosis using Boolean satisfiability. In *Proceedings of the Ninth Asia and South Pacific Design Automation Conference (ASP-DAC)* (2004), pp. 218–223. (p. 91)
- STRUSS, P. AND DRESSLER, O. 1989. “Physical Negation” Integrating fault models into the general diagnostic engine. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)* (1989), pp. 1318–1323. (pp. 3, 11)

-
- VARSHNEY, P. K., HARTMANN, C. R. P., AND DE FARIA, J., J. M. 1982. Application of information theory to sequential fault diagnosis. *IEEE Trans. Comput.* 31, 2, 164–170. (p.93)