MODELING TECHNIQUES AND ALGORITHMS FOR PROBABILISTIC MODEL-BASED DIAGNOSIS AND REPAIR

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Sampath Srinivas
July 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Richard Fikes
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Eric Horvitz
(Microsoft Research)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Ross Shachter

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Yoav Shoham

Approved for the University Committee on Graduate Studies:

Abstract

Model-based diagnosis centers on the use of a behavioral model of a system to infer diagnoses of anomalous behavior. For model-based diagnosis techniques to become practical, some serious problems in the modeling of uncertainty and in the tractability of uncertainty management have to be addressed. These questions include: How can we tractably generate diagnoses in large systems? Where do the prior probabilities of component failure come from when modeling a system? How do we tractably compute low-cost repair strategies? How can we do diagnosis even if only partial descriptions of device operation are available? This dissertation seeks to bring model-based diagnosis closer to being a viable technology by addressing these problems.

We address the tractability of model-based diagnosis in two complementary ways. We show how the structure of a hierarchical system specification can be exploited to yield an efficient diagnosis algorithm. We also develop polynomial-time algorithms to generate candidate diagnoses in decreasing order of likelihood both when component failures are independent and when they are dependent. These candidates are then tested for consistency with observations. In effect, this gives us a generate-and-test scheme for enumerating diagnoses in decreasing order of plausibility. To help with modeling, we relate the prior probability of failure of a component to a temporal model of the failure process of the component. This allows the prior probability to be computed from an empirical reliability measure, the Mean Time Between Failure, and the uptime of the component. In addition, this work gives us a principled way of modeling possible state changes in the system when performing diagnosis with multiple observations where each observation occurs at a different time. Turning to the repair problem, we develop a general-purpose algorithm for computing optimal repair plans. The algorithm makes no assumptions about the structure of the system model and hence has to exhaustively search through all possible plans. We go on to demonstrate an interesting and useful restriction on the system model which allows the optimal repair plan to be computed in polynomial time. To perform diagnosis when only partial device descriptions are available, we develop a way of "completing" the model such that repair cost estimates computed from the completed model are conservative upper bounds of the actual repair cost. We conclude with a discussion about promising extensions of this work.

Acknowledgements

I would like to thank:

My advisor, Richard Fikes, for being an ideal mentor. He gave me the freedom to explore a variety of topics as I began graduate school. Later, as my interest became clearer, his gentle yet firm guidance allowed me to both clearly define my thesis and maintain progress. Always generous with his time and *always* good humored, Richard has also been a close friend to whom I often turned for advice.

Eric Horvitz, for his guidance and help through graduate school. He played a crucial role in helping me define my dissertation and also in keeping it moving. His talent for instantly understanding a problem's essence and what's more, suggesting innovative ways of looking at it have been and continue to be an inspiration. As a friend, his help and advice have been an invaluable aid in crucial decisions affecting my life.

Ross Shachter, for his insight and feedback during our powwows about my work. I re-learned my Bayesian Network and Decision theory fundamentals in an intensely enjoyable quarter assisting him with his class. My sessions on Ross's "therapy chair" helped me define more clearly what I wanted to do next in my career.

Yoav Shoham, for defining clearly what he expected in a dissertation and for monitoring these requirements in high bandwidth meetings. My term paper in his class marked a crucial stage in the progress from a tentative idea to a clear thesis topic.

My friends at KSL: Edward Feigenbaum, Yumi Iwasaki, Grace Smith, Alon Levy, Robert Engelmore, Jim Rice, Tom Gruber, Greg Olsen and Adam Farquhar for their feedback on my work and their help on many varied occasions. A very special thank you to Ramanathan Guha, whose has always been there with his friendship and advice since our college days. I owe an irredeemable debt to Pandu Nayak—he has been simultaneously a role model, colleague and friend as I threaded my way through graduate school and life. Without his help, my Ph.D. simply would not have been possible.

Johan deKleer, Peter Struss, David Heckerman, Thomas Dean and Brian Williams for very useful feedback at various stages of my thesis work.

Rockwell Palo Alto Laboratory, for making it possible *and fun* to wear a student hat while I still worked there. I owe the company and the whole RPAL gang a very very deep debt. I would like to say an especial thank you to Jim Martin, for being such a supportive boss and to Corinne Ruokangas, Cathy Froio, Max Henrion, Mark Peot, Adnan Darwiche, Dave Smith, Phil Stubblefield, Dan Larner, Tom Chavez, Nir Friedman and Mike Buckley for being such good friends and colleagues. I owe a special debt to my pal and colleague Moisés Goldszmidt for his humor, help and advice. I owe an immense debt to Jack Breese for being the ideal boss on my first job and a wonderful friend. I would also like to say a very very special thank you to Ken Fertig, who is simply one of the nicest and smartest people I know. He has influenced my work deeply.

My mentors from the early days: Alice Agogino, Stuart Russell and Michael Fehling, for helping me get started in research and my career.

My friends and family in the Bay Area: Arun, Prabha, Rama, Aarthi, Prasad, Shyam, Sanju, Kalpana, Kishore, Krishna, Krishna Rao and Prem for making life fun.

My family: Amma, Appa, Sangeetha and Pumsie for the constant love and encouragement that I have been brought up with and take for granted.

And finally, my wife and other half, Prithi, whose love, friendship and unfailing good humor gives my life its meaning. This dissertation is as much hers as it is mine.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the most salient aspects of modern life is the pervasiveness of complex human-engineered systems. Malfunctions of the systems we depend on are commonplace while the problem of maintaining these systems becomes ever more complicated. As a result, it has become increasingly important to provide intelligent support to the tasks of diagnosis of system malfunctions and repair planning. This dissertation addresses key problems encountered in constructing computer-based tools that assist with diagnosis and repair.

1.1 Automated diagnosis: Early approaches

Intelligent support for diagnosis has been an active area of research in Artificial Intelligence since its early years. Early approaches to the problem concentrated on problems encountered in medical diagnosis. During the 1960s, several efforts were made to develop automated programs to perform medical diagnosis [Warner *et al.*, 1961; Gorry and Barnett, 1968; de Dombal *et al.*, 1972]. The approach was probabilistic—posterior probabilities of diseases were computed based on observations of symptoms. A very simple causal model was employed—the various possible diseases were assumed to be mutually exclusive, i.e., exactly one disease could occur. Given the existence of a disease it was assumed that the occurrence of any one symptom was conditionally independent of the existence of any other symptom. These early systems had surprisingly good performance, sometimes outperforming human experts. However, a variety of other reasons such as poor interfaces and distrust of the very simple model led to their not being accepted.

The 1970s saw the creation of rule-based expert systems [Buchanan and Shortliffe, 1984]. A diagnostic rule-based expert system consists of a set of rules elicited from an expert who is versed with the particular system or domain which is being modeled. The rules can be used to encode either causal knowledge or diagnostic knowledge.

Causal rules encode observations of system behavior that result from the occurrence of underlying faults. Diagnostic rules encode what causes are possible given observations of system behavior.

As the applications of rule-based expert systems proliferated in the late seventies, some serious problems came to light. First, the systems are hard to maintain. It is easy to write rules that conflict, especially as a knowledge base is refined over time. If some change in the modeled system occurs, it is hard to localize the corresponding changes which must be made to the knowledge base. Secondly, knowledge bases are often not resuable. A knowledge base developed for a particular system is not easily modified to suit another similar system. Finally, the knowledge bases are hard to validate. They cover the experiential knowledge of a particular expert and it is not easy to check the soundness of the rules against the actual operation of the system or the completeness of the rules in describing all possible aspects of system behavior. The reader is referred to [Horvitz *et al.*, 1988] for a detailed discussion of the development and limitations of the early probabilistic approaches and rule-based systems. The key problem in the use of rule-based expert systems is that rules do not match the way people think about the domains in which they perform diagnosis. As a result, eliciting rules to model a domain is a difficult task.

1.2 Current approaches to diagnosis

1.2.1 Model-based diagnosis

The notion of *model-based diagnosis* arose when trying to address the problems encountered in the early approaches to diagnosis [Davis and Hamscher, 1988]. In this approach, diagnoses are computed from a causal model that describes how the system works. The model of the system is built by the composition of causal models of system components. The representation of the system can be viewed as a simulation model. The diagnosis process takes observations of system behavior as input and reasons backwards with the model to infer what component failures could explain the observations.

The underlying rationale behind the approach is that causal models of systems are built when the systems are being designed and analyzed. Hence, such causal information would be easier to specify than diagnostic rules. The model-based approach addresses the main drawbacks of the rule-based approach. Since information about components is localized within component models, maintenance of the model becomes easier. A change to a component model can be incorporated as a local change. If a component is used in two different systems, the same component model can be used in both systems. If the diagnosis algorithm is sound and complete, then the validation problem reduces to verifying the accuracy of the model of the system in modeling its behavior.

In a purely deterministic setting, model-based diagnosis consists of computing all possible combinations of failures of the components that explain the observations. Each such combination of component failures is a possible cause of the observations. A purely deterministic approach has no notion of likelihood.

A notion of likelihood, however, is crucial for practical applications. When doing diagnosis, our ultimate goal is to compute some low cost repair action in response to our beliefs about the cause of the system anomaly. The choice of the appropriate action depends crucially on the relative likelihoods of the various possible causes of the system anomaly. Furthermore, in large systems, the space of possible causes of an observation are very large. When performing diagnosis, examination of the entire space is not practical. We need to be able to focus on more likely causes [de Kleer, 1991].

The notion of likelihood is introduced in model-based diagnosis by introducing a prior probability of failure for each component[de Kleer and Williams, 1987]. This prior probability models the component's reliability. In this framework, diagnosis is the computation of the posterior probability distribution over the set of possible causes conditioned on the available observations. We would also like to compute optimal actions based on some cost model of the repair actions available [Sun and Weld, 1993].

A large number of model-based diagnosis implementations are based on the architecture of the General Diagnostic Engine (GDE) [de Kleer and Williams, 1987]. GDE uses an Assumption-Based Truth maintenance system (ATMS) [de Kleer, 1986] to record predictions of system behavior inferred from the system model. The ATMS is also used to record actual observed behavior of the system and conflicts between predicted behavior and observed behavior. The inference mechanism of the ATMS computes a minimal description of all possible diagnoses which are consistent with the current state of information.

1.2.2 Bayesian networks

In parallel to the development of model-based diagnosis, the formalism of *Bayesian networks* [Pearl, 1988] has developed as a technique for modeling probabilistic reasoning and building diagnosis systems. Bayesian networks have been applied to building diagnostic expert systems in domains where a coherent handling of uncertainty about the diagnosis is crucial. A major motivating factor for this application is that extensions to rule-based systems for handling uncertainty, such as certainty factors, cannot be given a clear semantics unless very strong limitations on the structure of the knowledge base are imposed[Horvitz and Heckerman, 1986].

A Bayesian network consists of a set of discrete valued variables X_1, X_2, $\ldots$,

X_n. The Bayesian network is a structured representation of the joint distribution $P(X_1, X_2, \ldots, X_n)$ of these variables. Any joint distribution $P(X_1, X_2, \ldots, X_n)$ can always be factorized as:

$$\begin{aligned} P(X_1, X_2, X_3, \ldots, X_n) &= P(X_1|X_2, \ldots, X_n) \times \\ & \quad P(X_2|X_3, \ldots, X_n) \times \\ & \quad \ldots \times P(X_j|X_{j+1}, \ldots, X_n) \times \\ & \quad \ldots \times P(X_n) \end{aligned}$$

Say we know that the probability distribution of X_j is independent of the variables X_{j+3}, X_{j+4}, $\ldots$, X_n given the values of the variables X_{j+1} and X_{j+2}. We can utilize this knowledge of independence in the above equation by simplifying the factor $P(X_j|X_{j+1}, \ldots, X_n)$ to merely be $P(X_j|X_{j+1}, X_{j+2})$. We can do this for every variable X_j. This gives a simplified structured representation of the joint distribution. The structure comes from explicit recognition of conditional independences.

A Bayesian network is a directed acyclic graph that provides a representation for such conditional independence information. Each variable X_j corresponds to a node in the graph. The parents of the node correspond to the variables that X_j is conditioned on after independencies have been recognized in the above equation. The conditional distribution of a node given its parents is stored within the node. The independence semantics of a Bayesian network provide a sound scheme to compute inferred conditional independencies from the ones declared explicitly (i.e., by the structure of the network). When constructing a Bayesian network of a domain, the ordering of the variables X_j is typically chosen to be in the "causal" direction. Roughly speaking, the ancestors of a node are its "causes" and its descendants are its effects. Choosing this order is usually the most natural scheme for modeling the domain.

A Bayesian network inference algorithm allows efficient computation of any conditional probability $P(X_i|X_j = x_j, X_k = x_k, \ldots)$ of interest from the joint distribution. Here, efficiency means that the independencies are exploited to improve the computational performance. On one extreme, if the Bayesian network is a fully connected graph, there are no independencies to exploit and the computation is exponential in the number of nodes. This is because we have no structure in the distribution. On the other extreme, if the Bayesian network is tree structured, the computation can be done in linear time. For more detail on Bayesian networks the reader is referred to [Pearl, 1988], [Charniak, 1991] and [Heckerman *et al.*, 1995b].

Like rule-based expert systems, most Bayesian network diagnostic systems are hand constructed by an expert. Many significant applications have been in the field of medical diagnosis (for example, [Heckerman *et al.*, 1990]). Currently, there is a growing realization of the synergy between the techniques of Bayesian networks and

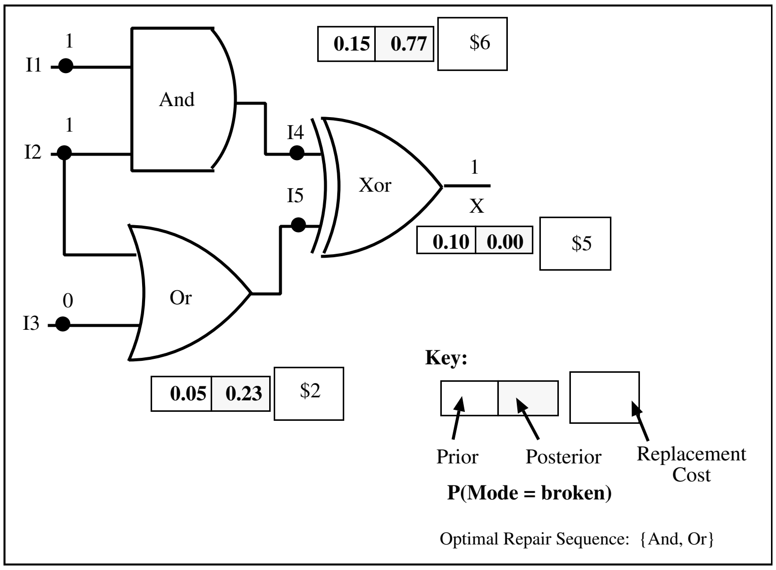


Figure 1.1: Introducing problems facing probabilistic model-based diagnosis: an illustrative example.

the methods used in probabilistic model-based diagnosis (for example, [Poole, 1993; Srinivas, 1994; Darwiche, 1995]).

In this dissertation, we address some crucial problems encountered in the practical application of probabilistic model-based diagnosis. Our solutions to some of these problems will apply techniques from Bayesian networks. A secondary goal of the dissertation is to provide some useful synthesis of probabilistic model-based diagnosis and Bayesian networks.

1.3 Making probabilistic model-based diagnosis practical

The practical application of probabilistic model-based diagnosis is hampered by some serious problems. We introduce them by means of an example.

Consider the system shown in Fig 1.1. The system is a simple digital circuit. The component model for each gate specifies a function that relates the input to the output. Each gate is assumed to be in one of two operating *modes*. The **ok** mode is the normal operating mode. When a gate is in the **ok** mode, the input determines the output according to the normal function of the gate. The **broken** mode models an anomalous state for the gate. The description of the input-output relation for an anomalous state is called a *fault model*. Fault models are often not available. However, in this case, let us assume that a *stuck-at-zero* fault model is appropriate for all the gates–i.e., when a gate is in the **broken** mode, the output is 0 irrespective of the inputs.

We also need a prior probability of failure for each gate. This is the *a priori* probability that the gate is in the **broken** state. This prior is often hard to specify. However, for the purposes of this example, assume we are able to assess the probabilities shown in the figure.

Diagnosis can now proceed. An *observation* consists of the reading of some input and output variables in the system. Consider the observation shown in the figure. Our goal is to compute posterior probabilities of failure for each gate given the observation (the posteriors are shown in the figure). The computation procedure needs to be one which will scale well from simple examples such as this to large systems with thousands of components.

Finally, say we have assessed a cost of replacement for each gate. Given the observation shown in the figure, say we want to compute some good repair strategy to correct the system anomaly. The actions available to us are replacements of gates. A repair strategy is some sequence in which the gates are to be replaced. After each gate replacement, we check whether the system anomaly is fixed. Each possible repair strategy has an associated expected cost. A good strategy has low expected cost. Thus, our goal is to compute a repair strategy that has a low (ideally, lowest) expected cost in the current context (the optimal sequence is marked on the figure). Once again, the computation procedure needs to be one that scales to large systems.

The modeling process and the computation goals sketched in the example brings up some important questions. These questions need to be answered before probabilistic model-based diagnosis and repair can be practical. In summary, the questions are:

- How do we do diagnosis tractably in large systems?
- How do we assess the prior probability of failure of components?
- How do we tractably compute inexpensive repair actions using the results of our diagnoses?
- How do we do diagnosis in a reasonable way if fault models are not available?

In this dissertation, we address these questions. In the following sections of this chapter, we describe these problems and our techniques for addressing them.

1.3.1 Tractable diagnosis

Computing a posterior probability distribution over the space of all assignments of modes to the components of a system is inherently intractable. We develop two complementary approaches to improve tractability—exploitation of hierarchy and focused generation of diagnoses.

Hierarchy

Hierarchical compositional models are widespread in engineering practice. The use of a hierarchy in system design has two interrelated advantages. Firstly, specification of the model becomes easier, and secondly, reasoning about the model becomes more tractable. We introduce a notion of hierarchy in the specification of models and develop an algorithm that exploits this hierarchy to give inferential gains during the diagnosis process.

We first develop a simple translation scheme that translates a system model into a Bayesian network. Computation of diagnoses now reduces to diagnostic inference within the Bayesian network. We then exploit the Bayesian network to develop a scheme that compiles a higher-level abstract model for a component from more detailed models of subcomponents by hiding internal detail. This is a pre-processing step that is performed off-line.

At run time, this compilation allows a tradeoff between time for inference and the level of granularity of the analysis. In other words, we can quickly get diagnoses at coarse levels of granularity (i.e., diagnoses involving higher level components only). If more inference time is available, we can also compute diagnoses over lower level components.

Other approaches to hierarchy in the model-based literature assume that the basic language for describing input-output specifications of components is propositional or first-order logic. Thus, specifications can either be deterministic or unspecified—they cannot be probabilistic. This causes problems when handling hierarchical models, since the higher level model cannot be compiled out of the lower level model. Hence, the hierarchy cannot be truly exploited to give inferential gains.

As an example, consider a component in which the higher level model has a non-normal mode called **broken** which subsumes two lower level mode assignments to the subcomponents of the component. Call these lower level mode assignments δ_1 and δ_2. Given an input $\mathbf{i}$, say the output of the subcomponent system is $\mathbf{o}_1$ if the mode assignment is δ_1 and the output is $\mathbf{o}_2$ if the mode assignment is δ_2. In this example, we see that the fault model of the higher level **broken** mode is not deterministic. However, it is not unspecified either. In actuality, it has to be describable by a probability distribution of the output given the input. Our technique computes this probability distribution.

Focused generation of diagnosis

For tractable diagnosis in large systems, we would like the diagnosis algorithm to be *focused*. That is, diagnoses which are more probable should be generated without examining the whole space of possible diagnoses.

We address this problem by developing a simple generate-and-test procedure. The generator is an efficient algorithm that successively generates *candidates* in decreasing order of prior probability. A candidate is an assignment of mode to each component in the system. The tester simply tests whether the candidate is consistent with the observations. This test can be performed by a simple forward simulation of the system model.

The first consistent candidate found is provably the diagnosis with the highest posterior probability. The practicality of this generate-and-test procedure hinges on the generator. The generator has to be efficient and has to provably generate candidates in decreasing order of probability.

When the component failures are independent, we first describe a generator algorithm that computes each successive candidate (in decreasing order of probability) in $O(n)$ where n is the number of components. We then develop a second algorithm that improves on the first algorithm by using information from previous consistency checks to rule out some of the candidates that have not yet been generated.

We then turn to the case where component failures are not independent. The dependencies between component failures are modeled as a Bayesian network. We develop an algorithm that generates the k-th most probable candidate in $O(Bk)$, where B is the size of the Bayesian network description.

The current approaches to focused generation of diagnoses cache inferences (for example, in ATMS labels) and prune the space of diagnoses based on the cached inferences. The amount of cached information grows very quickly with the size of the diagnosis problem and this causes tractability problems. An additional problem with the current state of the art is that there are no clear implementation independent specifications of the underlying algorithms or analyses of complexity. Our candidate generation algorithms cache no inferences and have clearly analyzed computational properties.

1.3.2 Eliciting the prior probabilities

Doing probabilistic model-based diagnosis requires specification of prior probabilities of failure of the components. This is often difficult. One of the main reasons for this is that the value of the failure priors depends on the time at which diagnosis is performed. This dependence has to be recognized and accounted for. For example, say we are asked to assess the prior probability that a car's timing chain is broken. The assessment clearly depends on the age of the timing chain. If the car has 10,000 miles on it, the prior is low. However, if the car has been driven for 90,000 miles the prior rises to a higher value.

To address this temporal dependence, we develop a model for the dependence of priors on time using techniques employed in Reliability Theory. As a consequence, we can calculate the prior probability of failure of a device from a better known *empirical* estimate of reliability, viz, the Mean Time Between Failures.

This work also provides a principled way to address the persistence problem when doing diagnosis with multiple observations. Say we get an observation Ω_1 at time t_1 and an observation Ω_2 at time t_2. What can we assume about the persistence of the mode of each component from the time of the first observation (t_1) to the time of the second observation (t_2)? Some model of persistence is necessary before we can compute a diagnosis which accounts for both observations.

This work provides a scheme for computing transition probabilities of the mode of each component from one state to another based on the time interval $t_2 - t_1$.

1.3.3 Computation of repair strategies

The goal of diagnosis is to repair a malfunctioning system. The posterior probabilities from the diagnosis process have to be used to choose an optimal sequence of actions that will bring the system back to working order.

Say each component in the system being diagnosed has an associated cost of repair. When the system is faulty, we need to determine a strategy for fixing the system. A strategy is an order in which to successively fix components until the system starts working. Ideally, the strategy we pick would have the least expected cost of all possible strategies that would fix the system.

We first develop an exhaustive technique that examines all possible repair strategies and computes the one with least expected cost. In the general case (i.e., without any restrictions on the nature of the system model), we cannot do better than examine all repair strategies.

We then demonstrate a special case of the general repair scheme that is often reasonable in practice and develop a polynomial time algorithm for computing the optimal repair strategy. In this special case, all components are assumed to fail independently. A system is assumed to fail if any of its components fail. Lastly, it is assumed that an observation of the system status is available after every step in the repair strategy. That is, after every component replacement, we can check whether the system is now working.

1.3.4 Incomplete fault models

A fault model specifies how the device behaves when it is in a non-normal mode. Fault models are often not available. However, fault models are necessary for doing probabilistic model-based diagnosis. We see why this is so in the following discussion.

Say we want to compute the posterior probability of a candidate m given an observation Ω. This is the basic goal of probabilistic diagnosis. All methods to compute this posterior implicitly or explicitly have to compute $P(\delta, \Omega) = P(\delta)P(\Omega|\delta)$. The quantity $P(\delta, \Omega)$ can then be normalized over all possible candidates δ to give $P(\Omega|\delta)$. The quantity $P(\delta)$ can be calculated by simply multiplying the appropriate priors of the components. The quantity $P(\Omega|\delta)$ is 1 if the mode assignment proves the observation. It is 0 if the mode assignment is inconsistent with the observation.

Consider the case where the mode assignment δ contains an "unknown" mode for a component $\mathcal{C}$. That is, the mode of $\mathcal{C}$ contained in δ does not have a specified fault model. In this case, we cannot compute $P(\Omega|\delta)$ and hence we cannot compute the posterior. Current methods approach this problem by assuming that when a mode assignment δ contains some "unknown modes", then all system outputs are equally likely. We first demonstrate that this approach can yield incorrect results. Specifically, it can be the case that no assignment of fault models to the individual components can result in all system outputs being equally likely for mode assignment δ.

A deeper problem with existing approaches is that they are *ad hoc*—there is no semantic understanding of what exactly the "equally likely outputs" assumption means. Since the ultimate goal of diagnosis is to do repair, we consider choosing a fault model such that the expected repair cost is as high as possible. Choosing a fault model in this way amounts to a conservative approach to estimating repair costs. We show that, with this approach, the problem of choosing the fault model reduces to a maximum entropy completion of a probability distribution applied locally at each component.

1.4 Organization of this dissertation

The organization of the rest of this dissertation follows the structure of the overview provided in this chapter. In Chapter 2, we introduce our notion of hierarchy and develop the hierarchical diagnosis algorithm. Chapter 3 discusses candidate generation and develops algorithms for candidate generation. Chapter 4 develops a model for the temporal dependence of the prior probability of failure on time. Chapter 5 introduces the problem of repair and develops an exhaustive strategy to compute optimal repair strategies. An interesting and useful special case which allows a polynomial time computation of the optimal repair strategy is developed and analyzed. Chapter

6 analyzes the problem of doing diagnosis with missing fault models. The drawbacks of current practice are first analyzed. A new method for choosing fault models based on maximizing expected repair costs is developed. Conclusions and future work are discussed in Chapter 7.

Chapter 2

Hierarchical Probabilistic Diagnosis

The use of hierarchical structure is ubiquitous in engineering models. A hierarchical model allows an engineer to manage the complexity of the modeling process. Each component is modeled as consisting of a set of interconnected subcomponents. The internal details of the subcomponents are independent of each other. Each subcomponent can, in turn, be modeled hierarchically. In addition to the modeling advantages, hierarchical models also make inference more tractable. This gain in tractability is achieved by exploiting the structure of the hierarchy.

In this chapter we develop a diagnosis algorithm that exploits a hierarchical model specification to give inferential gains. When reasoning with components at a high level of abstraction, diagnosis can be performed quickly. More computation can be performed to take the diagnosis to lower levels of abstraction. The hierarchical diagnosis algorithm allows a tradeoff between the complexity of inference and the granularity of the diagnosis.

Our approach is as follows. We first describe a method by which a non-hierarchical system model is translated into a Bayesian network. The process of diagnosis translates into inference in the Bayesian network. We then introduce hierarchy and describe how hierarchical models are defined. The translation scheme is extended to translate hierarchical models into a hierarchical Bayesian network.

Next, we develop a method by which the Bayesian network fragments obtained from the subcomponents of any component can be *compiled* into a Bayesian network fragment that represents the component at the next higher level of abstraction. This compilation process summarizes out details about internal variables in the subcomponents that are not relevant at the higher level of abstraction. This compilation process is performed off-line before the hierarchical model is used for diagnosis.

Finally, we adapt a Bayesian network inference algorithm to exploit the results of the compilation. Diagnostic inference at more abstract levels of the hierarchy is performed using pre-compiled network fragments of the abstract components. This results in a simpler Bayesian network and hence, less complexity when performing diagnostic inference.

2.1 From a system model to a Bayesian network

We now describe how the Bayesian network is created from a system model. A system model consists of a set of components. Each component has a set of discrete valued inputs $I_1, I_2, \ldots, I_n$ and a discrete valued output O. Each component also has a discrete valued *mode* variable M. Each state (i.e., value) of M is associated with a specific input-output behavior of the component. The components are connected according to the signal flow paths in the device to form the system model—we do not allow feedback paths.

The component specification requires two pieces of information—a function $F : I_1 \times I_2 \ldots I_n \times M \rightarrow O$ and a prior distribution over M. The prior distribution quantifies the *a priori* probability that the device functions normally. As an example, a component might have only two possible mode states **broken** and **ok**. If it is very reliable, a very low probability may be assigned to $P(M = \textbf{broken})$.

A Bayesian network fragment is created for a component as follows. A node is created for each of the input variables, the mode variable and the output variable. Arcs are added from each of the input variables and the mode to the output variable. The distribution $P(O|I_1, I_2, \ldots, I_n, M)$ is specified by the component function F. That is, $P(O = o|I_1 = i_1, I_2 = i_2, \ldots, I_n = i_n, M = m) = 1$ iff $F(i_1, i_2, \ldots, i_n, m) = o$. Otherwise, the probability is 0. The variable M is assigned the prior distribution given as part of the component specification.

The network fragments are now interconnected as follows: Whenever the output variable O^1 of a component $\mathcal{C}^1$ is connected to the input I_j^2 of a component $\mathcal{C}^2$, an arc is added from the output node O^1 of $\mathcal{C}^1$ to the input node I_j^2 of $\mathcal{C}^2$. This arc needs to enforce an equality constraint and so we enter the following distribution into node I_j^2: $P(I_j^2 = p|O^1 = q) = 1$ iff $p = q$, otherwise the probability is 0. After interconnecting the Bayesian network fragments created for each component, we have a nearly complete Bayesian network.

We now make some observations. The network created is indeed a DAG, and hence fulfills one of the necessary conditions for us to claim it is a Bayesian network. This condition is true because we did not allow any feedback in the original system model.

The probability distribution for every non-root node in the Bayesian network has been specified. This is because every non-root node is either (a) an output node or (b) an input node which is connected to a preceding output node. The probability distribution for every output node has been specified when creating the Bayesian network fragments. The probability distribution for every input node which has an output node as a predecessor has been specified when the fragments were interconnected.

The root nodes in the network fall into two classes. The first class consists of nodes corresponding to mode variables and the second class consists of nodes corresponding to some of the input variables. We note that the marginal probability distributions of all nodes in the first class (i.e., mode variables) have been specified.

The set of variables associated with this second class of nodes are those variables which are inputs to the entire system, i.e., these variables are inputs of components which are not downstream of other components. We will call this set of variables *system input* variables. Let us assume that the inputs coming from the environment to the system are all independently distributed. Further, let us assume for now that we have access to a marginal distribution for each system input variable[1]. We enter the marginal distribution for each system input variable into its corresponding node. We now have a fully specified Bayesian network.

Consider the original system model. We can interpret every component function and interconnection in the original system model as a constraint on the values that variables in the model can take (in the constraint satisfaction sense). We note that the Bayesian network that we have constructed enforces exactly those constraints that are present in the original model and no others. Further, it explicitly includes all the information we have about marginal distributions over the mode variables and the system input variables. The Bayesian network is therefore a representation of the joint distribution of the variables in the system model and the mode variables.

We now proceed to use the Bayesian network for diagnosis in the standard manner. Say we make an observation. An observation consists of observing the states of some of the observable variables in the system. As an example, we might have a observation which consists of the values (i.e., states) of all the system input variables and the output values of some of the components. We declare the observation in the Bayesian network. That is, we enter the states of every observed variable into the Bayesian network and then do a belief update with any standard Bayesian network inference algorithm (for example, see [Lauritzen and Spiegelhalter, 1988],[Jensen *et al.*, 1990]).

Say an observation $\Omega = \langle Y_1 = y_1, Y_2 = y_2, \ldots, Y_k = y_k \rangle$ has been made. After a Bayesian network algorithm performs a belief update, we have the posterior distribution $P(X|\Omega)$ available at every node X in the Bayesian network. The posterior distribution on each of the mode variables gives the updated probability of the corresponding component being in each of its modes. This constitutes the diagnosis

[1] If every observation of the system is guaranteed to contain a full specification of the state of the input, then the actual choice of priors is irrelevant.

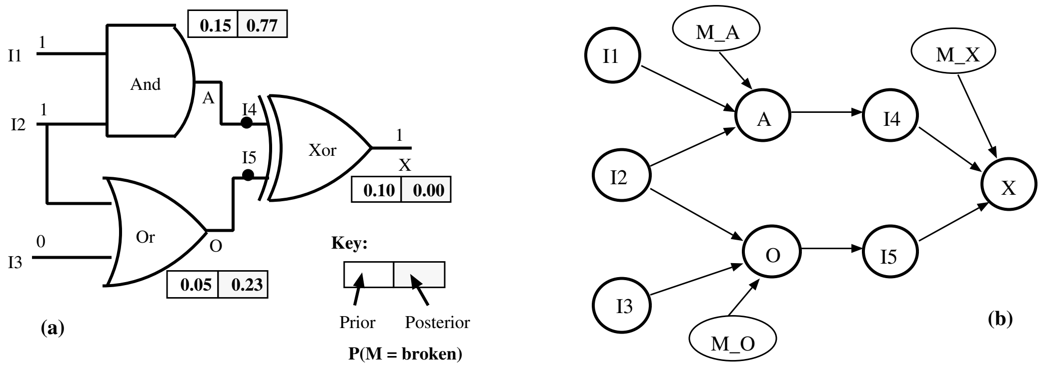


Figure 2.1: An example of (a) a system model and (b) the corresponding Bayesian network.

process.

We illustrate our scheme on the example introduced in Chapter 1. The system is a digital circuit (Fig 2.1(a)). We treat this circuit as our input system model. A particular observation (i.e., input and output values) is marked on the figure. We note that if the circuit was functioning correctly, the output for the marked inputs should be 0. Instead the output is a 1.

We assume, for this example, that each gate has two possible states for the mode variable, **ok** and **broken**. The modeler provides a prior on the mode of each gate—for each gate the prior probability of it being in the **ok** state is shown next to it in Fig 2.1(a). We also need a fault model. In other words, for each gate we need to have a function relating inputs to the output even if the mode of the gate is **broken**. We assume a *stuck-at-zero* fault model, i.e., if the gate is in state **broken** the output is 0 irrespective of what the input is. When the gate is in state **ok**, the function relating the inputs to the output is the usual Boolean function for the gate.

The Bayesian network corresponding to this system model is shown in Fig 2.1(b). We assume that the inputs are independently distributed, i.e., the value of any particular input is not dependent on the value of any other input. We also assume a uniform distribution as the prior for each of the inputs I_1, I_2 and I_3. Note that in this example, any (strictly positive) prior could be assumed without affecting the results of the diagnosis. This is because the state of the input is fully known when the diagnosis is performed. The observation is entered into the network and inference is performed. The posterior probability of being in the **ok** state for each gate is as shown in Fig 2.1(a).

2.2 Hierarchical system models

We now consider a situation where the modeler has conceptually broken up a system into a set of component subsystems. Say that each of the component systems has to be modeled (hierarchically) at a lower level of detail. We extend our modeling language to support such a feature.

The modeler must first fully specify the inputs, output and the mode variable of the component. By full specification we mean that the modeler specifies the number of inputs, the possible states of each input variable, the possible states of the output variable and the possible states of the mode variable.

If the modeler would now like to model the component at a lower level of abstraction, she can specify a new model as a detailed description of the component. This new model would have new components (we will call them subcomponents) which are interconnected. This lower level model is constrained in the following way: The system input variables of the lower level model should be the same as the input variables to the component specified at the higher level. Similarly, the system output variable of the lower level model should be the same as the component output variable at the higher level.

The modeler has to provide a final piece of information to complete the hierarchy—she has to relate the modes of the subcomponents to the modes of the component. To make this more concrete, consider a component which has two states for its mode variable—**ok** and **broken**. Say that it is modeled at a lower level of detail with 4 subcomponents, each of which has two possible states. If we consider the possible combinations of mode states at the lower level of abstraction there are $2^4 = 16$ possibilities. However, at the higher level of abstraction, there are only two possibilities to be considered, i.e., the granularity is not fine enough to distinguish individually between the 16 different possibilities at the lower level.

To relate the lower level to the higher level, the modeler has to provide a function describing how the lower level combinations of mode states relate to the higher level mode state. In other words, the modeler has to provide a categorization which separates the lower level state combinations into a set of bins. Each bin corresponds to one of the states of the mode variable at the higher level of abstraction. This function could be a simple rule. One possibility, for example, is the rule, "If anything is broken at the lower level, then consider the component broken at the higher level." This means, in our example, that 15 possibilities at the lower level fall into the **broken** bin at the higher level while only 1 possibility (i.e., no subcomponents broken) falls into the **ok** bin at the higher level.

Once this function is specified, the hierarchical model is complete. We will call this function the *abstraction function*. Note that we can have multiple levels of hierarchy. We also note two salient points. First, the modeler does *not* need to provide a component function at higher levels of the hierarchy. Second, the modeler does *not* need to provide a prior on the mode variable at higher levels of the hierarchy. In other words, if a component is modeled at a lower level of detail, then only the lower level model and the abstraction function are required. The component function and prior are required only for a component which is being modeled "atomically", i.e., it is not being modeled at any finer level of detail.

As an example of hierarchical modeling, consider an exclusive-OR (XOR) gate. We might represent the XOR gate at a lower level of detail and show that it is synthesized using AND gates, OR gates and inverters (Fig 2.2(a)). We use the following rule as the abstraction function: "If anything is broken at the lower level, then the XOR gate is broken".

2.2.1 Incorporating hierarchy into the translation scheme

When a component is modeled at a lower level of abstraction, the translation proceeds as follows: Assume that the higher level abstraction does not exist and just plug in the lower level system model between the system inputs and outputs and do the translation. In the resulting Bayesian network, introduce a new variable for the higher level mode. Call this M^h. Add an arc from the mode variable of each of the subcomponents to the higher level mode variable. Call the lower level mode variables $M^{l1}, M^{l2}, \ldots, M^{ln}$. Fill out the conditional probability distribution of the higher level mode variable as follows: $P(m^h|m^{l1}, m^{l2}, \ldots m^{ln}) = 1$ iff $m^h = Ab(m^{l1}, m^{l2}, \ldots, m^{ln})$, 0 otherwise.

Here Ab is the abstraction function relating combinations of mode states of the subcomponents to the mode of the higher level component. Fig 2.2(b) shows the Bayesian network for the XOR gate example.

As a first cut, diagnosis with a hierarchical functional model can proceed exactly as described with non-hierarchical models. If we want a fine grain diagnosis, we look at the updated posterior probabilities of the subcomponent modes. If we want a coarse grained diagnosis, we look at the updated posterior of the mode variable of the component at the higher level of abstraction. However, this simplistic solution does not exploit the hierarchy to deliver computational gains.

To get computational gains, we need to be able to reason with the higher level model in a way such that the detail of the lower level model has been "compiled away" into a more succinct higher level model. We now describe a scheme for doing so. Consider a component $\mathcal{C}^h$ which is modeled at a lower level of abstraction with a model consisting of subcomponents $\mathcal{C}^{l1}, \mathcal{C}^{l2}, \ldots, \mathcal{C}^{ln}$. The mode variable of $\mathcal{C}^h$ is M^h and the mode variable of subcomponent $\mathcal{C}^{li}$ is M^{li}. Let the inputs of $\mathcal{C}^h$ be $I_1^h, I_2^h, \ldots, I_m^h$. Let the output of $\mathcal{C}^h$ be O^h. Let all the internal variables of the lower level model (i.e., the input and output variables of the subcomponents excluding the

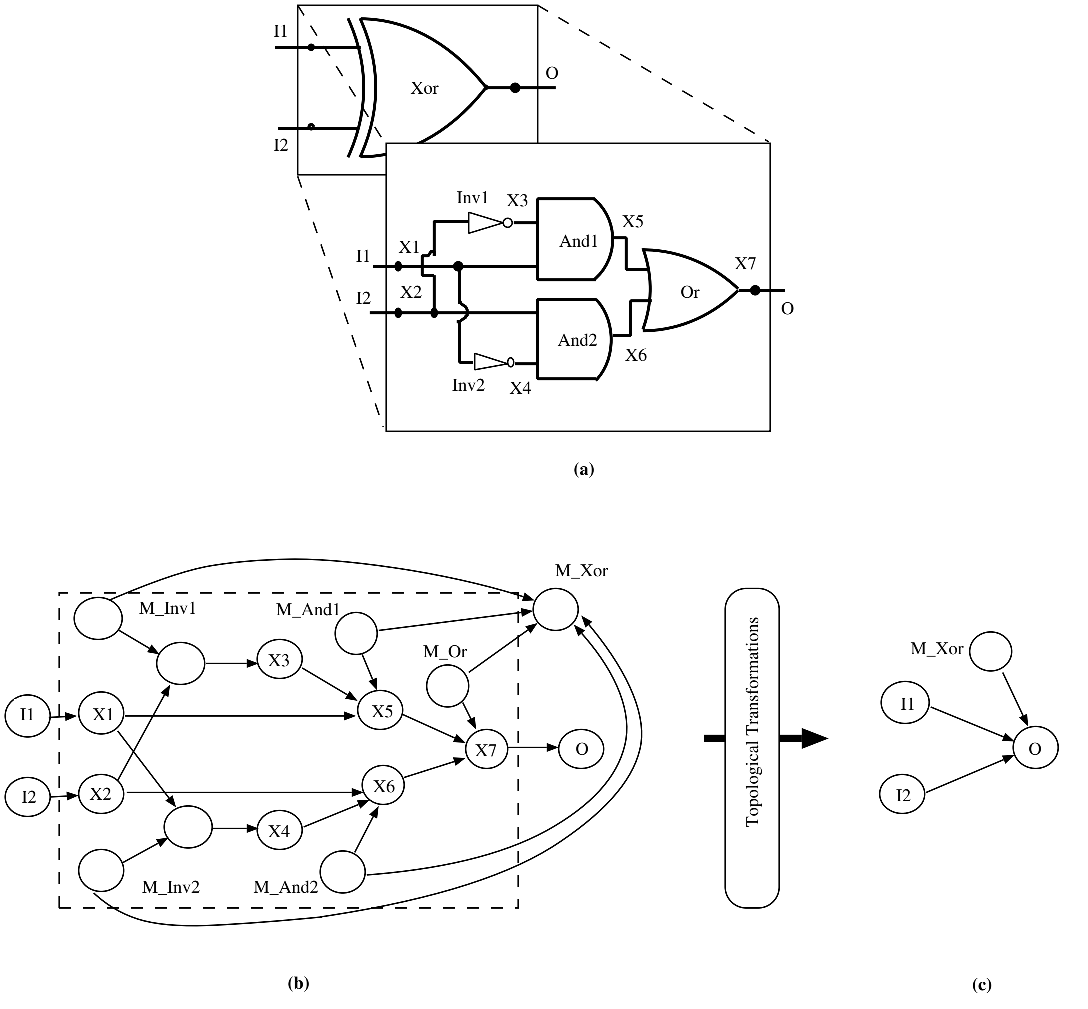


Figure 2.2: (a) A hierarchical system model (b) Corresponding Bayesian network fragment (c) The fragment after "compilation".

system inputs and outputs) be $X_1, X_2, \ldots, X_k$.

For simplicity, let us assume that all the inputs of $\mathcal{C}^h$ are system inputs, i.e., there are no components upstream of $\mathcal{C}^h$. We also assume, as described before, that we have a prior on each system input. Now consider the Bayesian network fragment created by the translation scheme for $\mathcal{C}^h$. We note that this fragment happens to be a fully specified Bayesian network.

A Bayesian network is a structured representation of the joint distribution of all the variables in the network. In this case, the network is a representation of the distribution $P(I_1^h, I_2^h, \ldots, I_m^h, O^h, M^h, M^{l1}, M^{l2}, \ldots, M^{ln}, X_1, X_2, \ldots, X_k)$. Call this the *lower level distribution.*

If now, we wanted to have a Bayesian network representation at the higher level of abstraction, we would not want to explicitly represent the detail about internal variables of the lower level model or the mode variables of the subcomponents. In other words we would like to have a Bayesian network which represents the joint distribution of only the input, mode and output variables of $\mathcal{C}^h$, i.e., the distribution $P(I_1^h, I_2^h, \ldots, I_m^h, O^h, M^h)$. Call this the *higher level distribution.*

We can generate the higher level distribution from the lower level distribution by simply marginalizing out all the irrelevant variables, viz, M^{l1}, M^{l2},..., M^{ln}, X_1, X_2,..., X_k. Ideally, we should do this marginalization in some efficient way. Such efficient marginalization is possible using topological transformations of Bayesian networks [Shachter, 1986]. Specifically, we can use the *arc reversal* and *node absorption* operations as follows:

1. Successively reverse the arcs $M^{l1} \rightarrow M^h$, $M^{l2} \rightarrow M^h$,..., $M^{ln} \rightarrow M^h$. At the end of this step M^h is a root node.

2. Let $\mathbf{X}$ be the set of internal variables of the lower level model, i.e., $\mathbf{X} = \{M^{l1}, M^{l2}, \ldots, M^{ln}, X_1, X_2, \ldots, X_k\}$. Sort $\mathbf{X}$ into a sequence $\mathbf{X}_{seq}$ in inverse topological order (descendants first). Absorb the nodes in $\mathbf{X}_{seq}$ in order into O^h.

This completes the process and leaves us with the topology shown in Fig 2.2(c). The successive absorption in the last step is always possible since there is no node N in the Bayesian network such that (a) N is not in $\mathbf{X}_{seq}$ and (b) the position of N has to necessarily be between two nodes contained in $\mathbf{X}_{seq}$ in a global topological order [Shachter, 1986]. Note that the topology which results from the marginalization process described above is the same as the one we would get if we had directly modeled $\mathcal{C}^h$ as an atomic component.

For simplicity of exposition, the description above assumes that the inputs of $\mathcal{C}^h$ are system inputs. However, this assumption is unnecessary. The identical marginalization process is possible for any hierarchically modeled component. We can consider

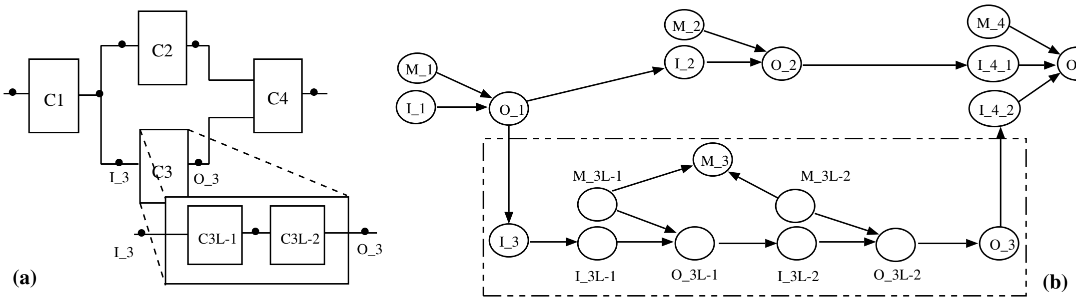


Figure 2.3: Exploiting hierarchy during diagnostic inference: (a) A hierarchical system model and (b) corresponding Bayesian network B^c.

the marginalization process that gives us the higher level distribution as a *compilation process* which is carried out after the model is created.

2.2.2 Exploiting hierarchy during diagnostic inference

The hierarchy in the system model can be exploited to improve diagnostic performance. We now describe a method of tailoring the clustering algorithm [Lauritzen and Spiegelhalter, 1988; Pearl, 1988; Jensen *et al.*, 1990] for Bayesian network inference to take advantage of the hierarchy. This is the most widely used algorithm in practice.

The clustering algorithm operates by constructing a tree of *cliques* from the Bayesian network as a pre-processing step. This construction is by a process called *triangulation* [Tarjan and Yannakakis, 1984]. The resulting tree is called the *join tree.* Each clique has some of the Bayesian network nodes as its members. As evidence arrives, a distributed update algorithm is applied to the join tree and the results of the update are translated back into updated probabilities for the Bayesian network nodes. The update process mentioned above can be carried out on any join tree that is legal for the Bayesian network.

We will now describe a method of constructing a legal join tree that is tailored to exploit the hierarchy. We explain by means of an example. Consider the hierarchical system model shown in Fig 2.3(a). This results in the hierarchical Bayesian network B^c shown in Fig 2.3(b).

After the lower level detail is compiled out we get the network in Fig 2.4(a). We add a dummy node D^h to this Bayesian network such that M_3, I_3 and O_3 are parents of D^h. Call this Bayesian network B^h. If we run a triangulation algorithm on this network we get a join tree (Fig 2.5(a)). Call this higher level join tree JT^h. We note

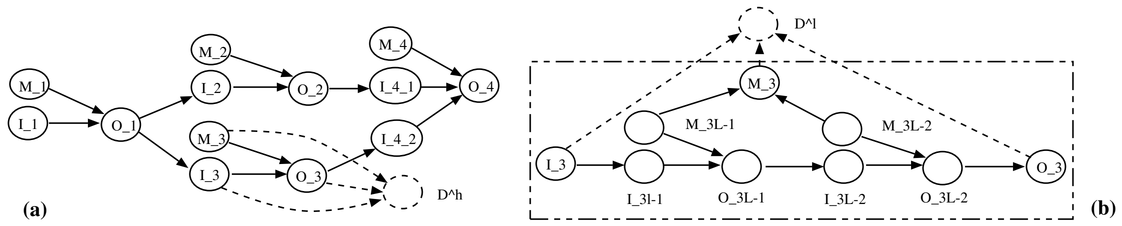


Figure 2.4: Expoliting hierarchy during diagnostic inference: (a) The compiled network B^h. (b) Lower level Bayesian network fragment B^l.

there exists a clique δ^h in JT^h such that I_3, M_3 and O_3 belong to δ^h. This is because I_3, M_3 and O_3 are parents of D^h. Triangulation guarantees that a Bayesian network node and its parents will occur together in at least one clique in the join tree.

Now consider the lower level network fragment by itself (Fig 2.4(b)). Call this B^l. Say we create a dummy node D^l and add arcs into it from I_3, M_3 and O_3 as shown in the figure. If we triangulate the graph we get a join tree (Fig 2.5(b)). Call this join tree JT^l. Once again, we are guaranteed that there is a clique δ^l in JT^l such that I_3, M_3 and O_3 belong to δ^l.

Now we construct a composite join tree JT^c from JT^h and JT^l. This is done by adding an link from δ^h to δ^l (shown as a dotted line in Fig 2.5). This composite join tree is a valid join tree for the network Fig 2.3(b) (Proof in a following section).

The composite join tree JT^c has the following interesting property. If the user is not interested in details about the lower level nodes, then the update operation can be confined purely to the JT^h segment of the join tree since only JT^h has any variables of interest. More precisely, if there is no evidence available regarding the states of the lower level nodes, *and* in addition, the user is not interested in the posterior distributions of the lower level nodes, then the update can be confined to the JT^h.

Say the user has finished an update in JT^h and then wants to view more detail by "opening the window" corresponding to the "iconified" component. In that case, the update process can proceed locally only though JT^l and give updated information. That is, the update process through the whole of JT^h need not be repeated—the information coming from the rest of JT^h is summarized in the message that δ_h sends δ_l when the incremental update process begins.

Along similar lines, if the user discovers evidence pertaining to a subcomponent, then she can "de-iconify" the containing component and assert the evidence. In this case, the update process begins in JT^l and proceeds through JT^h to make a global update. If one has multiple levels of hierarchy, the composite join tree has multiple

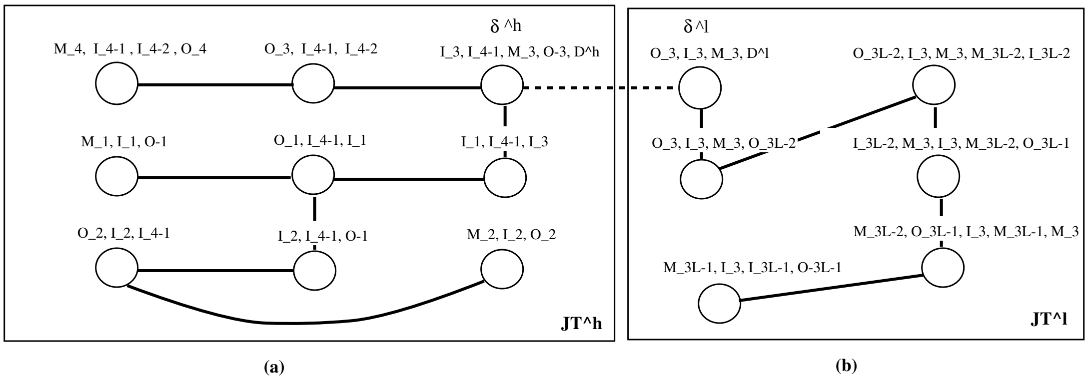


Figure 2.5: (a) JT^h (b) JT^l. Adding the link shown as a dotted line creates the composite tree JT^c.

levels of hierarchy too. At any time, the update process only affects that segment of the join tree that the user is interested in. This yields substantial savings in computation.

Quantifying the savings

Consider a hierarchical system specification that has k levels of branching. Say each component has at most b subcomponents. Further, assume that any variable in the system (i.e., input, output or mode) can take at most s states.

Consider computing the posterior of a leaf level component $\mathcal{C}_{leaf}$ while ignoring the hierarchy. To compute the posterior, we create the Bayesian network B_{leaf} of all the leaf level components and perform the posterior computation within this network. Let the number of nodes in B_{leaf} be N_{leaf}. We note that $N_{leaf} = O(b^k)$. The computation required to compute the posterior of a node in a Bayesian network with N nodes, each of which can take at most s states, is bounded by $O(N^s)$. Hence the computation of the posterior of B_{leaf} while ignoring hierarchy is $O(b^{(k \times s)})$.

Now consider computing the posterior of $\mathcal{C}_{leaf}$ while utilizing the hierarchy. Consider a path in the hierarchy tree from the root component $\mathcal{C}_{root}$ to the target leaf node $\mathcal{C}_{leaf}$. Let the components in this path be (in order): $\langle \mathcal{C}_{root}, \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n, \mathcal{C}_{leaf} \rangle$. Note that $\mathcal{C}_1$ is a subcomponent of $\mathcal{C}_{root}$, $\mathcal{C}_2$ is a subcomponent of $\mathcal{C}_1$ and so onwards.

Utilizing the hierarchy tree, we can compute the posterior probability of $\mathcal{C}_{leaf}$ as follows: We begin the update at the root component of the hierarchy tree and then extend the update process to its subcomponents. Of these subcomponents, we choose $\mathcal{C}_1$ and extend the update process to $\mathcal{C}_1$'s subcomponents. We proceed by extending the update process to $\mathcal{C}_2$'s subcomponents and so onwards, following the path from $\mathcal{C}_{root}$ to $\mathcal{C}_{leaf}$. At the end of this process, we have the posterior probability of $\mathcal{C}_{leaf}$.

Each extension of the update process takes $O(b^s)$. This is because there are $O(b)$ nodes in the lower level network in which the update is taking place. Since there are k levels of branching, the complexity of the entire process is $O(k \times b^s)$.

Define $p = b^s$. The complexity of inference without the hierarchy is then $O(p^k)$. The complexity of inference with the hierarchy is $O(k \times p)$. Thus, taking advantage of the hierarchy transforms an exponential computation to a linear computation.

2.2.3 JT^c is a valid join tree

The clustering algorithm constructs a valid join tree for a Bayesian network B as follows [Pearl, 1988]:

1. The Bayesian network B is converted into a Markov network G by connecting the parents of each node in the network and dropping the directions of the arrows in the DAG. G is an undirected graph.

2. A chordal supergraph G' is created from G by a process called *triangulation*. A chordal graph is one where any loop of length 4 or more has a chord (an arc connecting two non-consecutive edges in the loop). Basically, the triangulation process adds arcs to G until it becomes chordal.

3. The maximal cliques of the chordal graph G' are assembled into a tree JT. Each maximal clique is a vertex in the tree. The tree has the following *join tree property*: For every node n of B, the sub-tree of JT consisting purely of vertices which contain node n is a connected tree.

We now prove that JT^c is a valid join tree for the Bayesian network B^c. We do so by first describing the construction of a particular chordal supergraph $G^{c'}$ of the Markov network of B^c. We then show how JT^c is a valid join tree constructed from $G^{c'}$.

Consider a graph $G^{c'}$ constructed as follows: B^h is converted into a Markov network G^h. Similarly, B^l is converted into a Markov network G^l. Each of these networks are triangulated giving the chordal graphs $G^{h'}$ and $G^{l'}$.

$G^{h'}$ and $G^{l'}$ are merged to form a graph $G^{c'}$. This "merging" of the graphs is done as follows: The nodes M_3, I_3 and O_3 in $G^{h'}$ are merged with the corresponding nodes in $G^{l'}$. That is, $G^{c'}$ has only one copy of each of these nodes. Any link between any one of these nodes and a node in $G^{h'}$ is also present in $G^{c'}$. Similarly any link between any of these nodes and a node in $G^{l'}$ is also present in $G^{c'}$.

Lemma 1: $G^{c'}$ is a chordal supergraph of the Markov network G^c created from B^c.

Proof:

1. Consider a graph G^m obtained by merging the Markov networks G^h and G^l of B^h and B^l. We note that G^m is a supergraph of the Markov network G^c of B^c.

   $G^{h'}$ is a supergraph of G^h and $G^{l'}$ is a supergraph of G^l. Hence the graph $G^{c'}$ obtained by merging $G^{h'}$ and $G^{l'}$ is a supergraph of G^m.

   Since $G^{c'}$ is a supergraph of G^m and G^m is a supergraph of G^c, by transitivity we have: $G^{c'}$ is a supergraph of G^c.

2. We now consider loops in the graph $G^{c'}$. Any loop of length 4 or more which stays completely within the nodes of the subgraph $G^{h'}$ or the nodes of subgraph $G^{l'}$ has a chord. This is because the two sub-graphs are triangulated.

   Now consider a loop L which lies partially in $G^{h'}$ and partly in $G^{l'}$. We see that L has to pass through one of the M_3, I_3 and O_3 as it goes from $G^{h'}$ to $G^{l'}$. Further, L has to pass through one of these nodes as it passes back from $G^{l'}$ to $G^{h'}$. This is because these are the only common "boundary" nodes between the two subgraphs.

   We are guaranteed by the triangulation process that there are links between any pair of the nodes in the set $\{M_3, I_3, O_3\}$ in both $G^{h'}$ and $G^{l'}$. Hence, these links are also present in $G^{c'}$. Thus, the two "boundary" nodes in L are connected by a chord. This chord breaks L into two sub-loops, each of which lies either entirely within $G^{h'}$ or $G^{l'}$. Each of these sub-loops is guaranteed to have a chord if it has length greater than 4. This implies that L in $G^{l'}$ is also guaranteed to have a chord if it has length greater than 4. Thus $G^{c'}$ is a chordal graph.

We see from (1) and (2) that $G^{c'}$ is a chordal supergraph of the Markov network created from B^c. QED.

Lemma 2: JT^c is a valid join tree created from $G^{c'}$.
Proof:

1. Consider a maximal clique C^l in $G^{l'}$ which contains a node n such that n does not occur in $G^{h'}$. We note that no new links to n are added in the merger of $G^{h'}$ and $G^{l'}$. It follows that C^l is a maximal clique containing n in $G^{c'}$. A similar argument applies to any clique C^h in $G^{h'}$ which contains a node n which does not occur in $G^{l'}$. In sum, we conclude that any maximal clique in $G^{l'}$ (similarly, $G^{h'}$) which contains at least one node which occurs solely in $G^{l'}$ (similarly, $G^{h'}$) is also a maximal clique in $G^{c'}$.

   Say there exists a maximal clique C in $G^{l'}$ such that *all* its nodes are common to both $G^{l'}$ and $G^{h'}$. We see that C must contain a strict subset of the nodes in the set $\{M_3, I_3, O_3\}$. We note that there necessarily is a clique C' in $G^{l'}$ which contains M_3, I_3, O_3 and D^l. Since no subset of $\{M_3, I_3, O_3\}$ is a maximal clique, we conclude, by contradiction, that C cannot exist. Thus every maximal clique in $G^{l'}$ has at least one node which occurs only in $G^{l'}$.

As a result, every maximal clique of $G^{l'}$ is necessarily a maximal clique in $G^{c'}$. By a similar argument, every maximal clique in $G^{h'}$ is necessarily a maximal clique in $G^{c'}$. We also observe that no new maximal cliques are created when $G^{l'}$ and $G^{h'}$ are merged.

In sum, we see that the following is true: The set of maximal cliques of $G^{c'}$ is the union of two disjoint sets, namely, the set of maximal cliques of $G^{h'}$ and the set of maximal cliques of $G^{l'}$.

JT^c was constructed by linking a vertex of JT^h to a vertex of JT^l. The vertices of JT^h and JT^l are the maximal cliques of $G^{h'}$ and $G^{l'}$ respectively. Hence, it follows from the previous para that the vertices of JT^c are the maximal cliques of $G^{c'}$.

2. We now prove that JT^c has the *running intersection property* (r.i.p). We note that the r.i.p holds for any node n of B^c which appears solely in B^h or solely in B^l. This is because the r.i.p certainly is true in JT^h and JT^l and node n appears solely in vertices of one of these trees.

   The only nodes which appear in both B^h and B^l are M_3, I_3 and O_3. Consider the node M_3. Say we consider the subgraph J' of JT^c such that every vertex has M_3 as a member. Part of this subgraph lies in JT^h and part lies in JT^l. We know that the section which lies in JT^h is connected and the part which lies in JT^l is connected. This is because JT^h and JT^l are valid join trees.

   JT^c was constructed by linking vertex δ^h of JT^h to vertex δ^l of JT^l. We see that δ^h and δ^l both necessarily contain M_3. This means the subgraph J' is actually a tree. Hence the r.i.p is satisfied for M_3 in JT^c. A similar argument proves that the r.i.p holds for O_3 and I_3 too.

   Since the r.i.p holds for every node of B^c in JT^c, we conclude that JT^c satisfies the running intersection property. QED.

Theorem: JT^c is a valid join tree for the Bayesian network B^c.

Proof: This follows directly from Lemma 1, Lemma 2 and the construction procedure for join trees. QED.

The dummy nodes D^h and D^l are present solely to force a particular topology on the join trees JT^h and JT^l. After the triangulation process, they can be dropped from the cliques which contain them. This might sometime result in a simplification of the composite join tree. Consider the case where δ^l is reduced to $\{M_3, I_3, O_3\}$ after D^l is dropped. In this situation, δ^l can be merged with δ^h since it is a subset of δ^h.

Similarly δ^h can be merged with δ^l if δ^l reduces to $\{M_3, I_3, O_3\}$ after δ^h is dropped. JT^c continues to be a valid join tree after such mergers.

2.3 Discussion

Geffner and Pearl [Geffner and Pearl, 1987] describe a scheme for doing distributed diagnosis of systems with multiple faults. They devise a message passing scheme by which, given an observation, a most likely explanation is devised. An explanation is an assignment of a mode state to every component in the model. The translation scheme we describe can be used to achieve an isomorphic result. That is, instead of using a Bayesian network update algorithm to compute updated probabilities of individual faults, we could use a *dual* algorithm for computing composite belief [Pearl, 1987] and compute exactly the same result.

From the perspective of our work, [Geffner and Pearl, 1987] have integrated the inference in the Bayesian network into the system model as a message passing scheme. Separating out the network translation explicitly allows features such as hierarchical diagnosis and computation of updated probabilities in individual components as opposed to composite beliefs.

Mozetič [Mozetič, 1991] lays out a formal basis for diagnostic hierarchies and demonstrates a diagnostic algorithm which takes advantage of the hierarchy. The approach is not probabilistic. However, he includes a notion of non-determinism in the following sense: Given the mode of a component, he allows the input-output mapping of a component to be a relation instead of a function, i.e., there can be multiple possible outputs for a given input. The notion of hierarchy we have described here corresponds to one of three possible schemes of hierarchical modeling that he describes. Our scheme can be expanded to support a probabilistic generalization of the other two schemes of modeling and his notion of non-determinism.

Genesereth [Genesereth, 1984] describes a general approach to diagnosis including hierarchies. He distinguishes between structural abstraction and behavioral abstraction. In structural abstraction, a component's function is modeled as the composition of the functions of subcomponents whose detail is suppressed at the higher level. This is similar to what we have described. Behavioral abstraction corresponds to a difference in how the function of a device is viewed. For example, a low level description of a logic gate might model input and output voltages as real numbers while a high level description might model them as "high" and "low". Behavioral abstraction often corresponds to clustering sets of input values at the low level into single values at the higher level. Our method extends to support such abstractions in a straightforward manner.

Yuan [Yuan, 1993] describes a framework for constructing decision models for hierarchical diagnosis. The decision model is comprised of the current state of knowledge, decisions to test or replace devices and a utility function that is constructed on the fly. A two-step cycle comprising model evaluation and progressive refinement is proposed. The cycle ends when the fault is located. It is assumed that only a single fault exists. Model refinement is in accordance with the structural hierarchy of the device. The goal is to provide decision theoretic control of search in the space of candidate diagnoses. Such a framework needs a scheme for computing the relative plausibility of candidate diagnoses. Our work provides such a scheme in a general multiple fault setting.

The results described in this chapter have also been presented in [Srinivas, 1994].

Chapter 3

Focused Generation of Diagnoses

In a large system, there are usually a large number of possible diagnoses that can account for an anomalous observation. In the worst case, the number of possible diagnoses is exponentially large. As a result, a mechanism to focus the diagnosis process on more likely explanations has been an area of considerable interest in model-based diagnosis. This focusing mechanism has to be able to compute more probable diagnoses without examining the entire space of possible diagnoses.

Most published implementations of model-based diagnosis, e.g., [de Kleer and Williams, 1987; de Kleer and Williams, 1989; Struss and Dressler, 1989; de Kleer, 1991; Hamscher, 1991], are based on some variation of an ATMS [de Kleer, 1986]. ATMS-based implementations are well suited to situations in which all possible diagnoses need to be characterized, since the ATMS is optimized for reasoning simultaneously in all contexts. However, with the trend towards finding just a small number of leading diagnoses, the need for reasoning simultaneously in all contexts diminishes, and the benefits of using the basic ATMS are questionable.

In response to this trend, various focusing strategies have been developed that attempt to harness the combinatorial explosion inherent in ATMS algorithms, e.g., [Collins and DeCoste, 1991; Dressler and Farquhar, 1990; Forbus and de Kleer, 1988], with the most recent being the development of the HTMS [de Kleer, 1994], a hybrid TMS that combines properties of an ATMS with those of an LTMS [McAllester, 1980]. Intuitively, the basic ATMS caches all inferences in all contexts using *node labels*, while the focusing strategies use node labels to cache only some inferences in some contexts, potentially yielding dramatic speedups, e.g., see [de Kleer, 1991]. However, a drawback of ATMS-based methods, and of these focusing strategies, is that they are very complicated, and much of their benefit can be lost unless great care is taken in programming, e.g., by using sophisticated data structures. This is a significant hurdle for researchers and practitioners attempting to reimplement these methods in their own model-based diagnosis systems. Furthermore, even the best focusing strategy (the HTMS strategy) still caches enough information in node labels so that, on very large examples, the working set of the algorithm can significantly exceed real memory, leading to poor performance [de Kleer, 1991].

In this chapter, we address the above drawbacks by developing a set of simple, easy to implement, *generate-and-test* diagnosis algorithms. These algorithms enumerate diagnoses in decreasing order of posterior probability. Each algorithm has a generator that systematically generates candidates (potential diagnoses) in decreasing order of prior probability. The tester checks to see if a generated candidate is a diagnosis, i.e., whether the candidate is consistent with the observations. Unlike the ATMS-based algorithms, our algorithms cache almost no inferences. This effectively addresses the working set problem alluded to above.

An efficient candidate generation scheme is the key component of our algorithms. Testing of a candidate is simply a consistency check that determines whether a given diagnosis is consistent with the observations. As we shall see below, this can be accomplished through a simple forward simulation of the system. The description of our algorithm thus reduces to the description of the candidate generation. In the following sections, we first develop an algorithm which makes the standard assumption that component failures are independent. We then improve this algorithm by incorporating information from previous tests to focus future generation. This leads to a substantial improvement in performance. Finally, we develop a algorithm for the situation where the component failures may be dependent. The dependence of component failures is specified using a Bayesian network.

The results in this chapter are joint work with Pandurang Nayak. To the extent that our contributions can be separated, I am the primary contributor to the first algorithm for candidate generation with independent component failures (initial part of Section 3.2) and the algorithm for candidate generation when component failures are dependent (Section 3.3). Pandu is the primary contributor to the focused diagnosis algorithm developed in Section 3.2.1.

3.1 A generate-and-test approach

We now define the generate-and-test scheme more precisely. Say we are given a system model S with fully specified component fault models, a candidate diagnosis δ and an observation $\Omega = \{\mathbf{I} = \mathbf{i}, \mathbf{O} = \mathbf{o}\}$. A candidate is an assignment of a mode to each component in the system. An observation Ω consists of the values of the vector of system input variables (designated by $\mathbf{I}$) and the values of the output variables of some of the components (designated by $\mathbf{O}$). Given a number k, our goal is to efficiently compute a set of candidates of cardinality k such that each candidate δ in the set has a higher posterior probability $P(\delta|\Omega)$ than any candidate outside the set. In other words, we want to compute the k most probable candidates given the observation.

We can easily verify whether an observation is consistent with a candidate by simply simulating the system forward with the input $\mathbf{i}$ while assuming the components are in the states specified by δ. We compare the simulation output $\mathbf{o}_{sim}$ with the observed output $\mathbf{o}$. If $\mathbf{o}_{sim} = \mathbf{o}$, Ω is consistent with S and δ. Furthermore, we note that $\mathbf{i}, \delta, S \models \mathbf{o}$, or equivalently, $P(\mathbf{o}|\mathbf{i}, \delta, S) = 1$. If $\mathbf{o}_{sim} \neq \mathbf{o}$, Ω is inconsistent with S and δ. Furthermore, we note that $i, \delta, S \models \neg o$ or equivalently, $P(\mathbf{o}|i, \delta, S) = 0$.

We note that given a candidate δ with prior probability $P(\delta)$, we have:

$$\begin{aligned} P(\delta|S,\Omega) &= P(\delta|S,\mathbf{i},\mathbf{o}) \\ &= \frac{P(\mathbf{o}|\mathbf{i},\delta,S)P(\delta)}{P(\mathbf{o}|\mathbf{i},S)} \\ &\propto P(\mathbf{o}|i,\delta,S)P(\delta) \end{aligned} \tag{3.1}$$

The above equations account for the fact that the choice of the system description S, the distribution over the system input $\mathbf{I}$ and the distribution over the candidates δ are all mutually independent.

From Equation 3.1 and the preceding discussion about $P(\mathbf{o}|\mathbf{i}, \delta, S)$, we note that if the observation is consistent with the candidate and system description, the posterior probability of the candidate is proportional to its prior. If the observation is inconsistent with the candidate and system description, the posterior probability is zero.

Say we have a efficient generator which generates candidates in decreasing order of prior probability. As we noted above, we can easily test whether the observation is consistent with each of these candidates as they are generated. The first candidate found consistent by the tester is hence the candidate with the highest posterior probability. In general, the n-th candidate found consistent by the tester is the candidate with the n-th highest posterior probability. Though we cannot compute the probability of any consistent candidate (since we do not have the normalizing constant), we note that the ratio of the probabilities of any two consistent candidates is the ratio of their priors. This prior is available.

We now describe an efficient generator with the property described above, viz, that candidates are generated in decreasing order of prior probability. We first examine the case where the faults are independent. Later, we examine the case where the faults are not independent.

3.2 Independent Faults

Consider a system with n components, $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$. In this section we assume that the components fail independently, i.e., the random variables M_i are mutually

```
Process Gen(⟨C_n, C_{n-1}, ..., C_1⟩)
  If ⟨C_n, C_{n-1}, ..., C_1⟩ is empty then
    Sleep; Output(δ_φ);
    Sleep; Output(EOL); STOP.
  P_{n-1} = Gen(⟨C_{n-1}, ..., C_1⟩)
  L_{n-1} = [TBC]
  Set ptr_ok and ptr_broken to head of L_{n-1}.
  begin loop
    Sleep
    If ptr_ok and ptr_broken both point to EOL:
      Output(EOL); STOP.
    If either ptr_ok or ptr_broken points to TBC:
      δ^sub_next = Request(P_{n-1})
      Replace TBC with δ^sub_next in L_{n-1}.
      Add a new list element containing TBC
      at the end of L_{n-1}.
    Output(GetNext(C_n, ptr_ok, ok, ptr_broken, broken))
  end loop
End Gen
```

```
Procedure GetNext(C_x, ptr_1, s_1, ptr_2, s_2)
  Let δ^sub_1 be the element of L_{n-1} pointed to by ptr_1.
  Similarly, for ptr_2 we have δ^sub_2.
  If δ^sub_1 = EOL then
    winner = 2
  else if δ^sub_2 = EOL then
    winner = 1
  else if (P(M_x = s_1) × P(δ^sub_1) >
          P(M_x = s_2) × P(δ^sub_2)) then
    winner = 1
  else
    winner = 2
  Move ptr_winner to point to the next element
  ( i.e., the element following δ^sub_winner in L_{n-1}).

  Create δ_next by splicing "M_x = s_winner" onto δ^sub_winner.
  Return(δ_next)
end GetNext
```

Figure 3.1: Algorithm for generating all candidates in order.

independent. Hence, the prior probability distribution is specified by specifying the prior distribution over the mode variable M_i of each component $\mathcal{C}_i$. In other words, we specify $P(M_i = m)$ for each mode m of component $\mathcal{C}_i$. Given a candidate $\delta = \{M_1 = m_1, M_2 = m_2, \ldots, M_n = m_n\}$, the prior probability of δ is simply $P(\delta) = \Pi_i P(M_i = m_i)$. We now develop an efficient recursive algorithm that successively computes the candidates in decreasing order of prior probability.

Let us assume for now that each $\mathcal{C}_i$ has exactly two states, **ok** (the normal state) and **broken** (a "broken" abnormal state). Let $S_n = \langle \mathcal{C}_n, \mathcal{C}_{n-1}, \ldots, \mathcal{C}_1 \rangle$ be some (arbitrary) sequence of components. Let L_n be the list of candidates generated from the variables in S_n ordered in decreasing order of prior probability. Consider the subsequence $S_{n-1} = \langle \mathcal{C}_{n-1}, \mathcal{C}_{n-2}, \ldots, \mathcal{C}_1 \rangle$. Let a subcandidate be a mode assignment to all the components in S_{n-1}, and let L_{n-1} be the subcandidates generated in decreasing order of prior probability. Given L_{n-1}, we can generate L_n easily, as follows. Tack $M_n = \textbf{ok}$ onto every element of L_{n-1}. This gives a list L_{ok} of candidates in decreasing order of probability. Component $\mathcal{C}_n$ is in the **ok** mode in each of these candidates. The probability of each candidate is computed by multiplying $P(M_n = \textbf{ok})$ with the probability of the subcandidate. Similarly, for the mode **broken** of $\mathcal{C}_n$, we have the list L_{broken}. We then need only merge L_{ok} and L_{broken} to construct the list L_n. The above procedure generates the *entire* list of candidates in order and is therefore inherently exponential.

Our goal, however, is to develop a generator which will generate only the next most probable candidate on every call. We can achieve this goal by simply making the procedure described above evaluate *lazily*. The procedure can be viewed as a sleeping process. When this process is awakened, it does just enough computation to compute and output the next most probable candidate (i.e., the next element of L_n) and then goes back to sleep.

The algorithm *Gen* of Fig 3.1 implements exactly this observation. The call $Gen(\langle\langle \mathcal{C}_n, \mathcal{C}_{n-1}, \ldots, \mathcal{C}_1 \rangle)$ returns a process object $\mathcal{P}_n$, which behaves as follows: right after initialization it goes to sleep; when we request the first candidate from $\mathcal{P}_n$, it wakes up, outputs the first element of L_n and goes to sleep. The next request generates the next element of L_n, the third request generates the third element of L_n and so onwards. Finally, $\mathcal{P}_n$ outputs **EOL** to signify the end of L_n and stops. $\mathcal{P}_n$ thus computes L_n lazily.

We now examine the algorithm *Gen*. In the base case, when *Gen* is called with an empty sequence, the first request to the resulting process returns the empty candidate δ_ϕ. The empty candidate has no modes and has probability 1. The next request returns the end of list marker **EOL**. When we make the call $Gen(\langle \mathcal{C}_n, \mathcal{C}_{n-1}, \ldots, \mathcal{C}_1 \rangle)$ the resulting process $\mathcal{P}_n$ first initializes a subprocess $\mathcal{P}_{n-1} = Gen(\langle \mathcal{C}_{n-1}, \ldots, \mathcal{C}_1 \rangle)$ which, on request, will successively give it the elements of L_{n-1}. $\mathcal{P}_n$ then initializes the list L_{n-1} to contain just a single element **TBC**. **TBC** is a "to be computed" marker which signifies that the element in that location of L_{n-1} has *not yet been computed* by $\mathcal{P}_{n-1}$.

In its main loop, $\mathcal{P}_n$ simply merges L_{ok} and L_{broken} as described above. However, it does this lazily. The current position in L_{ok} is maintained by maintaining a pointer ptr_{ok} to the corresponding subcandidate in L_{n-1}. Similarly, ptr_{broken} into L_{n-1} maintains the current position in L_{broken}. On each request, $\mathcal{P}_n$ goes once through its main loop before going back to sleep. On each pass, it computes the next candidate in L_{ok} and L_{broken} from their corresponding subcandidates in L_{n-1}. The candidate with higher probability is selected by the procedure $GetNext$ and output.

When creating the candidates to compare, $\mathcal{P}_n$ may find that the subcandidate that is required is yet to be computed; this happens when one or both of ptr_{ok} and ptr_{broken} point to the **TBC** element in L_{n-1}. When this happens, $\mathcal{P}_n$ simply requests $\mathcal{P}_{n-1}$ for the next subcandidate (call it δ^{sub}_{next}) and replaces **TBC** by δ^{sub}_{next} in L_{n-1}. It then adds a new list element at the end of L_{n-1} containing **TBC**. This is to signify that now the subcandidate *after* δ^{sub}_{next} is yet to be computed. This algorithm can be easily generalized to the case where each $\mathcal{C}_i$ has k_i modes. In that situation, we will be merging k_i lists in the process $\mathcal{P}_i$.

We see that every time we request $\mathcal{P}_n$ for the next candidate, it (a) performs $(k_n - 1)$ comparisons (in $GetNext$) and (b) makes *at most* one request to $\mathcal{P}_{n-1}$. If the total number of comparisons made by $\mathcal{P}_n$ in one pass through the loop is $R(n)$, we see that $R(n) \leq (k_n - 1) + R(n-1)$. Hence $R(n) = O(k_n + k_{n-1} + \ldots + k_1)$. $\mathcal{P}_n$ thus computes the next candidate with $O(N)$ comparisons where $N = \Sigma_i k_i$, the total number of component modes, which is linear in n.

We note that the process oriented description of Gen is for ease of presentation only. The processes $\mathcal{P}_i$ are really coroutines since exactly one of them can be running at any given instant, i.e., there is no parallelism. We can simply summarize the *local* state of each process $\mathcal{P}_i$ by the variable L_i and the pointers into it. Call this state St_i. When using $\mathcal{P}_n$, the global state can be summarized by $\langle St_n, St_{n-1}, \ldots, St_1 \rangle$. Thus the "request" to the $\mathcal{P}_n$ "process" can be implemented as a simple manipulation of a global state data structure. Our implementation of Gen uses precisely this idea (see Section 3.2.2).

The above algorithm runs very fast in practice, and is quite adequate for small to medium sized examples (see Section 3.2.2). However, for large examples, the number of candidates that need to be generated and checked for consistency becomes very large, and the above algorithm is impractical. We now describe an algorithm that focuses candidate generation by incorporating information from candidates which have been examined earlier and found to be inconsistent.

3.2.1 Focused candidate generation

In focused candidate generation, the basic idea is to use information from previous consistency tests to rule out some future candidates. The candidates which are ruled out need not be generated and tested. For example, consider a system with four components $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$ and $\mathcal{C}_4$. Say the candidate $\delta_a = \{M_1 = \mathbf{ok}, M_2 = \mathbf{ok}, M_3 = \mathbf{ok}, M_4 = \mathbf{broken}\}$ has been found to be inconsistent with the observation. Furthermore, say that in the process of checking the consistency of δ_a, the tester discovers that the sub-assignment $\delta_{sub} = \{M_1 = \mathbf{ok}, M_2 = \mathbf{ok}\}$ is, in itself, inconsistent with the observation. Note that the candidate δ_a is a superset of this sub-assignment and hence also has to be inconsistent. Now we can conclude that any other superset of the sub-assignment δ_{sub} is also inconsistent. So, for example, say the candidate $\delta_b = \{M_1 = \mathbf{ok}, M_2 = \mathbf{ok}, M_3 = \mathbf{broken}, M_4 = \mathbf{ok}\}$ has not yet been generated. At this point, we already have enough information to rule it out and need not generate it.

An inconsistent sub-assignment such as δ_{sub} is called a *conflict*. Conflicts can be discovered during the simulation that checks the consistency of a candidate. Say the output of some component $\mathcal{C}_i$ is part of the observation. During the simulation, if the simulation output of $\mathcal{C}_i$ does not match the observed value, then the mode assignment of $\mathcal{C}_i$ together with the mode assignments to all its ancestors in the system model form a conflict.

The focused diagnosis algorithm operates as a best first search (see Fig 3.2). A typical best first search has three major components: (a) an agenda that holds unprocessed *nodes* in decreasing priority order; (b) a *goal test* that determines whether a node is a solution; and (c) a procedure to generate the *successors* of a node. The main loop in a best first search removes the best node from the agenda, checks whether its a solution, and adds the node's successors to the agenda. In our case, each node is a candidate, the priority order is the prior probability of the candidate, and the goal test is the consistency check against the observations.

We now turn to the definition of the "successors" of a candidate. In the following, assume that the **ok** mode is the most likely mode of each component, and assume that the remaining modes of each component are fault modes[1]. If δ is a candidate, let *faulty*(δ) be the subset of δ denoting just the faulty component modes in δ. Note that given *faulty*(δ), one can construct δ by assigning the **ok** mode to components not assigned modes in *faulty*(δ).

Let δ_1 and δ_2 be any two candidates. If *faulty*(δ_1) $\subset$ *faulty*(δ_2), then $P(\delta_1) \geq P(\delta_2)$. Hence, candidate generation can generate δ_1 before δ_2, i.e., candidates with a subset of faults can be generated earlier. Since δ_2 need not be considered until after δ_1 has

[1]This assumption is for ease of presentation only. The algorithm we develop can be applied irrespective of which mode is most likely.

```
function Diagnose(Ω)
   Initialize the set of diagnoses to be empty.
   Initialize the conflict database to be empty.
   Initialize the agenda to contain the most a priori probable
   candidate (usually, all components ok).
   while more diagnoses are needed do
      δ = First candidate in agenda.
      if δ is consistent with the observations Ω then
         Add δ to the set of diagnoses of Ω.
      else
         Update conflict database with new conflict resulting from
         consistency check of δ.
      endif
   Remove δ from agenda.
   Generate successors of δ and add to the agenda.
   endwhile
   return the set of diagnoses.
end Diagnose
```

Figure 3.2: Focused diagnosis algorithm (Independent faults).

been processed, δ_2 can be thought of as an eventual successor of δ_1. This means that the *immediate successors* of δ_1 are those candidates which add exactly one fault mode to the fault modes of δ_1, i.e., let δ_2 be an immediate successor of δ_1 if and only if $faulty(\delta_1) \subset faulty(\delta_2)$ and $|faulty(\delta_1)| = |faulty(\delta_2)| - 1$. The immediate successors of δ_1 can be generated by replacing an **ok** mode in δ_1 by a corresponding fault mode. In particular, let *new-faults*(δ_1) be defined as follows:

$$new\text{-}faults(\delta_1) = \{m_i : m_i \text{ is a fault mode of } \mathcal{C}_i \text{ and } [M_i = \mathbf{ok}] \in \delta_1\}$$

One can see that δ_2 is an immediate successor of δ_1 if and only if $faulty(\delta_2) = faulty(\delta_1) \cup \{M_i = m_i\}$ for some $m_i \in new\text{-}faults(\delta_1)$. While this provides the basic successor function for the best first search, two additional efficiencies dramatically speed up the search.

First, when δ_1 is a diagnosis (i.e., δ_1 is consistent with the observations), each of its immediate successors is a potential diagnosis. However, when δ_1 is *not* a diagnosis (i.e., δ_1 is inconsistent with the observation), some of its immediate successors are not potential diagnoses, and hence need not be considered (i.e., need not be added to the agenda). To see this, consider the following. If δ_1 has been shown to be inconsistent, then the algorithm of Fig 3.2 ensures that there is at least one conflict in the conflict database that is a subset of δ_1. Let N be any one such conflict. Now, if δ_2 is an immediate successor of δ_1 such that $N \subseteq \delta_2$, then δ_2 is not a diagnosis. Furthermore, say δ_2' is a successor of δ_2, such that δ_2' *is* a diagnosis (and hence $N \not\subseteq \delta_2'$). In that case, there must be some immediate successor δ_1' of δ_1 such that $N \not\subseteq \delta_1'$ and δ_2' is a successor of δ_1'. This means that there is *no need* to consider δ_2 in generating the immediate successors of δ_1.

Given the conflict N as above, the successors of δ_1 that are not supersets of N are easy to generate. In particular, let *modes-that-negate*(N) be the set of faulty component modes whose corresponding **ok** modes appear in N, i.e.:

$$\textit{modes-that-negate}(N) = \{m_i : m_i \text{ is a fault mode of } \mathcal{C}_i \text{ and } [M_i = \mathbf{ok}] \in N\}$$

One can see that δ_2 is an immediate successor of δ_1 that is not a superset of N if and only if *faulty*(δ_2) = *faulty*$(\delta_1) \cup \{M_i = m_i\}$, for some $m_i \in$ *modes-that-negate*(N). The number of such immediate successors are usually significantly less than the total number of immediate successors. This pruning of the search space dramatically speeds up candidate generation.

Second, consider the successors of δ_1 constructed from the fault modes in the set *modes-that-negate*(N) (or from the fault modes in the set *new-faults*(δ_1) if δ_1 is a diagnosis). Let δ_2 be a successor of δ_1 constructed by adding fault mode $M_i = m_i$ and removing $M_i = \mathbf{ok}$ from δ_1. The probability of δ_2 is less than the probability of δ_1 by the fraction $P(M_i = m_i)/P(M_i = \mathbf{ok})$. Hence, the most likely successor of δ_1 is the one resulting from adding the fault mode m_i with the largest value of $P(M_i = m_i)/P(M_i = \mathbf{ok})$. Furthermore, the remaining successors of δ_1 need not be considered until *after* the most likely successor has been processed. This leads to the following procedure.

With each conflict N, we sort the fault modes m_i in *modes-that-negate*(N) in decreasing order of $P(M_i = m_i)/P(M_i = \mathbf{ok})$. When generating the successors of δ_1 using N, we only generate the most likely successor, δ_2, using the first fault mode in *modes-that-negate*(N), and leave a pointer to the rest of *modes-that-negate*(N). When δ_2 is finally processed during the best first search, we not only generate its most likely successor (and leave a pointer to the rest), but we also generate the next most likely successor of δ_1 and move further down the rest of *modes-that-negate*(N). The net effect of this strategy is that whenever the current best candidate is popped off the agenda, at most two new candidates are added to the agenda, i.e., at any time the size of the agenda is bounded by the total number of candidates checked for consistency thus far.

Analyzing the worst-case running time to generate an additional candidate is straightforward. On each call, the algorithm has only two major parts. The first part is to sort the list *modes-that-negate*(N) (or *new-faults*(δ), as the case may be). The length of this list is bounded by M, the total number of component modes (though this list is often much smaller than M). If n is the number of components in the system, we note that $M = O(n)$. This sorting operation can thus be carried out in $O(n \log n)$.

The second major operation on each call of the algorithm is to insert two items into the agenda. As we noted above, the length of the agenda is bounded by the number of candidates generated thus far. Say k candidates have been generated so far. The agenda can be implemented as a balanced binary tree, so that insertion takes $O(\log k)$. Hence, the next candidate can be found in time $O(n \log n + \log k)$.

3.2.2 Evaluation

We implemented the diagnostic algorithms described above in Common Lisp. We evaluated their performance on ten benchmark digital circuits from a standard test suite [Brglez and Fujiwara, 1985]. Each component in these circuits was assumed to be in one of four modes: the **ok** mode representing the normal functioning of the component, the **sa1** and **sa0** modes representing fault modes in which the component output is stuck at one or stuck at zero, respectively, and the **unknown** mode with an empty model. Models of component modes were encoded as propositional clauses. Hence, rather than using forward simulation, the Davis-Putnam procedure for checking satisfiability of proprositional databases was used for consistency checking. Component faults were assumed to be independent, with the prior probabilities being: **ok** 0.8, **sa1** and **sa0** 0.099 each, and **unknown** .002.

We ran two sets of experiments. The first set used the unfocused candidate generation algorithm, while the second used the focused candidate generation algorithm. Each set of experiments consisted of 20 experiments per circuit. Each experiment was set up as follows. A random fault was introduced into the device, and an input vector sensitive to this fault was randomly generated. The output was generated by applying the input vector to the faulty circuit. Diagnosis terminated when the leading ten diagnoses were uncovered, or when the next candidate was 100 times more unlikely than the leading diagnosis. Furthermore, diagnosis with the unfocused candidate generator was terminated after about 300 seconds.[2]

Table 3.1 shows the results of our experiments. The first column shows the names of the circuits, the second shows the number of components in each circuit. The third, fourth and fifth columns relate to the unfocused candidate generator, while the sixth and seventh columns relate to the focused candidate generator. The numbers in these columns are averages over the 20 experiments.

The third column shows the number of consistency checks made by the unfocused algorithm. For this algorithm, note that the number of consistency checks is the same as the number of candidates generated. The fourth column shows the average number

[2]Our implementation is unable to cut off search at exactly 300 seconds.

Device	Numb. of components	Unfocused generator Consistency Checks	Unfocused generator Numb. of Diagnoses	Unfocused generator Time (secs)	Focused generator Consistency Checks	Focused generator Time (secs)
c17	6	52	10.0	0.4	18	0.1
c432	160	1550	9.0	95.4	58	4.7
c499	202	2119	7.6	168.5	43	4.5
c880	383	716	9.0	166.2	36	4.0
c1355	546	320	8.2	165.8	52	12.3
c1908	880	298	3.8	280.5	64	22.8
c2670	1193	210	5.0	307.0	93	28.8
c3540	1669	150	0.9	364.4	140	113.3
c5315	2307	92	0.6	376.2	84	61.2
c7552	3512	408	3.0	377.6	71	61.5

Table 3.1: Evaluation of algorithms (independent component failures).

of diagnoses before the 300 second limit expired. The fifth column shows the average run time in seconds on a Sun Sparc 2 workstation.

For the focused generator, the sixth column shows the number of consistency checks made. This algorithm was always able to terminate within the 300 second limit. The seventh column shows the average run time in seconds on a Sparc 2.

The results show that the unfocused candidate generator works well for small devices like c17, and works reasonably well for medium sized devices like c432 through c1355. However, for larger devices, the number of candidates that need to be checked becomes very large, and the top ten diagnoses cannot be generated within 5 minutes. In fact, in examples like c5315, generating even one diagnosis is difficult. On the other hand, the focused candidate generator is very efficient, being able to complete the task on all the devices in an average of under 2 minutes. This shows that the judicious use of conflicts can dramatically speed up model-based diagnosis. It also shows that caching of inference (as node labels) does not appear to be important. In particular, these results are comparable to the results from the very best ATMS-based implementations. Note that, the performance on the largest examples shows that no performance penalty is incurred due to memory limitations.

In summary, generate-and-test using the focused candidate generation algorithm is a fast and simple model-based diagnosis algorithm. The unfocused candidate generation algorithm can be used effectively for small to medium sized examples, and is particularly useful in cases where finding conflicts is difficult (so that the focused algorithm provides no benefit).

3.3 Dependent faults

We now turn our attention to the situation where the faults are not independent. Our goal is to construct a algorithm with properties similar to the one described above. We will assume that when faults are not independent, the dependencies between the set of mode variables of the system are specified to us in the form of a Bayesian network. We begin by assuming that the Bayesian network specified for the fault interactions has a particular topology, viz., that it is singly connected. A singly connected Bayesian network has at most one path from any node to any other node. Say there are k arcs going out of a node n. Say we delete every arc. This separates the Bayesian network into at least $k+1$ mutually disconnected segments. A singly connected Bayesian network has the following important property: If we observe the value of node n in a singly connected Bayesian network, the variables in each of these segments is rendered independent of variables in any other segment. This property can be exploited to develop a simple algorithm to compute a list of instantiations of the Bayesian network in decreasing order of probability. An instantiation of the Bayesian network is an assignment of state to every node. It thus corresponds exactly to our notion of candidate.

3.3.1 Message Passing

We now develop an algorithm to generate candidates in decreasing order of probability. The algorithm is developed in the same spirit as [Pearl, 1987]. [Pearl, 1987] describes a message passing algorithm for computing the most likely instance of a singly connected Bayesian network. Our algorithm can be considered as a generalization of this algorithm. In addition to computing the most likely instance, it successively computes (on request) the next most likely instance, the third most likely instance, and so on.

Our message passing algorithm will operate in the following fashion: An arbitrary node is chosen in the Bayesian network as the starting node. The starting node requests all its neighbors for messages pertaining to computing the list of most probable instances in decreasing order. These messages pertain to instantiations of part of the network reachable through the neighbor. When the starting node has received the messages, it combines them in some way and returns the list of full instantiations of the Bayesian network. When a neighbor is requested to give a message, it recursively requests each of its neighbors (except for the original requesting node) for a message. It then combines these messages in some way and passes them on to the requesting node. As we will see later, the independence properties of the singly connected network make such a message passing algorithm possible.

The singly connected network can be viewed as a tree if one ignores the direction

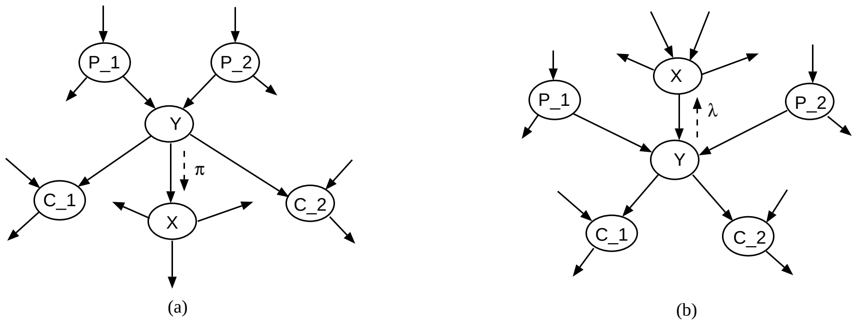


Figure 3.3: Message passing: the two possible cases.

of the arcs. We note that the algorithm we develop makes a single traversal through this tree. That is, messages are sent from leaf nodes to parents and from the parents to the ancestors until they all converge in some (arbitrarily chosen) root node.

The description of the message passing algorithm thus reduces to the description of the operations at a single node. The description needs to explain what the messages are and how the messages coming from neighbors are combined and sent to the requesting node.

Initially, we will describe how we can compute the *entire* list of candidates in decreasing order of probability. Later we will show how to modify this algorithm to make it compute one candidate at a time (on demand).

3.3.2 What are the messages?

We now define the messages sent between the nodes. We will describe how the message are actually computed later. We first define some terminology. Consider two nodes A and B that are connected by an arc in the Bayesian network. We will use $R_{A||B}$ to refer to the set of all the nodes in the subnetwork containing A when the arc connecting A and B is disconnected. Note that the arc between A and B could be in either direction.

Say node X requests node Y for a message. Say Y is a *parent* of X in the Bayesian network. We will refer to the message that Y sends X as $\pi^l_{Y \to X}$.[3] The direction of the arrow in the subscript refers to the direction of the message (from Y to X) and not the direction of the arc in the Bayesian network. The superscript l (for "list") reminds us that the message is being used to compute the ordered list of all instances of the Bayesian network.

$\pi^l_{Y \to X}$ is a vector indexed by the states y of Y. The location $\pi^l_{Y \to X}[Y = y]$ contains

[3]We follow Pearl in choosing π and λ as the message names.

a list of all instantiations i of the nodes in $R_{Y||X}$ such that Y has state y in i. The list elements are arranged in decreasing order of probability. The probability is stored with each list element. Note that the elements of $R_{Y||X}$ form a full Bayesian network in themselves. Hence, it is possible to compute the probability of each instantiation of $R_{Y||X}$ without regard to X or any node reachable through X.

Now consider the case where Y is a *child* of the requesting node X in the Bayesian network. We will refer to the message that Y sends X as $\lambda^l_{Y\to X}$. The $\lambda^l_{Y\to X}$ message is indexed by the states x of X. $\lambda^l_{Y\to X}[X = x]$ contains a list of instantiations r of $R_{Y||X}$ sorted by decreasing order of the probability $P(R_{Y||X} = r|X = x)$. This is the conditional probability of the instance r given that X is in the state x. The probability is stored with each list element.

We note that observing the value of X makes the nodes in $R_{Y||X}$ independent of all the other nodes reachable through $X \to Y$. Hence, given a state x of X and an instantiation r of $R_{Y||X}$, it is possible to compute the probability $P(R_{Y||X} = r|X = x)$ locally within the subnetwork formed by the nodes in $R_{Y||X}$.

3.3.3 Computing the messages

Say node X has requested node Y for a message. We describe the computations that Y performs in computing the message.

Y first recursively asks for messages from all its neighbors (except for X). After they are available, it computes the message meant for X. We note that there are two cases that need to be described. In the first case, Y is a parent of X. In the second case, Y is a child of X.

Y is a parent of X

We consider an example of the first case (Fig 3.3(a)). Consider an instance $r_{P_1||Y}$ of the set $R_{P_1||Y}$. Suppose that $P_1 = p_1$ in $r_{P_1||Y}$. Similarly, consider an instance $r_{P_2||Y}$ of $R_{P_2||Y}$. Suppose that $P_2 = p_2$ in $r_{P_2||Y}$. Further, let $r_{C_1||Y}$ and $r_{C_2||Y}$ be any two instances of $R_{C_1||Y}$ and $R_{C_2||Y}$.

We note that if we append all these instances together and add in a choice of state for Y, we get a full instance $r_{Y||X}$ of $R_{Y||X}$. Say we choose the state y of Y as the state added in to get $r_{Y||X}$. We note from the independence properties of a singly connected Bayesian network that the following is true:

$$\begin{aligned} P(r_{Y||X}) &= P(Y = y|P_1 = p_1, P_2 = p_2) \times \\ &\quad P(r_{P_1||Y})P(r_{P_2||Y}) \times \\ &\quad P(r_{C_1||Y}|Y = y)P(r_{C_2||Y}|Y = y) \end{aligned} \tag{3.2}$$

Note that $r_{Y||X}$ is an element of $\pi^l_{Y\to X}[Y = y]$. Similarly $r_{P_1||Y}$ is an element of

$\pi^l_{P_1 \to Y}[P_1 = p_1]$, and $r_{P_2||Y}$ is an element of $\pi^l_{P_2 \to Y}[P_2 = p_2]$. In addition, $r_{C_1||Y}$ is an element of $\lambda^l_{C_1 \to Y}[Y = y]$ and $r_{C_2||Y}$ is an element of $\lambda^l_{C_2 \to Y}[Y = y]$. The probabilities required in Equation 3.2 are exactly those stored with these elements (see previous section).

We now define some terminology. Given two ordered lists of instances L_1 and L_2 ordered in decreasing order of probability, let $L_1 \otimes L_2$ be the *ordered* list composed of all possible combinations of the instances where one element is chosen from L_1 and one element is chosen from L_2. The probability of the combination is the product of the stored probabilities of the components. For now, assume that $\otimes$ is realized by a naive algorithm which forms all combinations and then sorts them.

Given an ordered list of instances L and a number k, let $k \odot L$ be the list where the probability of every instance in L is multiplied by k. Finally, given a list of ordered instance lists LL, let $Merge(LL)$ be the list formed by merging the constituent lists of LL into a single ordered list. Each of the constituent lists of LL is assumed to contain instances of the same set of variables.

The discussion above leads directly to the algorithm shown in Fig 3.4 for computing $\pi^l_{Y \to X}$. This algorithm computes $\pi^l_{Y \to X}$ from the messages coming to it from P_1, P_2, C_1 and C_2. In essence, for each state y of Y, the algorithm is generating every element of $\pi^l_{Y \to X}[Y = y]$ and ensuring that the elements are put together into a list in decreasing order of probability. We note that the algorithm is adapted easily to the case where Y has an arbitrary number of parents and an arbitrary number of children.

Y is a child of X

We now consider the case where Y is a child of X and describe the message computations in Y. The situation is shown in Fig 3.3(b). The same argument used in the first case leads to the algorithm shown in Fig 3.5. [4]

3.3.4 Putting it all together

The algorithm **Compute-ordered-instances** for computing the ordered list of instances of the entire Bayesian network (Fig 3.6) follows directly from **Compute**-$\pi^l_{Y \to X}$ and **Compute**-$\lambda^l_{Y \to X}$.

We choose an arbitrary node R of the Bayesian network as the root node. We add a dummy node D as a parent of R. D has only one state $\hat{d}$. Hence automatically, $P(D = \hat{d}) = 1$. Say the parents of R were the set $\mathbf{S}_R$ before adding D. Let $\mathbf{s}_R$ be a joint state of $\mathbf{S}_R$. Say the conditional probability was defined by the table

[4] At the expense of clarity, this algorithm can be improved by computing and saving L^λ_y for all y before entering the main loop.

begin Compute-$\pi^l_{Y \to X}$(Y, X)
For all states y of Y:

1. $L^\lambda_y = \lambda^l_{C_1 \to Y}[Y = y] \otimes \lambda^l_{C_2 \to Y}[Y = y]$
2. Initialize LL to the empty list. LL is a list of lists.
3. For all combinations $< p_1, p_2 >$ of the states of P_1 and P_2:
   (a) Let $k = P(Y = y | P_1 = p_1, P_2 = p_2)$.
   (b) $L = k \odot \pi^l_{P_1 \to Y}[P_1 = p_1] \otimes \pi^l_{P_2 \to Y}[P_2 = p_2] \otimes L^\lambda_y$
   (c) Cons L onto LL.
4. $\pi^l_{Y \to X}[Y = y] = Merge(LL)$

end Compute-$\pi^l_{Y \to X}$

Figure 3.4: Algorithm for the case where Y is a parent of X.

begin Compute-$\lambda^l_{Y \to X}(Y, X)$
For all states x of X:

1. Initialize LL to the empty list. LL is a list of lists.
2. For all states y of Y:
   (a) $L^\lambda_y = \lambda^l_{C_1 \to Y}[Y = y] \otimes \lambda^l_{C_2 \to Y}[Y = y]$
   (b) For all combinations $< p_1, p_2 >$ of the states of P_1 and P_2:
      i. Let $k = P(Y = y | P_1 = p_1, P_2 = p_2, X = x)$.
      ii. $L = k \odot \pi^l_{P_1 \to Y}[P_1 = p_1] \otimes \pi^l_{P_2 \to Y}[P_2 = p_2] \otimes L^\lambda_y$
      iii. Cons L onto LL.
3. $\lambda^l_{Y \to X}[X = x] = Merge(LL)$

end Compute-$\lambda^l_{Y \to X}$

Figure 3.5: Algorithm for the case where Y is a child of X.

begin Compute-ordered-instances(B)

1. Choose an arbitrary node R of the Bayesian network B and add a dummy node D as parent.
2. **Compute-message**(R, D)
3. Return $\lambda^l_{R \to D}[D = \hat{d}]$.

end Compute-ordered-instances

begin Compute-message(Y, X)

1. For all neighbors N of requested node Y except the requesting node X do:

   Compute-message(N, Y)

2. If Y is a parent of X **then**:

   Compute-$\pi^l_{Y \to X}(Y, X)$

   else

   Compute-$\lambda^l_{Y \to X}(Y, X)$

end Compute-message

Figure 3.6: Algorithm for computing ordered instance list.

$P_{old}(R|\mathbf{S}_R)$. The conditional probability distribution of R after D's addition is set to be $P_{new}(R = r|\mathbf{S}_R = \mathbf{s}_R, D = \hat{d}) = P_{old}(R = r|\mathbf{S}_R = \mathbf{s}_R)$. We see that, effectively, R is independent of D. We see that if D requests R for a message, then $\lambda^l_{R \to D}[D = \hat{d}]$ contains exactly the list of ordered instances of the entire network.

We reiterate once again that the algorithm computes the *full* list of ordered instances. Hence, though it takes full advantage of the independence properties of the network to decompose the problem, it's run time is inherently exponential since the number of instances is exponential.

3.3.5 Computing one instance at a time

We now modify the algorithm to return one instance at a time from the ordered list. The next instance is computed only when demanded. The idea is very similar to the idea in the case of independent faults, i.e., we make the computation lazy.

Specifically, all that is required is to make the computation of the list operations $\otimes$, $\odot$ and $Merge$ lazy. The modified **Compute-ordered-instances** returns a *lazy list*. Initially, a lazy list contains only the first element of the list. The rest of the elements are stored as a delayed computation in the list's data structure. Each time we demand the next element, the delayed computation is called. It performs only the necessary computations to compute the next element. This element is added to the end of the list. The computation then delays itself again.

Note that we do not make the evaluation of the loop that creates list LL in **Compute**-$\pi^l_{Y\rightarrow X}(Y,X)$ and **Compute**-$\lambda^l_{Y\rightarrow X}(Y,X)$ lazy . Hence, all the list elements of the call to $Merge$ are computed initially. However, each element is a lazy list. This observation implies that the delayed computations will perform only the list operations $\otimes$, $\odot$ and $Merge$. The call graph of the message passing algorithm is stored implicitly in the delayed computation.

Making the computation of $\odot$ and $Merge$ lazy is straightforward. We will refer to the lazy version of $\odot$ as $\odot_z$. Given a constant factor k and a lazy list L_z as arguments, $\odot_z$ multiplies k into the probability of the first element of L_z and returns it. It then wakes up the delayed computation in L_z. This results in the second element of L_z being generated. It then goes to sleep. On the next call, it multiplies the constant factor into the second element and returns it. It then generates the third element of L_z and goes to sleep and so on. On each call, it performs $O(1)$ computations (not counting the computation performed due to the awakening of L_z's delayed computation).

Let the lazy version of $Merge$ be $Merge_z$. On each call, $Merge_z$ goes through its argument LL looking at the probability of the current element of every delayed list in LL. It returns the element C_{max} with maximum probability. Let L_{max} be the list from which C_{max} came. C_{max} is popped off L_{max}. $Merge_z$ now wakes up the computation of L_{max} till the next element of L_{max} is generated and this is made the current element of L_{max}. It then goes to sleep. On each call, $Merge_z$ performs $O(Length(LL))$ comparisons (not counting the computation performed due to the awakening of L_{max}'s delayed computation).

An efficient way of making $\otimes$ lazy

The operation $\otimes$ takes two *ordered* lists L_I and L_J as arguments and returns an ordered list where each element is a compound element composed of one element from L_I and one element of L_J. The numerical value associated with the compound element is the product of the numerical values associated with the constituents. The list which is returned is ordered by this numerical value.

Consider the example shown in Fig 3.7. In this example, the elements of the list are the numerical values themselves. Each location in the matrix is the product of the appropriate elements of L_I and L_J.

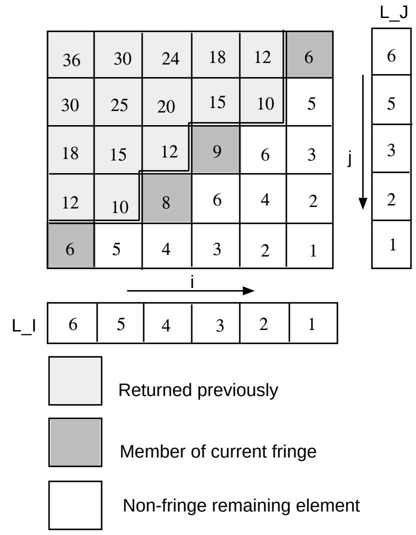


Figure 3.7: Making the $\otimes$ operation lazy: The fringe in $\otimes_z$.

An element $a(i_2, j_2)$ in the matrix is *dominated* by an element $a(i_1, j_1)$ if it is necessarily less than or equal to $a(i_1, j_1)$ regardless of the actual values in the ordered lists L_I and L_J. We see directly that $a(i_2, j_2)$ is dominated by $a(i_1, j_1)$ iff $i_1 \leq i_2$ and $j_1 \leq j_2$. We will call the element $a(i+1, j)$ the *dominated neighbor along dimension* i of $a(i, j)$. That is, an element's dominated neighbor along a dimension is the element immediately "below" it along that dimension.

We will now describe a delayed version of $\otimes$ which we will call $\otimes_z$. Say $\otimes_z$ has been called a few times. Every time it was called, it returned the next largest element out of the matrix. The remaining elements are those elements of the matrix that have not yet been returned during previous calls to $\otimes_z$. Say $\otimes_z$ maintains a set in its internal state. The set F consists only of those of the remaining elements that are *not dominated* by any of the other remaining elements. We will call the set F the *fringe* (See Fig 3.7).

We see that each time $\otimes_z$ is called, it only has to return the maximum element C_{max} of the fringe F and then update the fringe. The fringe update is easily accomplished by the procedure shown in Fig 3.8. The procedure computes and returns C_{max} and updates the fringe. Note that this procedure does not explicitly generate the matrix. It simply retains the matrix indices with each element of F to perform domination tests.

We have assumed in the above discussion that $\otimes_z$ takes only two arguments. However, we see that the identical discussion applies if there are n lists given as

begin Update-fringe

1. Choose the max element C_{max} of the fringe F and delete it from F.
2. Along each dimension K do:
   Let C_K be the dominated neighbor of C_{max} along K. If C_K is not dominated by any element in F then:
   (a) Compute C_K (i.e., actually multiple the probabilities and create the compound element). This computation might require the computation of the next yet uncomputed element of the list L_K. If so, awaken the computation of L_K so that this element is available.
   (b) Add C_K to the set F.
3. Return C_{max}.

end Update-fringe

Figure 3.8: Updating the fringe in $\otimes_z$.

arguments. Instead of a two dimensional matrix, we have an n dimensional matrix. The **Update-fringe** procedure applies even when there are n arguments. A general implementation that can handle any number of argument lists can be used to compute $(L_1 \otimes_z L_2 \otimes_z \ldots \otimes_z L_n)$ as $\otimes_z(L_1, L_2, \ldots, L_n)$. Such an implementation is immediately applicable in **Compute**-$\pi^l_{Y\rightarrow X}(Y, X)$ and **Compute**-$\lambda^l_{Y\rightarrow X}(Y, X)$.

Let L_{result} be the entire ordered list which would result if all the elements returned successively by $\otimes_z(L_1, L_2, \ldots, L_n)$ were computed (by repeated calls to the delayed computation). Consider the situation where the first k elements of L_{result} have been computed and the rest are yet uncomputed. We see that every time we update the fringe, we add at most n elements to it. At the start of the computation, the fringe consists of exactly 1 element (viz, the first element of L_{result}). Hence after k elements of L_{result} have been computed, the size of the fringe is at most nk. Examining **Update-fringe**, we see that when computing the $k+1$st element, we need $O(nk)$ comparisons to determine C_{max}. In addition, we need to make $O(nk)$ domination tests along each of the n dimensions in Step 2. Each domination test can be implemented as a single comparison. Hence the the $k+1$st element of L_{result} can be computed with $O(n^2k)$ comparisons (not counting any operations resulting from waking up computations in any argument list). We note that this is a loose bound. In practice, as we shall see later, $\otimes_z$ does much better.

3.3.6 Complexity of the full lazy algorithm

We now compute an upper bound on the complexity of the full lazy algorithm, i.e., the complexity of generating the kth most probable instance of the Bayesian network.

Consider a node Y which is computing a message to be sent to node X using **Compute-message**(Y, X). Let $Size(Y)$ be the size of the conditional probability table of Y. That is, Y is the product of the cardinalities of Y and each of its parents. Let $Degree(Y)$ be the number of neighbors of Y, i.e., the sum of the number of parents and number of children.

We examine the complexity of computing the message where each message element is a lazy list. Specifically, we look at the total complexity of computing the next element in each of these lazy lists. We consider only the computations performed within Y, i.e., we exclude the comparisons performed in recursive calls to **Compute-message**.

Examining **Compute**-$\pi^l_{Y\to X}(Y, X)$ and **Compute**-$\lambda^l_{Y\to X}(Y, X)$, we note that the $Merge_z$ and $\odot_z$ operations together perform $O(Size(Y))$ comparisons[5].

We now examine the number of comparisons performed by the $\otimes_z$ operation. Say we have generated the first k elements of every message element list and are looking to generate the $k + 1$st element of each of these lists. We see that the number of comparisons performed by $\otimes_z$ is bounded by $O(Size(Y)Degree(Y)^2k)$.

Given a Bayesian network B, let $Size(B) = \Sigma_{Y\in B} Size(Y)$. We see that $Size(B)$ measures the amount of information required to specify the network. In addition, define $MaxDegree(B) = \max_{Y\in B} Degree(Y)$.

Say we have generated the k most probable instances of the Bayesian network and are now computing the $k+1$th most probable instance. We see that $\odot_z$ and $Merge_z$ together perform $O(Size(B))$ comparisons. $\otimes_z$ performs $O(Size(B)MaxDegree(B)^2$ $k)$ comparisons.

Thus, the overall complexity of generating the $k + 1$st most probable instance is $O(Size(B)MaxDegree(B)^2k)$. We note that this is a loose upper bound. There are two reasons. The first is that the bound that we computed earlier on $\otimes_z$ is loose. The second is that we are assuming that every delayed list will be forced to compute its next element in the process of computing the next most probable instance of the entire network. This need not be true. In practice, the algorithm runs much faster (as described later).

We note that when $k = 1$ the algorithm computes the most probable instance of the network. This is exactly what is computed by [Pearl, 1987].

[5] Actually, the operation performed by the $\odot_z$ operation is a multiplication. We count one multiplication as equivalent to one comparison in this analysis.

3.3.7 Multiply-connected networks

The algorithm we have presented so far can handle only singly connected Bayesian networks. When Bayesian networks are not singly connected, there is a general scheme called conditioning which can be used to adapt the singly connected algorithms to handle multiply connected networks [Pearl, 1988].

Conditioning chooses a set of nodes in the Bayesian network such that observing the values of those nodes leaves the resulting network singly connected. This is in accordance with the independence semantics of Bayesian networks. The set of conditioning variables is called the *cutset*. A computation is performed for every possible *joint instance* of the cutset using the singly connected algorithm and these computations are then combined. A joint instance consists of a choice of state for each node in the cutset. In general, domains suitable for modeling with Bayesian networks have a large number of independences and so typically, the size of the cutset is small.

Our algorithm can be adapted directly to handle multiply connected networks using conditioning. For every joint instance c of the cutset, we compute an ordered list of instances L_c of the network. Each element e_c of L_c will be a full network instance. Each element e_c will necessarily have each of the conditioning variables in the state specified by c. The probability stored with e_c will be $P(e_c|c)$. L_c can be computed with the algorithm we have developed above. For each list L_c, we then compute $L_c^{'} = P(c) \odot L_c$. Here $P(c)$ is the prior probability of the cutset instance c. The lists $L_c^{'}$ (one for each cutset instance c) are then merged to give the list of all instances in decreasing order of probability.

Let the cutset of Bayesian network B be $\mathbf{C}_B$. Let $Size(\mathbf{C}_B)$ be the size of the joint state space of the variables in the cutset. From the discussion above and in the previous subsection, we see that a loose upper bound for generating the $k+1$st most probable instance of the Bayesian network is $O(Size(\mathbf{C}_B)Size(B)MaxDegree(B)^2k)$.

3.3.8 Evaluation

The algorithm **Compute-ordered-messages** was implemented in Common Lisp. The algorithm is implemented on top of IDEAL, a software package for Bayesian network inference [Srinivas and Breese, 1990]. Since there are no standard example suites of systems with dependent faults, we report just the running time of the candidate generator rather than using the structure of the experiments of Section 3.2.2.

Run times for **Compute-ordered-instances** are shown in Table 3.2. The times shown are for two randomly generated singly connected belief networks. Given the number of nodes n, we generated a singly connected Bayesian network with n nodes. The maximum number of neighbors for any node in the network (i.e., the $MaxDegree$ of the network) and the maximum number of states for each node are also specified before the random Bayesian network is generated. The distribution for the belief

Time to generate next candidate		
Number of components	300	600
Max. numb. of modes per comp.	5	6
MaxDegree	5	6
Number of cands. generated	600	600
Setup time	11 secs	2 mins
Max. time	34 msec	50 msec
Min. time	0 msec	0 msec
Avg. time	7.8 msec	11.0 msec

Run times for **Compute-ordered-instances**

Table 3.2: Evaluation of algorithm (dependent component failures).

network is set randomly.

We see that we can compute each candidate in the order of tens of milliseconds on the average for medium sized systems (when the number of components is in the order of hundreds). The variability of the distribution is also quite small. We have also noted that the time to compute candidates varies fairly uniformly as the candidates are generated. In other words, there is no trend towards increase or decrease in the average time as the number of candidates generated increases. We note here that if the algorithm performed in accordance with its worst case analysis, there should be a linear increase in run time. In practice, the algorithm does much better.

We note that the time to initialize the algorithm data structures is substantial relative to the time to generate candidates. The initialization, of course, is a one time cost and need not be repeated for each diagnosis run.

3.4 Discussion

In this chapter, we have approached the problem of computing most probable diagnoses tractably from a generate-and-test view point. The crucial component of our diagnosis method is an efficient candidate generator. We have demonstrated how to perform candidate generation efficiently both when faults are independent and when they are dependent.

Focusing of diagnosis systems has been a subject of considerable interest in model-based diagnosis [Collins and DeCoste, 1991; Dressler and Farquhar, 1990; Forbus and de Kleer, 1988]. The HTMS [de Kleer, 1994], which is the current state of the art, achieves impressive performance results. Our results are in the same spirit but differ in an important respect. Our generate-and-test approach leads to extremely simple algorithm specifications that are not tied to a particular implementation method of the diagnosis system (such as an ATMS). As a result, our algorithms are general purpose algorithms. They come with provable properties, both in complexity and correctness.

One can view our unfocused candidate generation algorithm as performing a best first search in the space of all candidates. In effect, we have taken advantage of properties of the domain to reduce best first search to a direct algorithm. As discussed before, our work in generating candidates in the case of dependent faults is a direct generalization of Pearl's method for computing the most probable instance of a Bayesian network [Pearl, 1988]. The algorithm **Compute-ordered-instances** should also be of interest to those applying Bayesian networks in settings other than model-based diagnosis.

Chapter 4

Specifying component failure probabilities

Probabilistic model-based diagnosis requires failure probabilities for each component in the system model. Specification of these probabilities is difficult—an expert cannot often reliably give such numerical estimates. A primary reason for this problem is that there is an implicit notion of time in these prior probabilities.

Consider a component A which has been up and running for a week. Say the prior probability of failure assessed at this time is p_{week}. Now consider the same component when it has been running for a month. Say the failure prior assessed at this time is p_{month}. Intuitively, we should have $p_{month} > p_{week}$, i.e., the longer the component has been up, the higher the chance that some event that caused it to fail has occurred. For example, consider the prior probability that the timing chain in a car has failed. The prior probability of timing chain failure typically depends on the age of the car. If the car has low mileage (say 10K miles) the prior probability that the timing chain has broken is very low. However if the car has high mileage (say 90K miles), the prior probability of the timing chain being broken rises to a high value.

A related problem occurs when doing diagnosis with multiple observations, where each observation occurs at a different time. When performing diagnosis with multiple observations, we need some model of the persistence of the state of the system between the observations. Let $M_i[t]$ be a state variable representing the state of the component $\mathcal{C}_i$ at time t. Let $\Delta[t]$ be a compound variable which represents the joint state of all the components in the system. That is, $\Delta[t]$ is the vector $\langle M_1[t], M_2[t], \ldots, M_n[t]\rangle$. We will use $\delta[t]$ to denote a state of $\Delta[t]$. Note that each state of $\Delta[t]$ is simply a *candidate* (see Section 3.1). We will refer to $\Delta[t]$ as the *candidate state variable* at time t.

Consider a situation where we have an observation $\Omega[t_1]$ at time t_1 and an observation $\Omega[t_2]$ at time t_2. Say we want to perform diagnosis at time t_2. The goal of diagnosis is to compute the distribution $P(\Delta[t_2] \mid \Omega[t_1], \Omega[t_2])$. This distribution can be computed as:

$$\begin{aligned} & P(\Delta[t_2] = \delta[t_2] \mid \Omega[t_1], \Omega[t_2]) \propto \\ & \quad P(\Delta[t_2] = \delta[t_2], \Omega[t_1], \Omega[t_2]) \\ \propto & \quad P(\Omega[t_2] \mid \Delta[t_2] = \delta[t_2], \Omega[t_1]) \times P(\Delta[t_2] = \delta[t_2] \mid \Omega[t_1]) \\ = & \quad P(\Omega[t_2] \mid \Delta[t_2] = \delta[t_2]) \times P(\Delta[t_2] = \delta[t_2] \mid \Omega[t_1]) \end{aligned} \tag{4.1}$$

The last step follows because knowing the state $\delta[t_2]$ of the system renders the observation $\Omega[t_2]$ independent of the history. The probability $P(\Omega[t_2] \mid \Delta[t_2] = \delta[t_2])$ can be calculated from the diagnostic Bayesian network. In the above equation, we also need the distribution $P(\Delta[t_2] \mid \Omega[t_1])$, i.e., an estimate of the state at time t_2 given the observation at time t_1. This can be computed as follows:

$$\begin{aligned} & P(\Delta[t_2] = \delta[t_2] \mid \Omega[t_1]) = \\ & \quad \Sigma_{\delta[t_1]} P(\Delta[t_2] = \delta[t_2] \mid \Delta[t_1] = \delta[t_1], \Omega[t_1]) \times P(\Delta[t_1] = \delta[t_1] \mid \Omega[t_1]) \\ = & \quad \Sigma_{\delta[t_1]} P(\Delta[t_2] = \delta[t_2] \mid \Delta[t_1] = \delta[t_1]) \times P(\Delta[t_1] = \delta[t_1] \mid \Omega[t_1]) \end{aligned} \tag{4.2}$$

The last step follows from a Markov assumption that the system state $\delta[t_2]$ is rendered independent of all history up to time t_1 when the state $\delta[t_1]$ at time t_1 is known. In the above equation, $P(\Delta[t_1] \mid \Omega[t_1])$ is the posterior over the state of the system at time t_1. This can be computed easily as:

$$P(\Delta[t_1] = \delta[t_1] \mid \Omega[t_1]) \propto P(\Omega[t_1] \mid \Delta[t_1] = \delta[t_1]) \times P(\Delta[t_1] = \delta[t_1]) \tag{4.3}$$

The first term in the right hand side of the above equation can be computed from the diagnostic Bayesian network. The second term is the prior probability of the candidate. This can be computed as a product from the prior distribution over the mode of the individual components.

We see that the distribution $P(\Delta[t_2] \mid \Delta[t_1])$ in Equation 4.2 is still unspecified. This distribution quantifies the persistence of the state of the system. For example, if the system never changed state spontaneously, then we would have:

$$P(\Delta[t_2] = \delta[t_2] \mid \Delta[t_1] = \delta[t_1]) = \begin{cases} 1 & \text{if } \delta[t_1] = \delta[t_2] \\ 0 & \text{if } \delta[t_1] \neq \delta[t_2] \end{cases}$$

However, in the real world, components do fail on-line. If we had a model of how the components failed when they were on-line, this could be used to compute the distribution $P(\Delta[t_2] \mid \Delta[t_1])$.

We address the problems of specifying failure priors and modeling persistence using techniques from Reliability Theory. Reliability Theory gives us a model that relates the probability of component failure and the age of the component. We use this model to compute a component's prior probability of failure from its uptime and its Mean Time Between Failure (MTBF). The MTBF is a summary measure of the reliability of a device and is often available with manufacturer specifications. We also use the component failure model to compute the conditional distribution over the states of a component at time t_2 given the state at time t_1. This conditional distribution is used to model persistence.

4.1 Reliability models

Reliability Theory offers empirically validated models for the failure process of devices. The failure process of a device relates the probability of failure of the device to time (for example, see [Tsokos and Shimi, 1977]). We now describe one standard model of the failure process.

Consider a device A. It has two states—**ok** and **broken**. It is initially in the **ok** state when it is brought on-line. At some point in time, it transitions into the **broken** state. Once it is broken, it stays broken. It is assumed that the failure of A, i.e., the transition of A from the **ok** to **broken** state, occurs due to random unmodeled events which occur with a probability that is constant over time. For example, an electronic component may fail due to surges in the power supply, and these surges may occur randomly with a uniform probability.

The modeling assumption is interpreted as follows: Given that the device has not failed at time t, the conditional probability that it will fail in the interval $[t, t + dt]$ is proportional to the length of the interval. The probability is thus given by λdt where λ is a proportionality constant. Say we model the failure of A with a continuous real variable X. That is, "$X < t$" denotes the event that A fails in the interval $[0, t]$. Say $F(t)$ is the cumulative distribution of X, i.e., $F(t) = P(0 \leq X < t)$. $F(t)$ is the probability that A fails in the time interval $[0, t]$.

From the modeling assumption, we have:

$$\begin{aligned} P(t \leq X \leq t + dt \mid X \geq t) &= \lambda dt \qquad (4.4) \\ \frac{F(t+dt) - F(t)}{1 - F(t)} &= \lambda dt \\ \frac{dF(t)}{dt} &= \lambda (1 - F(t)) \\ F(t) &= 1 - e^{-\lambda t} \\ & \quad \text{(Since F(0) = 0)} \end{aligned}$$

We see that the probability of A failing before time t is small when t is small and

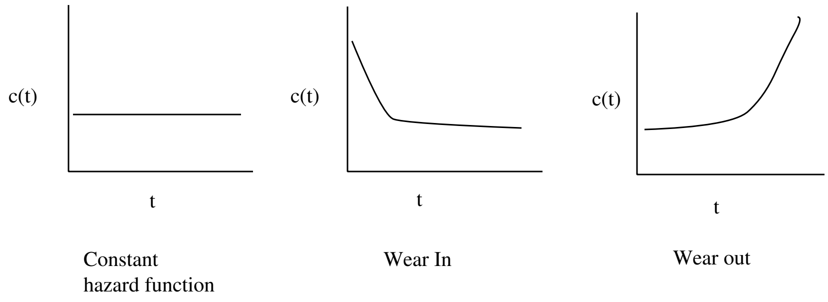


Figure 4.1: Different choices for the hazard function.

converges to 1 as t tends to infinity.

In general, instead of assuming a constant λ in the equation above, we might have a time varying conditional density $c(t)$, called the *hazard function*. This allows us to model various different degradation phenomena in the device. For example, if we want to model a situation where a component has high probability of failure early in its life (called "wear in"), we might choose $c(t)$ to be high for small values of t. If we want to model wear out (and hence, increased chance of failure) of the device as it grows older, we might choose $c(t)$ to be high as t becomes large (See Fig 4.1). The differential equation Equation 4.4 can be solved for the appropriate choice of hazard function to give the corresponding density function.

The expectation of the variable X is called the Mean Time between Failures. In the case of $c(t) = \lambda$, the MTBF can be shown to be $\frac{1}{\lambda}$.

4.1.1 Computing prior probabilities of failure

Using the above model, we can specify more precisely what we mean by the prior distribution over the mode of a component A. The probability that A is broken is a function of time. We will denote this probability at time t by $P(M_A[t] = \textbf{broken})$. Say that the last time we knew that A was certainly not broken was t_{ok}. Usually, t_{ok} is the time that A first went on-line.

We see that $P(M_A[t] = \textbf{broken})$ is the probability that A failed in the interval $[t_{ok}, t]$. Say we assume a constant hazard function. We see that:

$$\begin{aligned} P(M_A = \textbf{broken}) &= P(t_{ok} \leq X \leq t \mid X > t_{ok}) \\ &= \frac{F(t) - F(t_{ok})}{1 - F(t_{ok})} \\ &= 1 - e^{-\lambda(t - t_{ok})} \end{aligned} \tag{4.5}$$

$P(M_i[t_2] \mid M_i[t_1])$	$M_i[t_1] = \mathbf{ok}$	$M_i[t_1] = \mathbf{broken}$
$M_i[t_2] = \mathbf{ok}$	$e^{-\lambda_i(t_2-t_1)}$	0
$M_i[t_2] = \mathbf{broken}$	$1 - e^{-\lambda_i(t_2-t_1)}$	1

Table 4.1: Quantifying the persistence of M_i from t_1 to t_2.

The above equation allows us to directly relate MTBFs to the prior probabilities that we need.

Note that when we assume $c(t) = \lambda$, $P(M_A = \mathbf{broken})$ is only a function of $t - t_{ok}$. That is, all we need to know is the length of the interval from the last time at which we were sure that A was not broken. However, if we assume more complicated hazard functions $c(t)$, we would also need to know when A first went on-line. This is because we will measure t and t_{ok} using that as the start time. Hence, in general, we will find that $P(M_A = \mathbf{broken}) = f(t, t_{ok})$ where f is some function whose form depends on the choice we make for the hazard function $c(t)$.

4.1.2 Modeling persistence

Earlier, we saw that some persistence model was necessary to compute the distribution $P(\Delta[t_2] \mid \Delta[t_1])$. The reliability model described above gives us a method to compute this distribution.

Say the system has n components, $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$. The mode variable corresponding to component $\mathcal{C}_i$ is M_i. We assume that the failure processes of the individual components are independent. Thus, given the state of $\mathcal{C}_i$ at time t_1, we can compute a conditional probability distribution over the state at time t_2 without regard to any of the other components. If we assume a constant hazard function λ_i for $\mathcal{C}_i$, this distribution is given by Table 4.1.

The state $\delta[t]$ of the variable $\Delta[t]$ consists of a state assignment to each of the mode variables M_i. Denoting the state of $M_i[t]$ by $m_i[t]$, we have $\delta[t] = \langle M_1[t] = m_1[t], M_2[t] = m_2[t], \ldots, M_n[t] = m_n[t]\rangle$. From the independence assumption of the failure processes, we have:

$$
\begin{aligned}
& P(\Delta[t_2] = \delta[t_2] \mid \Delta[t_1] = \delta[t_1]) = \\
& \qquad P(M_1[t_2] = m_1[t_2], M_2[t_2] = m_2[t_2], \ldots, M_n[t_2] = m_n[t_2] \mid \\
& \qquad\quad M_1[t_1] = m_1[t_1], M_2[t_1] = m_2[t_1], \ldots, M_n[t_1] = m_n[t_1]) \\
& \quad = \Pi_{1 \le i \le n} P(M_i[t_2] = m_i[t_2] \mid M_i[t_1] = m_i[t_1])
\end{aligned}
$$

Each of $P(M_i[t_2] = m_i[t_2] \mid M_i[t_1] = m_i[t_1])$ can be computed as shown in Table 4.1.

Thus, the failure processes of the individual components and an assumption about their independence gives us a model for the persistence of the system state. The distribution $P(\Delta[t_2] \mid \Delta[t_1])$ can be computed using t_2, t_1 and the MTBFs of the individual components M_i.

4.2 Using reliability models in diagnosis

We illustrate the use of reliability models in probabilistic model-based diagnosis with some examples. We will first describe a system and a decision theoretic repair model for the system. We will then describe two different scenarios with this system.

The first scenario demonstrates how diagnosis/repair depends crucially on *when* an observation is made. The time of observation affects the priors used for the diagnosis and this affects the posterior probabilities and the repair decision. The observation in the first scenario is a "nothing wrong" observation. That is, the observation does not indicate any discrepancy. We show that if this observation was made when the system was new, then the probability of everything being **ok** is very large (as we would expect) and the optimal decision is to do nothing (again, as we would expect). However, if the same observation is made instead when the system is older, the posterior probability that everything is **ok** decreases substantially. In this case, the optimal decision is to actually replace two of the components. This decision can be seen as an example of preventive maintenance.

The second scenario demonstrates the need for persistence models in the presence of multiple observations and actions. We first make an observation at time t_1. The corresponding best action is computed and executed and is found to fix the anomaly exhibited by the system. Now, at a later time t_2, the same anomalous observation occurs. A persistence model for the component states is required to compute the posterior probabilities and the optimal decision after the second observation. Though the observation at time t_2 and time t_1 are the same, the posterior distribution and the optimal decision are very different.

4.2.1 The example system

The example we consider is similar in structure to the example introduced in Chapter 1. However, we do not have prior probability estimates for the failure of the gates. Instead, we have MTBF reliability estimates. In addition, we add a simple cost model to the system to demonstrate how repair decisions are affected by the time of the observation.

Consider the digital circuit shown in Fig 4.2. A is an And gate, O is an Or gate and X is an Xor gate. Each of the gates can be **ok** or **broken**. When a gate is **ok**, it works as it is supposed to. When it is **broken** we need a fault model. Specifically,

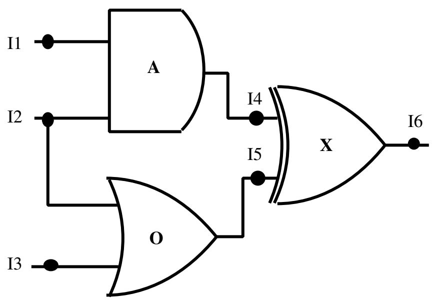


Figure 4.2: Using reliability models: An example.

we will assume *stuck-at-0* fault models for A and O. For gate X, we will depart from the example of Chapter 1 and choose a *stuck-at-1* fault model. The fault model is interpreted as follows: when A (or O) is in the **broken** state, the output of the gate is 0 irrespective of the input. Similarly, when X is **broken**, its output is 1 irrespective of the input. Say the MTBF of the And gate A is 100 hours, that of the Or gate O is 250 hours and that of the Xor gate X is 350 hours.

We also have a cost model that describes the repair costs of the system. We associate a repair decision with each component. The decision alternatives are **fix** and **dont-fix**. We describe the repair cost for each component with a cost function whose arguments are its mode state and the repair alternative. The cost functions for each of the gates is shown in Table 4.2. The system repair cost is assumed to be the sum of the repair costs of the components. That is, given a state for each mode variable of the system and a repair decision alternative for each component, we can sum the cost of the repair decision alternative for each component to get the repair cost function L_S for the entire system. The arguments to this cost function are a candidate (i.e., state assignment to each component of the system) and a *composite* decision. A composite decision consists of a choice of a repair alternative for each component in the system.

The cost function L for each component is quite intuitively assessed. Consider the case where the component A is **ok** and the decision is **dont-fix**. We use this as the reference situation and assign it a cost of \$0. Hence L_A(**ok**,**dont-fix**) = \$0. When we do decide to **fix**, say that we throw the old component away and replace it with a new one irrespective of what its actual state is. The cost of implementing the **fix** decision summarizes the temporary downtime cost required to bring the system down to change the component *and* the cost of the new component. Say we determine this cost to be \$2. Hence L_A(**broken**,**fix**) = L_A(**ok**, **fix**) = \$2. Finally, we assign a cost to the case where the gate is **broken**, but we choose **dont-fix**. This cost

$L_A(m,d)$			$L_O(m,d)$			$L_X(m,d)$		
broken	**fix**	\$2	**broken**	**fix**	\$3	**broken**	**fix**	\$4
broken	**dont-fix**	\$8	**broken**	**dont-fix**	\$12	**broken**	**dont-fix**	\$14
ok	**fix**	\$2	**ok**	**fix**	\$3	**ok**	**fix**	\$4
ok	**dont-fix**	\$0	**ok**	**dont-fix**	\$0	**ok**	**dont-fix**	\$0

Table 4.2: Repair cost functions for the example system.

basically summarizes the cost of system downtime as a result of this non-operational component. Say we determine the cost to be \$10. Hence L_A(**broken**,**dont-fix**) = \$10. There is an implicit time horizon over which these costs are being measured—say it is 1 hour in this example. In this framework, diagnosis involves computing the posterior distribution over the candidate state variable given observations. Repair involves choosing the optimal composite decision for the entire system given the posterior distribution over the candidates. The optimal decision d_{opt} is the one which results in the least expected cost. That is:

$$d_{opt} = min_d(\Sigma_c P(C = c|\mathbf{\Omega})L_S(D = d, C = c))$$

In this equation, d ranges over all possible composite decisions and c ranges over all possible candidates.

4.2.2 Scenario 1: Effect of time on diagnosis/repair

Consider the system of Fig 4.2. Say that it has been been up for 10 hours and the observation $\mathbf{\Omega} = \langle I_1 = 1,\ I_2 = 1,\ I_3 = 0,\ I_6 = 0\rangle$ is recorded. Note that $\mathbf{\Omega}$ seems to suggest that there is no problem with the system since the inputs generate the expected output. We assume that the time is counted from the time the system was new. We will further assume that when the system was new every component was certainly **ok**.

The system model can be translated into a Bayesian network as shown in Fig 4.3 (see Section 2.1). The node Δ has 8 states, one corresponding to each combination of states of M_A, M_O and M_X. Let m_A be a state of M_A. We define m_O and m_X similarly. The conditional distribution of Δ given its parents is just a deterministic distribution which maps each state combination of its parents to the corresponding state of C. That is, we have $P(\Delta = \langle m_A, m_O, m_X\rangle \mid M_A = m_A, M_O = m_O, M_X = m_X) = 1$.

We now perform diagnosis using the observation $\mathbf{\Omega}$ and the time of observation (viz, 10 hours). The prior probability of failure for each component is calculated on

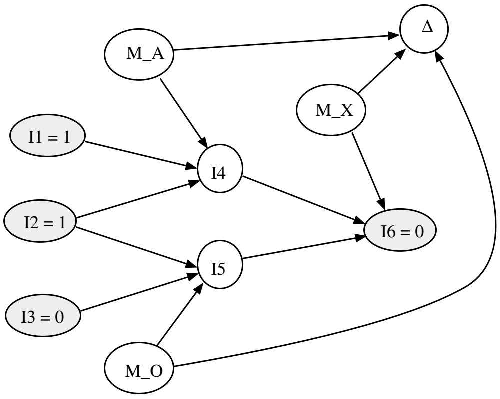


Figure 4.3: Bayesian network for Scenario 1.

the basis of it being up for 10 hours. The evidence $\mathbf{\Omega}$ is declared in the network (shown as grayed nodes) and then network inference is performed. We then look up the posterior distribution of Δ and compute the expected costs of the decisions. These are shown in Table 4.3(a). The results confirm our intuitions, viz, the most probable diagnosis is that nothing is wrong. The optimal decision is to not fix anything.

Consider now a situation where, instead of 10 hours, the system was running for 90 hours when observation $\mathbf{\Omega}$ was made. In this case, the prior probability of failure of each component is computed on the basis of it being up for 90 hours. We then do the diagnosis and compute the expected value of decisions for this situation. The posterior distributions and expected decision costs are shown in Table 4.3(b). In this situation, the most probable diagnosis is again that everything is ok, the same as when $\mathbf{\Omega}$ was observed after 10 hours. However, the most probable diagnosis is much less probable. The optimal decision in this situation is to replace the And gate and Or gate. This is an example of preventive maintenance. The expected costs of replacing now is lower than deferring the replacement. This is because the probability of failure is rising to a critical value.

4.2.3 Scenario 2: Modeling persistence

Say the system of Fig 4.2 has been been up for time $t_1 = 20$ hours and the observation $\mathbf{\Omega}[t_1] = \langle I_1 = 0,\ I_2 = 0,\ I_3 = 0,\ I_6 = 1\rangle$ is recorded. Note that $\mathbf{\Omega}[t_1]$ indicates some problem with the system—if everything was working correctly, the output I_6 would have value 0. Say we compute the posterior probabilities and the optimal decision. This computation is similar to that of the previous section. The posterior probabilities and optimal decision are shown in Table 4.4(a). The optimal decision is to fix the

(a)

Failure priors			
Comp.	Uptime	MTBF	$P(M = \textbf{broken})$
A	10	100	0.0952
O	10	250	0.0392
X	10	350	0.0282

Posterior $P(\Delta\|\boldsymbol{\Omega})$				Expected cost			
A	O	X		D_A	D_O	D_X	$
ok	**ok**	**ok**	0.9957	**dont**	**dont**	**dont**	0.0855
b	**b**	**ok**	0.0043	**fix**	**dont**	**dont**	2.0513
ok	**ok**	**b**	0.0000	**dont**	**fix**	**dont**	3.0342
ok	**b**	**ok**	0.0000	**dont**	**dont**	**fix**	4.0855
ok	**b**	**b**	0.0000	**fix**	**fix**	**dont**	5.0000
b	**ok**	**ok**	0.0000	**fix**	**dont**	**fix**	6.0513
b	**ok**	**b**	0.0000	**dont**	**fix**	**fix**	7.0342
b	**b**	**b**	0.0000	**fix**	**fix**	**fix**	9.0000

(b)

Failure priors			
Comp.	Uptime	MTBF	$P(M = \textbf{broken})$
A	90	100	0.5934
O	90	250	0.3023
X	90	350	0.2267

Posterior $P(\Delta\|\boldsymbol{\Omega})$				Expected cost			
A	O	X		D_A	D_O	D_X	$
ok	**ok**	**ok**	0.6126	**fix**	**fix**	**dont**	5.0000
b	**b**	**ok**	0.3874	**dont**	**fix**	**dont**	6.0995
ok	**ok**	**b**	0.0000	**fix**	**dont**	**dont**	6.6493
ok	**b**	**ok**	0.0000	**dont**	**dont**	**dont**	7.7488
ok	**b**	**b**	0.0000	**fix**	**fix**	**fix**	9.0000
b	**ok**	**ok**	0.0000	**dont**	**fix**	**fix**	10.0995
b	**ok**	**b**	0.0000	**fix**	**dont**	**fix**	10.6493
b	**b**	**b**	0.0000	**dont**	**dont**	**fix**	11.7488

Table 4.3: Posterior probabilities and expected costs in Scenario 1: (a) System up 10 hours, $\boldsymbol{\Omega}$ observed. (b) System up 90 hours, $\boldsymbol{\Omega}$ observed.

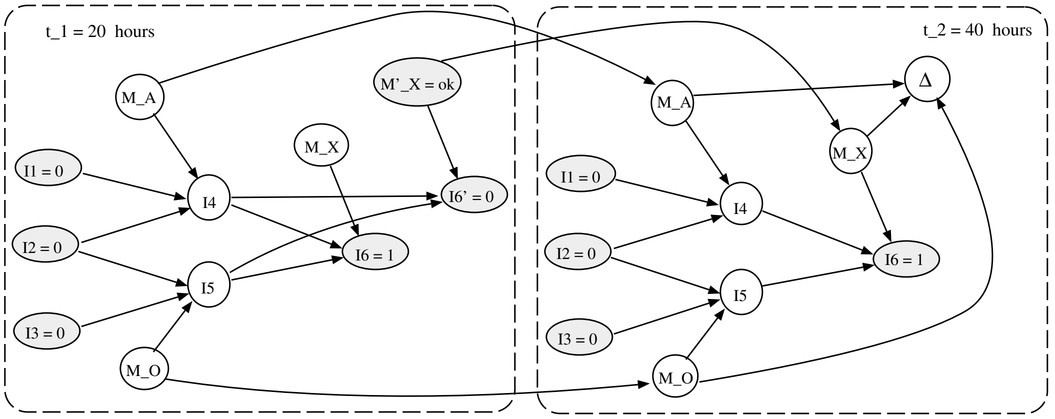


Figure 4.4: Bayesian network for Scenario 2.

XOR gate alone. Say we carry out this decision and then observe (immediately) that the output I_6 changes to 0. Thus, the observed discrepancy is fixed. Now say 20 hours more elapse and at time $t_2 = 40$ hours, we observe $\mathbf{\Omega}[t_2] = \langle I_1 = 0,\ I_2 = 0,\ I_3 = 0,\ I_6 = 1\rangle$. Note that $\mathbf{\Omega}[t_1]$ and $\mathbf{\Omega}[t_2]$ have the same values for the input and output variables.

The situation is represented by the dynamic Bayesian network of Fig 4.4. The section of the figure within the boundary marked t_1 represents the situation at time t_1. The variable $M'_X[t_1]$ represents the state of the XOR gate immediately after it is replaced. Note that immediately after replacement, we know that the gate is in the **ok** state. This is represented in the Bayesian network as evidence (grayed node). The variable $I'_6[t_1]$ represents the output immediately after the replacement action. As per our scenario, this variable is observed to have value 0.

The section of Fig 4.4 within the boundary marked t_2 represents the situation at time t_2. The link between $M_A[t_1]$ and $M_A[t_2]$ represents the persistence of the state of the AND gate. We compute the distribution $P(M_A[t_2]|M_A[t_1])$ using Table 4.1. When using the table, we set $\lambda_A = \frac{1}{MTBF_A} = \frac{1}{100}$ and $t_2 - t_1 = 20$. The conditional distributions $P(M_O[t_2]|M_O[t_1])$ and $P(M_X[t_2]|M'_X[t_1])$ are computed similarly. The anomalous observation at time t_2 is also entered as evidence in the Bayesian network (shown as grayed nodes).

The node $\Delta[t_2]$ has 8 states, each of which corresponds to one joint state of $M_A[t_2]$, $M_X[t_2]$ and $M_O[t_2]$. The conditional distribution of $\Delta[t_2]$ is quantified as described in the previous section. The posterior distribution over the states of the system at time t_2 can be computed simply by doing inference in the Bayesian network and looking up the posterior distribution of $\Delta[t_2]$. The posterior distribution at time t_2 and the corresponding decision costs are shown in Table 4.4(b). The optimal decision now is to replace both the XOR and the AND gate.

Superficially, the situation at time t_1 and t_2 seem to be quite similar, viz, that (a) the system has been working for 20 hours since "things were ok" (b) the observation $\langle I_1 = 0,\ I_2 = 0,\ I_3 = 0,\ I_6 = 1\rangle$ is made. However, the situations are actually quite different. At time t_2 we have to account for the (a) the observations and actions at time t_1 and (b) possible failures of components between time t_1 and t_2.

The persistence model for the system allows us to incorporate this information when computing the posterior distribution at time t_2. Note that the Bayesian network inference is implicitly carrying out the computation of $P(\Delta[t_2] \mid \Delta[t_1])$.

4.3 Discussion

Modeling persistence in diagnosis has been of interest both in the model-based diagnosis community and in the Bayesian network community. Early work in the model-based diagnosis community has handled persistence only to the extent of assuming that in the case of multiple observations, the state of each component stays the same across all observations. This corresponds, in our framework, to the situation where multiple observations are made very close in time.

Portinale [Portinale, 1992] addresses the problem of temporal evolution of state in model-based diagnosis. The evolution is modeled as a discrete time Markov chain. The state transition matrix is assumed to come from reliability measures of components. A uniform initial prior on all possible world states is used. A definition of a temporal diagnosis is proposed that generalizes the definition of a diagnosis in a static system. A method of eliminating very unlikely diagnoses is proposed. Our approach has a similar motivation, viz, to use models of component failure processes to model change of system state. However, our approach is significantly more general. We show that the problem of specification of priors can be addressed within the same framework. The priors are computed directly from the time at which diagnosis takes place and the MTBFs of the components. Our approach also allows modeling of effects of repair actions (as in the example of Section 4.2.3). Time is considered to be continuous. Finally, rather than eliminating unlikely diagnoses, we compute posterior probabilities over candidates. This is necessary if a decision theoretic scheme for choosing actions is to be used.

The model of persistence developed in this chapter is closely related to work in modeling persistence in Bayesian networks. Dean and Kanazawa [Dean and Kanazawa, 1989] propose a modeling framework for persistence and change using temporal probabilistic networks. A Markov assumption is employed. The temporal probabilistic network is quantified by specifying the conditional probability of each proposition in the network given the state of the same proposition at the immediately preceding

(a)

Posterior $P(\Delta\|\mathbf{\Omega})$				Expected cost			
A	O	X		D_A	D_O	D_X	\$
ok	**ok**	**b**	0.7558	**dont**	**dont**	**fix**	6.3728
b	**ok**	**b**	0.1673	**fix**	**dont**	**fix**	6.9226
ok	**b**	**b**	0.0629	**dont**	**fix**	**fix**	8.4502
b	**b**	**b**	0.0139	**fix**	**fix**	**fix**	9.0000
b	**b**	**ok**	0.0000	**dont**	**dont**	**dont**	16.3728
b	**ok**	**ok**	0.0000	**fix**	**dont**	**dont**	16.9226
ok	**b**	**ok**	0.0000	**dont**	**fix**	**dont**	18.4502
ok	**ok**	**ok**	0.0000	**fix**	**fix**	**dont**	19.0000

(b)

Posterior $P(\Delta\|\mathbf{\Omega})$				Expected cost			
A	O	X		D_A	D_O	D_X	\$
ok	**ok**	**b**	0.5712	**fix**	**dont**	**fix**	7.7743
b	**ok**	**b**	0.2809	**dont**	**dont**	**fix**	8.4117
ok	**b**	**b**	0.0991	**fix**	**fix**	**fix**	9.0000
b	**b**	**b**	0.0487	**dont**	**fix**	**fix**	9.6374
b	**b**	**ok**	0.0000	**fix**	**dont**	**dont**	17.7743
b	**ok**	**ok**	0.0000	**dont**	**dont**	**dont**	18.4117
ok	**b**	**ok**	0.0000	**fix**	**fix**	**dont**	19.0000
ok	**ok**	**ok**	0.0000	**dont**	**fix**	**dont**	19.6374

Table 4.4: Posterior probabilites and Expected costs for Scenario 2: (a) at time t_1 (before repair) (b) at time t_2.

time point and the states of the causes of the proposition at the previous time point. An exponentially decaying survivor function is proposed to model the probability of a proposition being true if it was true at the previous time and none of the causes that make it untrue are active. The survivor function accounts for unmodeled causes that might change the state of the proposition.

Our model can be viewed as a specific instantiation of this general framework. We assume that the only system state that persists across time is the mode of the variables and that the failure processes of the components are independent. This allows us to model persistence of system state by simply modeling the persistence of each component individually. The persistence model for each component falls directly out of the physical process of failure postulated by the reliability model. This persistence model is an exponential decay function.

Dagum et. al.[Dagum *et al.*, 1992] develop a forecasting approach using dynamic network models. Either an additive or a multiplicative model is used to combine predictions from current observations and the predictions from historical observations. The modeling of action (and hence of persistence) in causal probabilistic networks has been a field of active interest [Balke and Pearl, 1994; Darwiche and Goldszmidt, 1994; Heckerman and Shachter, 1994; Pearl, 1994]. When reasoning about actions which occur over separated points in time, one also has to account for possible changes in system state occurring due to unmodeled events. Our work advances a specific method for doing so in the domain of diagnosis of physical systems.

Our method for modeling persistence has been developed assuming that the behavior of each component is deterministic, both when it is in the **ok** state and when it is in the **broken** state. This allows us to consider only the mode variables of the components as the persisting state of the system. This is because the joint state of the mode variables completely determines the behavior of the system. In the situation where components are not deterministic[1], as a first approximation, one might use the same modeling scheme as the one presented in this chapter. This would mean, in effect, that we assume that each component "resamples" to compute its output in each observation. Thus, the same component with the same input could possibly have two different outputs in two observations (if the component were in the broken state). If this approximation is inaccurate, modeling extensions along the lines suggested by Darwiche and Goldszmidt can be incorporated.

The work described in this chapter has appeared in [Srinivas, 1995a].

[1] For example, we might assume that all outputs are equally likely if the component is **broken** in the case that no fault model is available.

Chapter 5

Computing Repair Strategies

The goal of doing diagnosis is to recommend cost effective repair and maintenance actions in response to inferences about the state of the system. A repair *strategy* can be considered to be a set of situation-action rules. The situations are the various possible observations and the actions are repair actions in response to these observations. In this chapter, we investigate methods for computing optimal (i.e., lowest expected cost) repair strategies.

We first extend the diagnosis framework developed thus far to include a formalization of the repair problem. In the initial formalization, the only repair actions allowed are replacement of components. Using this formalization, we develop a general algorithm for computing an optimal repair strategy. However, this general algorithm is not tractable for large systems. Without any further structure in the problem, we are forced to consider each possible strategy in a combinatorial space of repair strategies to compute the optimal strategy.

To address this tractability problem, we introduce a restricted but interesting formulation of the repair problem. The restriction is on the behavior of the system modeled—it is assumed that the system will certainly exhibit an anomalous observation if any component fails. In this restricted formulation, we develop a polynomial time algorithm to compute the optimal repair strategy when component failures are independent.

We then extend the formulation to include component inspection (testing). We also extend the formulation to include hierarchical systems. Using the extended formulation, we develop a linear time algorithm to compute the optimal repair strategy for a hierarchical system where the possible repair actions include both component replacement and component inspection.

This chapter is structured as follows. In the next section we describe our formalization of the general repair problem. We then develop the general algorithm for computing the optimal repair strategy. As noted earlier, this algorithm is practical only for small systems.

In the following section, we introduce our restricted formulation of the repair problem. We first derive a condition for a repair strategy to be optimal in this formulation. This condition is derived for the general situation where component failures may be dependent. In the case where component failures are independent, we show that this optimality condition has a simple form. This simple form allows the optimal strategy to be computed in polynomial time with a simple sorting procedure.

We then go on to introduce a new class of repair actions, viz, component inspections. A component inspection tests whether a component is working or not. Following this, we introduce a system hierarchy and define the notion of a hierarchical repair strategy.

The optimality condition mentioned above is then used to develop a linear time algorithm for computing the optimal hierarchical repair strategy for a hierarchical system. A hierarchical repair strategy includes both inspection and component replacement actions.

In the final section of the chapter, we discuss related work and possible extensions of the results of the chapter to systems with dependent faults.

5.1 A general formulation

Consider a system with n components, any of which may fail. Say we have a system model (see Section 2.1) for this system. As we have seen earlier, the system model specifies a set of discrete-valued inputs I_i^1, I_i^2, $\ldots I_i^k$ and one discrete-valued output O_i for each component $\mathcal{C}_i$. We refer to the vector of variables $\langle I_i^1, I_i^2, \ldots, I_i^k \rangle$ as $\mathbf{I}_i$. We will refer to the normal mode of operation of $\mathcal{C}_i$ as **ok** and the mode variable associated with $\mathcal{C}_i$ as M_i. We assume that component failures are independent.

The system model specifies a prior distribution $P(M_i)$ over the possible modes of each component. The model of operation of the component specifies the value of the output of the component given the state of the component and the values of the inputs. When the component $\mathcal{C}_i$ is in the **ok** state, we will assume that a deterministic model of operation is available. In other words, given an input state, exactly one output state is possible. However, for the other states, we will leave open the possibility that the behavior is non-deterministic. For each of these states, the user can specify a probability distribution over the output for every possible input state. Specifying the model of operation of the component amounts to specifying the distribution $P(O_i|M_i, \mathbf{I}_i)$ (with the restriction that $P(O_i = o_i|M_i = \mathbf{ok}, \mathbf{I}_i = \mathbf{i}_i)$ always takes the value 0 or 1 for any o_i and $\mathbf{i}_i$).

Finally, we assume that the system has a single designated output variable. Hence, when viewed as a black box, the system has a set of input variables which we will call the *system input* variables and one output variable which we will call the *system*

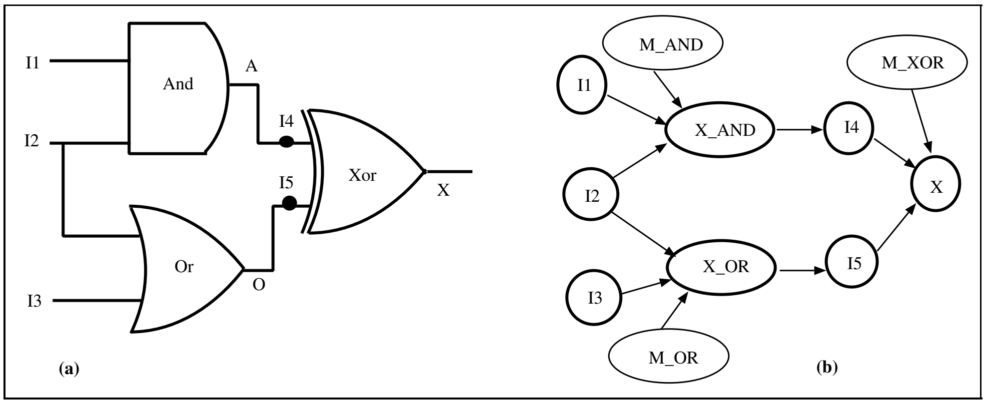


Figure 5.1: An example illustrating the general formulation of the repair problem: (a) the system model (b) the corresponding Bayesian network.

output variable.

Say we are observing the artifact modeled by our system model and that we observe some anomaly. That is, the system is given some input vector (which we can observe), and we observe that the value of the system output variable is inconsistent with the correct operation of the system.

We wish to take actions to repair the system. We will define repairing the system as correcting the *perceived anomalous observation*. The actions available to us are replacement of components. A *repair strategy* is a sequence in which the components are replaced. After each successive replacement, we check the output of the system. We assume the input remains fixed at what it was when the anomaly first appeared. If the output is still anomalous, we continue onwards to replace the next component recommended by the strategy. If the output is no longer anomalous, we stop. Executing each possible repair strategy (i.e., each possible repair sequence) has an expected cost. The optimal repair strategy is the one with the least expected cost for a particular system input.

If we compute the optimal strategy for each possible system input value, we determine the *optimal repair plan* for the system. The optimal repair plan is a set of situation–action rules. If a system has anomalous behavior, the optimal repair plan gives us a repair strategy to use as a function of the input. We will now develop an algorithm to compute optimal strategies and the overall optimal repair plan.

5.1.1 Computing the optimal repair plan

Consider a system which has a vector of input variables $\mathbf{I}$. Let the system output variable be X. Let us assume that the system has been given an input $\mathbf{i}$. Further, we assume that the *correct* system output for this input is $x(\mathbf{i})$ and we have observed that the output X has some value other than the correct output[1].

Consider a repair strategy $T = \langle \mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n \rangle$. T is a sequence describing the order in which components will be replaced. We now develop an expression for the expected cost of T. We will refer to the action of replacing $\mathcal{C}_i$ as fix_i. In addition, we will refer to the system output X after replacement of the i-th component in the sequence as X_i. Note in particular that the variable X_0 denotes the value of X before replacement of any component. Let S_j be the sequence of observations and actions up to and including the replacement of $\mathcal{C}_j$. That is: $S_j = \langle \mathbf{I} = \mathbf{i},\ X_0 = \neg x(\mathbf{i}),\ fix_1,\ X_1 = \neg x(\mathbf{i}),\ fix_2,\ X_2 = \neg x(\mathbf{i}),\ \ldots,\ fix_{j-1},\ X_{j-1} = \neg x(\mathbf{i}),\ fix_j \rangle$. The expected cost of T is given by:

$$\begin{aligned}
&EC(T|\mathbf{I} = \mathbf{i}, X = \neg x(\mathbf{i})) = \\
&\quad (c_1 + \\
&\quad P(X_1 = \neg x(\mathbf{i})|S_1)(c_2 + \\
&\quad P(X_2 = \neg x(\mathbf{i})|S_2)(c_3 + \\
&\quad \ldots \\
&\quad P(X_j = \neg x(\mathbf{i})|S_j)(c_{j+1} \\
&\quad \ldots \\
&\quad P(X_{n-1} = \neg x(\mathbf{i})|S_{n-1})c_n \ldots) \ldots)))
\end{aligned} \tag{5.1}$$

To determine the expected cost as described in Equation 5.1, we need to compute the conditional probabilities $P(X_j = \neg x(\mathbf{i})|S_j)$, $1 \leq j < n$.

We explain how these probabilities can be computed using the example introduced in Chapter 1. Consider the electronic circuit displayed in Fig 5.1(a). This system model for this circuit can be translated into a Bayesian network as in Fig 5.1(b) (see Chapter 2).

Consider the repair sequence $T = \langle AND, XOR, OR \rangle$. The network of Fig 5.2 represents the situation after the AND gate has been replaced. The modes of the XOR and OR gates are unaffected by this replacement and have the same value both before and after the repair action. The arcs between the copies of the static network in Fig 5.2 model this persistence.

The probability $P(X_1 = \neg x(\mathbf{i})|S_1)$ can be computed by declaring the evidence S_1 in the network, propagating it and then looking up the posterior belief of the event $X_1 = \neg x(\mathbf{i})$ in node X_1. The evidence S_1 consists of: (a) The known state of the

[1]Note that we can determine the correct system output by simply simulating the system forward from the input $\mathbf{i}$ while assuming that each of the components $\mathcal{C}_i$ are in the **ok** state.

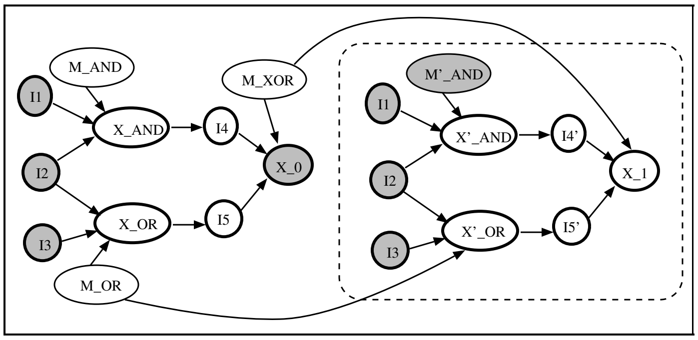


Figure 5.2: Representation of the situation after replacing the AND gate. Arcs between copies of the static network represent persistence of state.

input both before and after the repair action. (b) The output X_0 (which has the value $\neg x(\mathbf{i})$) and (c) the state of the AND gate after repair ($M'_{AND} = \mathbf{ok}$). The corresponding nodes are shown shaded gray in the figure.

We will now describe a method for making this computation without explicitly constructing a dynamic Bayesian network. We note that there are *active paths* [Pearl, 1988] to node X_1 from node X_0 through the nodes M_{XOR} and M_{OR}. Hence, the computation of the posterior of X_1 will necessarily have to consider cases for every possible joint state[2] of the variables M_{XOR} and M_{OR}. The computation can be written as follows:

$$
\begin{aligned}
&P(X_1 = \neg x(\mathbf{i}) | S_1) = \qquad (5.2)\\
&\quad \Sigma_{m_{XOR}, m_{OR}} P(X_1 = \neg x(\mathbf{i}) | \mathbf{I} = \mathbf{i}, M'_{AND} = \mathbf{ok}, M_{XOR} = m_{XOR}, M_{OR} = m_{OR})\\
&\qquad\qquad P(M_{XOR} = m_{XOR}, M_{OR} = m_{OR} | \mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}))
\end{aligned}
$$

In the above equation, m_{XOR} and m_{OR} represent generic states of M_{XOR} and M_{OR} respectively. Hence, the summation in the equation iterates over all possible joint states of M_{XOR} and M_{OR}. The equation also accounts for the fact that knowing the state of M_{XOR} and M_{OR} makes X_1 conditionally independent of X_0.

Assume that we have access to the probability distribution $P(M_{XOR}, M_{OR} | \mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}))$ (i.e., the second term of Equation 5.2). We will see how this distribution is computed with an iterative scheme later.

We note that the probabilities needed for the first term of the equation, $P(X_1 = \neg x(\mathbf{i}) | \mathbf{I} = \mathbf{i}, M'_{AND} = \mathbf{ok}, M_{XOR} = m_{XOR}, M_{OR} = m_{OR})$, can be computed directly

[2] A joint state of a set of discrete random variables assigns a value to each of the variables in the set.

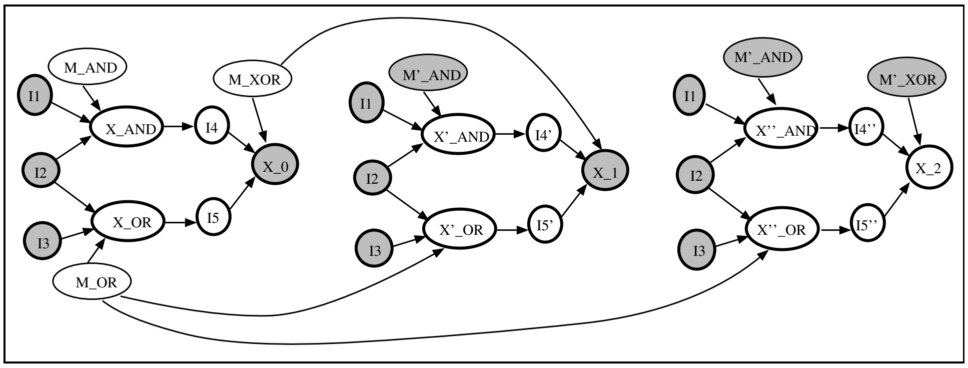


Figure 5.3: Situation after replacing the XOR gate.

from the static Bayesian network of Fig 5.1(b). This is so because knowing the states of M_{XOR} and M_{OR} makes the post-repair network fragment (shown within the dotted lines in Fig 5.2) independent of the rest of the network. The post-repair network fragment is identical to the static Bayesian network. Thus, the required probability can be computed by (a) declaring the evidence $M_{AND} = \mathbf{ok}$, $M_{XOR} = m_{XOR}$, $M_{OR} = m_{OR}$, $\mathbf{I} = \mathbf{i}$ in the static network, and (b) propagating the evidence and looking up the posterior of the event $X = \neg x(\mathbf{i})$ in node X.

Now, let us consider the replacement of the next component specified by the repair strategy, i.e., the XOR gate. The situation is shown in Fig 5.3. By analogy with the previous situation, we can compute $P(X_2 = \neg x(\mathbf{i})|S_2)$ as:

$$\begin{aligned} & P(X_2 = \neg x(\mathbf{i})|S_2) = \\ & \quad \Sigma_{m_{OR}} P(X_2 = \neg x(\mathbf{i})|\mathbf{I} = \mathbf{i}, M'_{XOR} = \mathbf{ok}, M'_{AND} = \mathbf{ok}, M_{OR} = m_{OR}) \\ & \quad P(M_{OR} = m_{OR}|\mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}), M'_{AND} = \mathbf{ok}, X_1 = \neg x(\mathbf{i})) \end{aligned} \tag{5.3}$$

Here, again, the probabilities for the first term in the equation can be directly computed from the system model. We will now see how the probabilities for the second term can be computed from the computations of the previous repair step (i.e., Equation 5.2). Note that the product within the summation of Equation 5.2 is equal to:

$$P(X_1 = \neg x(\mathbf{i}), M_{XOR} = m_{XOR}, M_{OR} = m_{OR}|\mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}), M'_{AND} = \mathbf{ok})$$

If we normalize the above quantity over all joint states of M_{XOR} and M_{OR}, we obtain the distribution:

$$P(M_{XOR} = m_{XOR}, M_{OR} = m_{OR}|\mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}), M'_{AND} = \mathbf{ok}, X_1 = \neg x(\mathbf{i})) \tag{5.4}$$

Now, we consider the action of replacing the XOR gate. This does not affect our estimate of what state the OR gate is in, i.e., it has no affect on the posterior probability distribution of M_{OR} given the current state of information. This posterior probability is:

$$P(M_{OR} = m_{OR} | \mathbf{I} = \mathbf{i}, X_0 = \neg x(\mathbf{i}), M'_{AND} = \mathbf{ok}, X_1 = \neg x(\mathbf{i}))$$

This posterior distribution can be computed by simply summing the distribution of Equation 5.4 over all states of M_{XOR}. Note that this distribution is the second term of Equation 5.3. Hence we can now compute $P(X_2 = \neg x(\mathbf{i}) | S_2)$ using Equation 5.3.

This example can be generalized to yield a simple iterative scheme for computing $P(X_j = \neg \mathbf{x}(\mathbf{i}) | S_j)$ for $1 \leq j < n$, given a repair sequence T. The basic idea is the following: All the information coming from the first $j-1$ observations and repair actions is summarized by the posterior probability distribution over the joint states of the components *which have not yet been fixed* (i.e., $\mathcal{C}_j$ through $\mathcal{C}_n$). This posterior probability is used iteratively to perform the following calculations:

For $j = 1$ to n:

1. Compute the probability of an anomaly after the j-th fix action (i.e., $P(X_j = \neg \mathbf{x}(\mathbf{i}) | S_j)$).
2. Compute the new updated posterior, accounting for the j-th action. Note that this posterior is over the joint states of $\mathcal{C}_{j+1}$ through $\mathcal{C}_n$.

To begin the iteration, we need to have the posterior over all joint states of all the components given that no observation and repair actions have been performed. Note that this posterior probability is just the prior probability over the joint states of the components. Since components fail independently, this distribution is the product of the marginal distributions over the modes of each component.

In general, when computing $P(X_j = \neg x(\mathbf{i}) | S_j)$ we note that there is an active path in the corresponding dynamic Bayesian network from the observed node X_{j-1} to the target node X_j through the mode variable of each unfixed component (i.e., there are active paths through M_{j+1}, M_{j+2}, $\ldots$, M_n). As a result, a cutset for the network necessarily includes each of these variables. This implies that an inference algorithm computing $P(X_j = \neg x(\mathbf{i}) | S_j)$ will necessarily have to condition on each joint state of the modes of the unfixed components. Thus, our iterative scheme can be considered as carrying forward the posterior over the cutset nodes. In this sense, the scheme is optimal for computing the probabilities $P(X_j = \neg x(\mathbf{i}) | S_j)$.

Computing the best strategy

We now have the probabilities required in the right hand side of Equation 5.1, enabling us to compute the expected cost $EC(T | \mathbf{I} = \mathbf{i}, X = \neg x(\mathbf{i}))$, given a strategy T and a system input $\mathbf{I} = \mathbf{i}$. Identifying the best strategy can be done by checking the expected cost of all strategies. Computing the best strategy can be done in $O(n!S_M)$ where n is the number of components and S_M is the joint space size of the mode variables of all the components. S_M is exponential in n.

If we wish to go on to compute the optimal repair plan, we must compute the best possible strategy for every possible input value to the system. Let the joint space size of the inputs to the system be S_I. The overall complexity of computing the optimal strategy for every possible input to the system is then $O(n! \times S_M \times S_I)$. Computing the optimal repair plan with the algorithm described above is impractical for large systems. We now introduce a restricted formulation of the repair problem which allows the optimal repair plan to be computed tractably.

5.2 The restricted formulation

Consider a system with n components $\mathcal{C}_i$, $1 \leq i \leq n$, for which we want to develop good repair strategies. Say each component can be either be in an ok state (**ok**) or broken state (**b**). The state of $\mathcal{C}_i$ is represented by a mode variable M_i. Component failures may be dependent. The prior over component failures is specified by some joint distribution $P(M_1, M_2, \ldots, M_n)$ of the mode variables. In addition, we are given a repair cost c_i for each component $\mathcal{C}_i$. After a component is repaired, we assume that it is in the **ok** state. The cost c_i can also be interpreted as the cost of replacing $\mathcal{C}_i$.

The restriction on the system behavior is as follows: We assume that the system works normally only if all the components are in the **ok** state. If any of the components are in the **b** state, the system exhibits a fault. In terms of the general formulation, this means that for every possible input, the system exhibits an anomalous output if any of the components is faulty. The output is not anomalous if and only if all the components are working normally. We assume that the system status (denoted by X) is observable. If $X = \mathbf{ok}$, then it means the system is working normally. If $X = \mathbf{b}$, it means the system is broken (i.e., exhibiting a fault).

The repair protocol is as follows—we will observe the system status X_0 before we choose any fix action. If $X_0 = \mathbf{ok}$ we stop. If $X_0 = \mathbf{b}$, then we choose to fix some component $\mathcal{C}_1$ and then observe the system status X_1. If $X_1 = \mathbf{ok}$ we stop. If $X_1 = \mathbf{b}$, we continue, choosing some other component $\mathcal{C}_2$ to fix and so on. As before, we will refer to the action of fixing $\mathcal{C}_j$ as fix_j. A repair *strategy* is a sequence in which to replace components in the repair protocol described above.

Consider a strategy $T = \langle \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \ldots, \mathcal{C}_n \rangle$. Say that the first k components have been repaired according to strategy T. We will refer to the sequence of observations and actions up to this point as S_k. Hence, $S_k = \langle X_0 = \mathbf{b}, fix_1, X_1 = \mathbf{b}, fix_2, X_2 =$

$\mathbf{b}, \ldots, fix_{k-1}, X_{k-1} = \mathbf{b}, fix_k\rangle$[3].

After performing fix_k, say we find that the system is still faulty, i.e., $X_k = \mathbf{b}$. This event occurs with probability $P(X_k = \mathbf{b}|S_k)$. When this event occurs, we go on to perform the action fix_{k+1} incurring cost c_{k+1}.

We can thus compute the expected cost $EC(T)$ of the strategy T as:

$$\begin{aligned} EC(T) \quad = \quad & P(X_0 = \mathbf{b})[c_1 + \\ & P(X_1 = \mathbf{b}|S_1)[c_2 + \\ & P(X_2 = \mathbf{b}|S_2)[c_3 + \\ & \ldots \\ & P(X_{m-1} = \mathbf{b}|S_{m-1})[c_m \\ & P(X_m = \mathbf{b}|S_m)[c_{m+1} \\ & \ldots \\ & +P(X_{n-1} = \mathbf{b}|S_{n-1})c_n \ldots] \ldots]]] \end{aligned} \tag{5.5}$$

We will now simplify the above equation. We note that the following identity is true:

$$\begin{aligned} & P(X_k = \mathbf{b}|S_k)P(X_{k-1} = \mathbf{b}|S_{k-1}) \\ & = \quad P(X_k = \mathbf{b}|fix_k, X_{k-1} = \mathbf{b}, S_{k-1})P(X_{k-1} = \mathbf{b}|S_{k-1}) \\ & = \quad P(X_k = \mathbf{b}|fix_k, X_{k-1} = \mathbf{b}, S_{k-1})P(X_{k-1} = \mathbf{b}|S_{k-1}, fix_k) \\ & = \quad P(X_k = b, X_{k-1} = \mathbf{b}|fix_k, S_{k-1}) \end{aligned} \tag{5.6}$$

The second step of the derivation above follows because fix_k occurs in the future, after X_{k-1} is observed. Thus, the distribution of X_{k-1} is not dependent on the action fix_k. By the same argument, we derive:

$$\begin{aligned} & P(X_k = \mathbf{b}|S_k)P(X_{k-1} = \mathbf{b}|S_{k-1})P(X_{k-2} = \mathbf{b}|S_{k-2}) \\ & = \quad P(X_k = \mathbf{b}, X_{k-1} = \mathbf{b}, X_{k-2} = \mathbf{b}|fix_k, fix_{k-1}, S_{k-2}) \end{aligned} \tag{5.7}$$

We define the notation $fix_{[i,j]}$ to refer to the sequence $\langle fix_i, fix_{i+1}, \ldots, fix_j\rangle$. Similarly, $X_{[i,j]} = \mathbf{b}$ refers to the event $\langle X_i = \mathbf{b}, X_{i+1} = \mathbf{b}, \ldots, X_j = \mathbf{b}\rangle$. Using the argument described above repeatedly, we derive:

$$\{\Pi_{0 \leq i \leq k} \; P(X_i = \mathbf{b}|S_i)\} = P(X_{[0,k]} = \mathbf{b}|fix_{[1,k]}) \tag{5.8}$$

[3] Note that S_0 refers to the empty sequence of observations and actions (i.e., no observations and no fix actions).

The above equation allows us to simplify Equation 5.5 to:

$$EC(T) = \Sigma_{0\leq k\leq n-1}\ \ c_{k+1} \times P(X_{[0,k]} = \mathbf{b}|fix_{[1,k]}) \tag{5.9}$$

We now derive an expression for $P(X_{[0,k]} = \mathbf{b}|fix_{[1,k]})$ from first principles. Let a world be a state assignment to all the mode variables of the system. Given that $fix_{[1,k]}$ have been performed, consider the worlds that are *inconsistent* with $X_{[0,k]} = \mathbf{b}$ being observed. A world w is inconsistent with $X_{[0,k]} = \mathbf{b}$ iff the observation $X_{[0,k]} = \mathbf{b}$ could *not* have occurred if the true state of the system was w.

We see that the worlds *inconsistent* with $X_{[0,k]} = \mathbf{b}$ are exactly those worlds $\hat{w}$ in which *all* the components which have not yet been fixed (i.e., $\mathcal{C}_{k+1}$ through $\mathcal{C}_n$) are in the **ok** state. The reason is as follows. If any of the worlds $\hat{w}$ had been the true situation, then we know that the broken components are some subset of $\{\mathcal{C}_i|1 \leq i \leq k\}$. Hence, the repair sequence $fix_{[1,k]}$ would necessarily have resulted in $X_j = \mathbf{ok}$ for some $j \leq k$ (when all the broken components were fixed). Since such an observation is inconsistent with $X_{[0,k]} = \mathbf{b}$, we conclude that $\hat{w}$ is inconsistent with $X_{[0,k]} = \mathbf{b}$.

By a similar line of argument, we can conclude that any world in which at least one of the remaining unfixed components is broken is consistent with $X_{[0,k]} = \mathbf{b}$. The total probability mass of the worlds in which all of $\mathcal{C}_{k+1}, \mathcal{C}_{k+2}, \ldots, \mathcal{C}_n$ are in the **ok** state is $P(M_{k+1} = \mathbf{ok}, M_{k+2} = \mathbf{ok}, \ldots, M_n = \mathbf{ok})$. Hence $P(X_{[0,k]} = \mathbf{b}|fix_{[1,k]}) = 1 - P(M_{k+1} = \mathbf{ok}, M_{k+2} = \mathbf{ok}, \ldots, M_n = \mathbf{ok})$. We will use the notation $M_{[i,j]} = \mathbf{ok}$ as a short form for $\langle M_i = \mathbf{ok}, M_{i+1} = \mathbf{ok}, \ldots, M_j = \mathbf{ok}\rangle$. Hence, Equation 5.9 simplifies to:

$$\begin{aligned} EC(T) &= \Sigma_{0\leq k\leq n-1}\ \ c_{k+1} \times [1 - P(M_{[k+1,n]} = \mathbf{ok})] \\ &= \Sigma_{1\leq k\leq n}\ \ c_k \times [1 - P(M_{[k,n]} = \mathbf{ok})] \end{aligned} \tag{5.10}$$

5.3 The optimality condition

We will now derive a condition under which a strategy is optimal (i.e., has the lowest possible expected cost).

Consider a strategy $T^j = \langle\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_j, \mathcal{C}_{j+1}, \ldots, \mathcal{C}_n\rangle$. Let T^{j+1} be identical to T^j except that the positions of the $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ are transposed in the sequence. We compare the expected costs of T^j and T^{j+1}. We have:

$$\begin{aligned} &EC(T^j) - EC(T^{j+1}) = \\ &\quad (c_j[1 - P(M_{[j,n]} = \mathbf{ok})] + \\ &\quad c_{j+1}[1 - P(M_{[j+1,n]} = \mathbf{ok})]) \\ &\quad -(c_{j+1}[1 - P(M_{[j,n]} = \mathbf{ok})] + \end{aligned} \tag{5.11}$$

$$c_j[1 - P(M_j = \mathbf{ok}, M_{[j+2,n]} = \mathbf{ok})])$$

Strategy T^j is less expensive that T^{j+1} if $EC(T^j) - EC(T^{j+1}) \leq 0$. Let us use the notation $R_{\mathbf{ok}}$ for the event $M_{[j+2,n]} = \mathbf{ok}$. Simplifying Equation 5.11, the condition $EC(T^j) - EC(T^{j+1}) \leq 0$ simplifies to:

$$\begin{aligned} & c_j[P(M_j = \mathbf{ok}, R_{\mathbf{ok}}) - P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok}, R_{\mathbf{ok}})] \\ \leq \quad & c_{j+1}[P(M_{j+1} = \mathbf{ok}, R_{\mathbf{ok}}) - P(M_{j+1} = \mathbf{ok}, M_j = \mathbf{ok}, R_{\mathbf{ok}})] \end{aligned} \tag{5.12}$$

Thus, given a distribution $P(M_1, M_2, \ldots, M_n)$ and a strategy T, we can check whether the strategy is a (local) optimum by checking whether Equation 5.12 holds for adjacent components in the strategy. If the condition does hold for every pair of adjacent components, the strategy is a local optimum. That is, exchanging the order of any two adjacent components in the strategy will always lead to a strategy with increased cost. Note that computation of the probabilities needed in Equation 5.12 from the joint distribution $P(M_1, M_2, \ldots, M_n)$ can be expensive. We will see, however, that the optimality condition takes a simple form when the failures of the components are independent.

5.3.1 A sanity check: The single fault case

We have derived the above condition assuming a general distribution $P(M_1, M_2, \ldots, M_n)$. We now show that if we enforce a single fault assumption, Equation 5.12 reduces to the optimality condition of [Kalagnanam and Henrion, 1990] (see Section 5.7). They prove that in the case of a single fault, the optimal strategy replaces components in increasing order of the ratio $\frac{c_i}{p_i}$ where c_i is the cost of replacement of $\mathcal{C}_i$ and p_i is the prior probability that $\mathcal{C}_i$ is faulty.

Consider a single fault distribution. There are only n possible worlds. Let these worlds be w_1, w_2, $\ldots w_n$. w_i is the world in which M_i is in the $\mathbf{b}$ state and all the other M_j (i.e., $j \neq i$) are in the $\mathbf{ok}$ state. Let the probability of world w_i be p_i. That is, the probability that $\mathcal{C}_i$ is the (only) faulty component is p_i.

Consider the probability $P(M_j = \mathbf{ok}, R_{\mathbf{ok}})$ in Equation 5.12. The worlds consistent with $\langle M_j = \mathbf{ok}, R_{\mathbf{ok}}\rangle$ are $w_1, w_2, \ldots, w_{j-1}$ and w_{j+1}. Hence $P(M_j = \mathbf{ok}, R_{\mathbf{ok}}) = (\Sigma_{[1 \leq i \leq j-1]} p_i) + p_{j+1}$. Let us refer to the quantity $(\Sigma_{[1 \leq i \leq j-1]} p_i)$ as p_{prev}. We have:

$$P(M_j = \mathbf{ok}, R_{\mathbf{ok}}) = p_{prev} + p_{j+1}$$

Using a similar line of reasoning:

$$P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok}, R_{\mathbf{ok}}) = p_{prev}$$

Hence the left hand side of Equation 5.12 reduces to $c_j[(p_{prev} + p_{j+1}) - p_{prev}] = c_j p_{j+1}$. Symmetrically, the right hand side reduces to $c_{j+1}p_j$. This simplifies Equation 5.12 to the result in [Kalagnanam and Henrion, 1990]:

$$\frac{c_j}{p_j} \leq \frac{c_{j+1}}{p_{j+1}}$$

In this special case, a strategy which satisfies the condition for every pair of adjacent components is globally optimal.

5.4 Independent faults

Say that each component $\mathcal{C}_i$ can fail independently with probability p_i. That is, $P(M_i = \mathbf{b}) = p_i$. We derive a simplification of the optimality condition (Equation 5.12) for this case.

Consider the first term of Equation 5.12. In this special case of multiple independent faults we have $P(M_j = \mathbf{ok}, R_{\mathbf{ok}}) = P(M_j = \mathbf{ok})P(R_{\mathbf{ok}}) = (1 - p_j)P(R_{\mathbf{ok}})$. Similarly $P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok}, R_{\mathbf{ok}}) = (1 - p_j)(1 - p_{j+1})P(R_{\mathbf{ok}})$. The first term of Equation 5.12 hence becomes $P(R_{\mathbf{ok}})c_j[(1 - p_j)p_{j+1}]$. Symmetrically, the second term of Equation 5.12 becomes $P(R_{\mathbf{ok}})c_{j+1}[(1 - p_{j+1})p_j]$. Hence Equation 5.12 reduces to:

$$c_j \frac{1 - p_j}{p_j} \leq c_{j+1} \frac{1 - p_{j+1}}{p_{j+1}} \tag{5.13}$$

From this result we note that we can compute the globally optimal strategy by sorting the components $\mathcal{C}_i$ by the quantity $(c_i \frac{1-p_i}{p_i})$. For an n component system, this can be done in $O(n \log n)$.

We get an expression for the expected cost of any strategy (including the optimal strategy) by simplifying Equation 5.10 for the case of multiple independent faults. This gives:

$$EC(T) = \Sigma_{1 \leq k \leq n} \;\; c_k \times [1 - \Pi_{k \leq i \leq n}(1 - p_i)] \tag{5.14}$$

5.5 Introducing component inspection

In the discussion thus far, we have assumed that the only kind of action that is allowed is the replacement of a component. We now introduce the notion of inspecting a component. Inspection of a component $\mathcal{C}_i$ determines what state it is in. Hence, if we inspect $\mathcal{C}_i$ and find that it is in the **ok** state, we do not have to take any further action to fix the component. Carrying out the inspection of $\mathcal{C}_i$ costs d_i. This cost is specified by the user. We note that $d_i \leq c_i$. If this was not the case, there would be no incentive to inspect the component—we could always replace it at less cost.

Now say that $\mathcal{C}_i$ has been inspected and found to be broken. In that case, we will assume the component can be *refurbished* and restored to normal operation by incurring cost H_i. Note that refurbishment is a new kind of action and is different from outright replacement of the component. The cost H_i of refurbishment is specified by the user. Note that $H_i \leq c_i$. If this was not the case, we would always replace the component rather than refurbish it.

As an example, say a particular component $\mathcal{C}_i$ of a system is a motor. Say the cost of a new motor is \$10. Thus we have $c_i = \$10$. Say we can inspect the motor (perhaps by disconnecting it and applying voltage to it directly). Say the cost of this procedure is \$4. Thus $d_i = \$4$. If the motor is indeed found to be not working after inspection, say it can be refurbished by replacing the winding for a cost of \$8. In this case, $H_i = \$8$. Note that the actual choice of repair action for the motor depends on our estimate of the posterior probability of failure of the motor at the point at which we decide to take action. If the probability of failure of the motor is high, then it may be cheaper to simply replace it without inspection. However, if the probability is small, inspection followed by possible refurbishment may be cheaper since inspection may often reveal that the motor is working normally, in which case no refurbishment cost needs to be incurred.

In this extended formulation of the repair problem, a repair strategy specifies an order in which to repair the components (as before). In addition, for each component it also specifies whether the component is to be inspected and then refurbished or simply replaced. An optimal strategy is the strategy with least expected cost. Later, when we introduce hierarchy, we will see that the "refurbishment" of a component is realized by an optimal repair strategy applied to its subcomponents.

We now turn to the problem of computing an optimal repair strategy in this extended formulation. Consider a strategy $T^m_{ins} = \langle\, [\mathcal{C}_1, rep], [\mathcal{C}_2, rep], \ldots, [\mathcal{C}_m, ins], \ldots, [\mathcal{C}_n, rep]\, \rangle$. The notation ins says that the associated component is to be inspected. The notation rep says that the associated component is to be simply replaced without inspection. Note that T^m_{ins} specifies that all components except $\mathcal{C}_m$ be replaced without inspection. $\mathcal{C}_m$ alone is inspected before it is repaired. We now compute the expected cost of strategy T^m_{ins}.

The cost of T^m_{ins} can be computed by simply replacing c_m in Equation 5.5 by $d_m + H_m P(M_m = \mathsf{b}|\Omega)$. That is, instead of the replacement cost c_m, we have to pay the inspection cost d_m. In addition, if component $\mathcal{C}_m$ is indeed broken, we have to pay cost H_m. The probability that we will find that m is broken after inspection is $P(M_m = \mathsf{b}|\Omega)$ where Ω is the current state of information. Ω includes all actions and observations up to the replacement of $\mathcal{C}_{m-1}$ and the subsequent observation that the system is still not functioning (i.e., $X_{m-1} = \mathbf{b}$). Hence $\Omega = \langle S_{m-1}, X_{m-1} = \mathbf{b}\rangle$.

Simplifying Equation 5.5 using the same technique that was used to derive Equation 5.9 gives us:

$$\begin{aligned} EC(T^m_{ins}) &= \left[\Sigma_{0\leq j<m-1}\ c_{j+1} \times P(X_{[0,j]} = \mathbf{b}|fix_{[1,j]})\right] \\ &\quad +d_m P(X_{[0,m-1]} = \mathbf{b}|fix_{[1,m-1]}) \\ &\quad + \left[\Sigma_{m\leq j\leq n-1}\ c_{j+1} \times P(X_{[0,j]} = \mathbf{b}|fix_{[1,j]}))\right] \\ &\quad +H_m P(M_m = \mathbf{b}|S_{m-1}, X_{m-1} = \mathbf{b}) P(X_{[0,m-1]} = \mathbf{b}|fix_{[1,m-1]}) \end{aligned} \tag{5.15}$$

We now simplify the last term in the above equation. By the definition of S_{m-1}, we have $\Omega = \langle S_{m-1}, X_{m-1} = \mathbf{b}\rangle = \langle X_{[0,m-1]} = \mathbf{b}, fix_{[1,m-1]}\rangle$. Hence, the last term in the above equation simplifies to $H_m P(M_m = \mathbf{b}, X_{[0,m-1]} = \mathbf{b}|fix_{[1,m-1]})$.

Consider the probability $P(M_m = \mathbf{b}, X_{[0,m-1]} = \mathbf{b}|fix_{[1,m-1]})$. We saw before that when the actions $fix_{[1,j]}$ have been carried out, the worlds inconsistent with $X_{[0,m-1]} = \mathbf{b}$ are those worlds in which the remaining unrepaired components, $\mathcal{C}_m$, $\mathcal{C}_{m+1}, \ldots, \mathcal{C}_n$ are all in the **ok** state. Note that $M_m = \mathbf{b}$ (i.e., $\mathcal{C}_m$ is in the broken (**b**) state) is inconsistent with all of these worlds. Hence, the set of worlds *consistent* with $M_m = \mathbf{b}$ (call the set W_1) is a subset of the set of worlds *consistent* with $X_{[0,m-1]} = \mathbf{b}$ (call this set W_2). As a result, we have:

$$\begin{aligned} P(M_m = \mathbf{b}, X_{[0,m]} = \mathbf{b}|fix_{[1,j]}) &= P(W_1 \cap W_2) \\ &= P(W_1) \\ &= P(M_m = \mathbf{b}) \end{aligned}$$

Hence, the trailing term in Equation 5.15 reduces to $H_m P(M_m = \mathbf{b})$. Note that this term is not dependent on the position of $\mathcal{C}_m$ in the repair sequence.

In general, if we are given a strategy T_P where some subset P of the components are inspected, we can come up with an expression for the cost as follows. Start with the expression for the case where every element in the strategy T_P is assumed to be replaced without inspection (i.e., start with Equation 5.9). In this expression, replace c_j by d_j for each component j which is in P. For each such component j, also add a trailing constant term $H_j P(M_j = \mathbf{b})$.

We now consider how we might compute an optimal strategy given a subset of components P to be inspected. We will assume that the component failures are independent. We note that given any strategy T_P which inspects just the components in P, the expression for the cost consists of two parts. One part is similar to Equation 5.9. The other part consists of constant terms of the type $H_j P(M_j = \mathbf{b})$ where

$j \in P$. The latter part is unaffected by the order in which the components appear in strategy T_P.

Therefore, to find the optimal strategy T_P^{min}, we have to only minimize the first part of the cost expression. Since the component failures are independent, we can directly apply Equation 5.13 to find the optimal sequence in which to repair the components. The optimal sequence satisfies:

$$cd_j \frac{1-p_j}{p_j} \leq cd_{j+1} \frac{1-p_{j+1}}{p_{j+1}} \tag{5.16}$$

where:

$$cd_j = \begin{cases} d_j & \text{if } j \in P \\ c_j & \text{if } j \notin P \end{cases}$$

Given the optimal strategy T_P^{min}, the optimal repair cost can be computed as:

$$\begin{aligned} & EC(T_P^{min}) = \\ & \quad \Sigma_{1 \leq k \leq n} \;\; cd_k \times [1 - \Pi_{k \leq i \leq n}(1-p_i)] \\ & \quad + \Sigma_{\mathcal{C}_i \in P} \;\; H_i P(M_i = \mathbf{b}) \end{aligned} \tag{5.17}$$

Thus, given a subset P of components which are to be inspected, we can compute an optimal strategy and its cost in $O(n \log n)$.

5.5.1 The globally optimal strategy

Say we consider every possible subset P of the set of components and compute T_P^{min}. The cheapest of all these strategies is necessarily the globally optimal strategy T^{opt}. Thus, if there are n components, we can compute T^{opt} in $O((n \log n)2^n)$. This, of course, is practical only when n is small. However, our intent is to use this result for computing optimal strategies for hierarchical systems. As we shall see below, in that context, n is indeed small.

5.5.2 The conditional expected cost of repair

Note that the expected cost EC in Equation 5.17 is the overall expected cost. This cost is computed assuming that no observations have been made of the system as yet. In particular, it is not yet known whether the system is faulty.

Consider instead the expected cost given that we know the system is faulty when we begin repair. This expected cost estimate will be needed later when computing optimal hierarchical repair strategies. For any strategy T, let $EC^f(T)$ denote the expected cost of repair given that we know the system is faulty. Note that the observation "System is faulty" is exactly $\langle X_0 = \mathbf{b} \rangle$. We see that $EC(T)$ and $EC^f(T)$

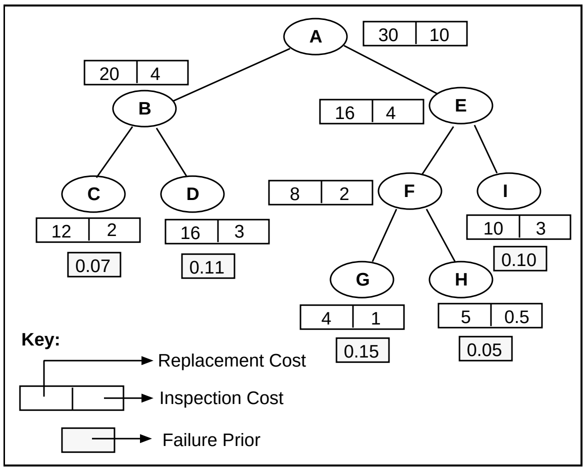


Figure 5.4: Computing hierarchical repair plans: An example of a hierarchical system model.

are related as follows:

$$EC(T) = P(X_0 = \mathbf{b}) \times EC^f(T) + P(X_0 = \mathbf{ok}) \times 0$$

Hence:

$$\begin{aligned} EC^f(T) &= \frac{EC(T)}{P(X_0 = \mathbf{b})} \\ &= \frac{EC(T)}{1 - P(M_{[1,n]} = \mathbf{ok})} \end{aligned} \tag{5.18}$$

Note that for any strategy T, $EC^f(T)$ and $EC(T)$ are related by the constant $\frac{1}{1-P(M_{[1,n]}=\mathbf{ok})}$. Thus, the strategy T^{opt} with the lowest possible value of EC is also the strategy with the lowest value of EC^f.

5.6 Hierarchical repair

We now extend the restricted formulation of the repair problem to include hierarchical systems. As we have seen in Chapter 2, hierarchies are ubiquitous in engineering practice. Chapter 2 developed a method to exploit hierarchy when performing diagnosis. In this section, we examine how hierarchy is exploited when computing optimal repair strategies.

A *hierarchical component model* consists either of an *atomic model* or a *subcomponent model*. If the component $\mathcal{C}$ is modeled atomically, we specify a probability of failure of the component p, a cost of replacement c and a cost of inspection d. If the component $\mathcal{C}$ is modeled as consisting of subcomponents, we do the following:

1. Specify hierarchical models (recursively) for each of the subcomponents $\mathcal{C}_i^s$ of $\mathcal{C}$.

2. Specify a cost of replacement c of $\mathcal{C}$ and an inspection cost d.

We assume that a component works normally iff all its subcomponents are working normally[4]. In other words, a component is in the **ok** state iff all its subcomponents are in the **ok** state. If any of the subcomponents are in the broken state, the component is assumed to be in the broken state. Note that the probability of failure of the component can easily be computed from the subcomponent probabilities. Also, note that the top level component in the hierarchy represents the entire system. A *hierarchical system model* is simply the hierarchical component model for this top level component. Figure 5.4 is an example of a hierarchical system model. The tree in the figure represents the hierarchy tree of the system. Each node represents a component. The replacement cost and inspection cost of each component are marked next to it. In addition, the prior probability of failure for each of the leaf level components is also specified. Note that the prior probability of failure of the non-leaf components in the hierarchy tree can be computed from the probabilities at the leaves.

We now define a *hierarchical repair plan*. A hierarchical repair plan for a component specifies an action that will repair a component if it has been observed and found to be broken. The action specified is either:

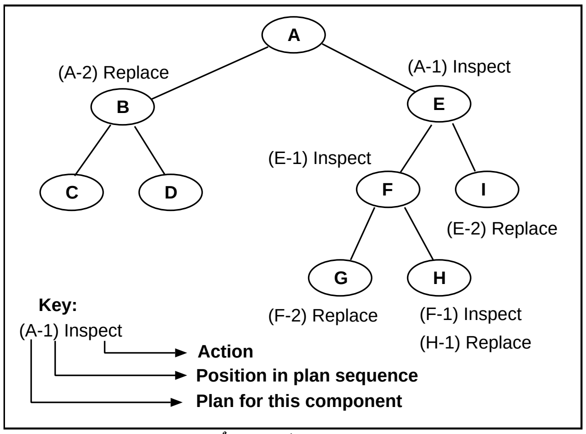
- Replacement of the entire component.

or

- A strategy for repair of the subcomponents. As we saw before, a strategy specifies an order in which to repair the subcomponents. In addition, it specifies whether each subcomponent is to be inspected before repair or not.

If a strategy specifies that a subcomponent is not to be inspected before repair, it is simply replaced. If a strategy specifies that the subcomponent is to be inspected, then the inspection procedure of the subcomponent is carried out before it is repaired. If the result of the inspection is that the subcomponent is **ok**, then the subcomponent needs no further attention.

[4]This corresponds to a specific choice of the *abstraction function* defined in Chapter 2.



$$EC^f(T^{opt}) = 18.90$$

Figure 5.5: An (optimal) hierarchical system repair plan.

If the result of the inspection is that the subcomponent is broken, then the subcomponent is repaired according to a hierarchical repair plan specified for the subcomponent. A hierarchical repair plan for a component thus includes the specification of a hierarchical repair plan for each of the subcomponents that are inspected by the plan.

Figure 5.5 is a possible hierarchical repair plan for the system. This plan specifies that if the system A is known to be faulty we first repair E after inspection and then, if A is still faulty, replace B without inspection.

The repair of E proceeds as follows: If E is found to be faulty after inspection we first repair F after inspection and then, if E is still faulty, replace I without inspection. If F is found to be faulty after inspection, we first repair H after inspection and then, if F is still faulty, replace G without inspection. If H is found to be faulty after inspection, it is replaced.

An optimal hierarchical repair plan for a component is the hierarchical repair plan with least expected cost. An *optimal hierarchical system repair plan* is simply the optimal hierarchical component repair plan for the top level component in the hierarchy tree. The repair plan shown in Figure 5.5 is also the optimal repair plan for the system.

5.6.1 Computing the optimal hierarchical plan

We will now describe a way of computing the optimal hierarchical repair plan for a component from the optimal hierarchical repair plans of its subcomponents. This procedure can then be used in a bottom-up traversal of the hierarchy tree to compute the optimal hierarchical system repair plan.

Say component $\mathcal{C}$ has k subcomponents $\mathcal{C}_1^s$, $\mathcal{C}_2^s$, ..., $\mathcal{C}_k^s$. The replacement cost of the component is c. Say the optimal hierarchical plan for each subcomponent $\mathcal{C}_i^s$ has already been computed and the cost of the plan is H_i^s.

We first compute an optimal strategy T^{opt} in which to fix the subcomponents $\mathcal{C}_i^s$. As we saw in Section 5.5.1, the optimal strategy T^{opt} and its cost $EC(T^{opt})$ can be computed in $O((k \log k)2^k)$. The computation takes into account the optimal hierarchical repair cost H_i^s, the inspection cost d_i^s and the replacement cost c_i^s for each of the subcomponents $\mathcal{C}_i^s$.

The cost estimate $EC(T^{opt})$ is the cost estimate for repairing $\mathcal{C}$ given no evidence. Consider the situation where $\mathcal{C}$ has been inspected and found to be broken. In this case, the cost estimate needs to be conditioned on this knowledge. As we saw in Section 5.5.2, the conditional cost estimate $EC^f(T^{opt})$ is given by:

$$\begin{aligned} EC^f(T^{opt}) &= \frac{EC(T^{opt})}{1 - P(M_{[1,n]} = \mathbf{ok})} \\ &= \frac{EC(T^{opt})}{1 - \Pi_{1 \le i \le n}(1 - p_i^s)} \end{aligned}$$

The optimal hierarchical plan specifies the optimal repair action (and accompanying cost) for a component given that it is broken. The two possible actions are: (a) replacement of the component and (b) repair of subcomponents. We can choose the better of the two options by simply comparing the replacement cost c and the optimal cost $EC^f(T^{opt})$ of repairing subcomponents.

If $c \le EC^f(T^{opt})$, then the optimal hierarchical repair plan for the component $\mathcal{C}$ is to simply replace it if it has been found to be broken. The cost of the optimal hierarchical component repair plan in this case is c. If we find that $EC^f(T^{opt}) > c$, then the optimal hierarchical repair plan is to follow strategy T^{opt}. The cost of the hierarchical component repair plan for component $\mathcal{C}$ in this case is $EC^f(T^{opt})$.

We note that we can compute the optimal hierarchical system repair plan by working up from the leaves of the hierarchy tree while computing the optimal component repair plan for each component. Say each component in the system can have at most k subcomponents. Let us suppose the system has n leaf level components in all. A tree with a branching factor of k with n leaf nodes has $O(n)$ nodes in the tree (including leaf nodes). So the complexity of computing the optimal hierarchical repair plan is $O(n(k \log k)2^k)$. Hence, for a fixed k, the optimal hierarchical repair plan can be computed in $O(n)$.

We have implemented the algorithm in Common Lisp. For testing purposes, we have also implemented a random system generator that creates a system hierarchy

k	d		
	3	4	5
3	17	50	117
4	83	233	833
5	167	934	4750

k : Branching Factor, d : Tree Depth
Run time in milliseconds on a Sun 10/40.

Table 5.1: Running time of optimal hierarchical repair plan algorithm.

with a user specified branching factor and a user specified tree depth. The repair costs, inspection costs and failure probabilities are chosen randomly from user specified intervals. The times taken to compute optimal repair strategies for systems of various sizes are shown in Table 5.1. The optimal policy for a system with a branching factor of 5 and a tree depth of 5 (i.e., with 3125 leaf level components) can be computed in about 5 seconds. Thus, the algorithm scales well to systems with thousands of components.

5.7 Discussion

In a general formulation of the repair problem, the pre-computation of an optimal repair strategy is intractable. The reason is that in a general formulation, there are no restrictions on the kind of system modeled. Since there is no special structure that we can take advantage of, we are reduced to considering each possible strategy in a combinatorial space of repair strategies to compute the optimal strategy.

There are two classes of approaches used to address this tractability problem. The first is to make some restricted formulation of the repair problem which is still applicable in some domain of interest. The properties of the restricted formulation can then be exploited to develop tractable algorithms to compute repair strategies with provable properties. Our work falls into this class. In the second class of approaches, the diagnosis/repair problem is formulated as an interactive process. At each stage of the process, an action that is to be carried out immediately is chosen with a greedy heuristic or limited lookahead. The chosen action is then carried out, and this leads to new information being obtained. This information is used to compute the next action to be carried out.

In [Kalagnanam and Henrion, 1990], the authors derive an optimality condition for the optimal repair strategy in a multi-component system which is assumed to have a single fault. The repair protocol is similar to ours with the exception that only component replacements are allowed. There is no notion of inspection of components. Our work generalizes their result to the case of multiple independent failures. It also introduces a formulation of component inspection and extends the scope of the algorithm to hierarchical systems.

Repair is formulated as an interactive process in [Heckerman *et al.*, 1995a]. The system is modeled with a Bayesian network and both component replacement and information gathering actions are possible. An action is chosen at each step of the process with a myopic heuristic. The heuristic computes the least cost action to take next assuming that the current fault in the system is a single fault. The restricted system behavior we have proposed corresponds to a restriction on the form of the Bayesian network in their framework. When no component inspections are allowed, we have developed a polynomial time algorithm for computing the optimal strategy. This algorithm is thus a tractable solution to a special case of the problem attacked by [Heckerman *et al.*, 1995a].

The optimality result of Equation 5.13 is potentially applicable within their framework as an improved heuristic for choosing actions myopically. Instead of assuming a single fault, the improved heuristic would allow for multiple faults.

The work in the model-based diagnosis community has also addressed the repair problem as an interactive process. [de Kleer and Williams, 1987] introduce an entropy based method for observation planning. [Friedrich and Nejdl, 1992] develop a set of greedy algorithms for choosing observation and repair actions in interactive model-based diagnosis. Their approach explicitly considers downtime costs in case of unanticipated failure. Hence, their repair scheme implicitly includes a notion of preventive maintenance. [Poole and Provan, 1991] use repair actions to partition the world into a set of classes. All the worlds in a class result in the same action response. The diagnosis problem now becomes one of determining which class the current state of the system falls into. The action response can then be looked up. [Sun and Weld, 1993] develop a system that uses partial order planning to generate repair plans. Repair plans include both component replacement and information gathering tests. The cost of each repair action is computed with a n-step lookahead. [Yuan, 1993] proposes a decision theoretic framework for modeling interactive model-based diagnosis. At each step of the diagnosis, a decision model in the form of an influence diagram is synthesized and solved to compute the next action. The model is successively refined along the system hierarchy using a single fault assumption until the fault is located.

The results of this chapter appear in [Srinivas, 1995b]. This work has been generalized in [Srinivas and Horvitz, 1995]. The latter paper directly extends the general formulation of repair introduced in Section 5.1 to hierarchical systems. A linear time algorithm for pre-computation of an optimal repair strategy is developed. A particular repair protocol is assumed: Repair begins when the system exhibits an anomalous output for some input. The repair process consists of successively repairing components of the system until the output is no longer anomalous for the same input. The hierarchy is exploited to gain tractability when computing the optimal repair strategy. The algorithm is tractable if the branching factor of the system hierarchy is small. The hierarchical algorithm we have developed in this chapter addresses a special case of this general formulation. The constant in the linear time algorithm for this special case is far smaller than the constant in the general formulation.

5.7.1 Dependent faults

In this chapter, we have concentrated on the situation where component failures are independent. Note however that the optimality condition (Equation 5.12) applies in the general case where the component failures may be dependent.

Consider the situation where a model for the dependencies between the failures of the components is available in the form of a Bayesian network B. Thus, B is a model for $P(M_1, M_2, \ldots, M_n)$. The optimality condition of Equation 5.12 can be simplified to:

$$\begin{aligned} & c_j[P(M_j = \mathbf{ok} \mid R_{\mathbf{ok}}) - P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok} \mid R_{\mathbf{ok}})] \\ \leq\ & c_{j+1}[P(M_{j+1} = \mathbf{ok} \mid R_{\mathbf{ok}}) - P(M_{j+1} = \mathbf{ok}, M_j = \mathbf{ok} \mid R_{\mathbf{ok}})] \end{aligned} \tag{5.19}$$

Given a repair sequence T, we can check whether the condition holds at the position j as follows. Declare evidence $R_{\mathbf{ok}}$ (i.e., $M_{[j+2,n]} = \mathbf{ok}$) in the network B. Do a network inference and look up the probabilities $P(M_j = \mathbf{ok}|R_{\mathbf{ok}})$ and $P(M_{j+1} = \mathbf{ok}|R_{\mathbf{ok}})$. Subsequently declare additional evidence $M_j = \mathbf{ok}$ and do another network inference to compute $P(M_{j+1} = \mathbf{ok}|M_j = \mathbf{ok}, R_{\mathbf{ok}})$. Note that the probability $P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok}|R_{\mathbf{ok}})$ can now be computed as:

$$P(M_j = \mathbf{ok}, M_{j+1} = \mathbf{ok}|R_{\mathbf{ok}}) = P(M_{j+1} = \mathbf{ok}|M_j = \mathbf{ok}, R_{\mathbf{ok}})P(M_j = \mathbf{ok}|R_{\mathbf{ok}})$$

We now have the quantities required to check whether the condition holds. Thus, the verification of the optimality condition at any point in the sequence T can be accomplished with 2 network inferences.

If the optimality condition does not hold at position j, flipping the position of $\mathcal{C}_j$ and $\mathcal{C}_{j+1}$ will lead to a better repair sequence. If we consider doing this repeatedly till quiescence is reached, then the resulting sequence will be a local optimum. A good starting sequence might the sequence T^{ind} computed assuming that the component failures are independent. This can be done by initially doing network inference with no evidence in the network. That gives us the priors $p_i = P(M_i = b)$ for every node in the network and thus the sequence sorted by increasing order of $c_i\frac{1-p_i}{p_i}$ can be computed.

However, certain questions still need to be addressed. Firstly, it is not clear that the sequence produced at quiescence is necessarily the global optimum. The second question is whether the number of network propagations required is tractable in the worst case.

With regard to the second question: the first naive estimate is that the number of propagations is $O(n!)$ since there are only $n!$ sequences. However, we can observe from the structure of Equation 5.10 that if k components have been fixed, then the optimal repair sequence of the remaining $n-k$ components does not depend on the order in which the first k components were fixed. This allows dynamic programming to be used to construct a scheme which will compute the optimal strategy with $O(n2^n)$ network propagations. This is still exponential.

Our speculation is that we cannot do better without more structure (for example, specific network topologies) in the problem. A promising direction seems to be to adapt an exact algorithm for computation of the optimal strategy in the case of dependent faults to have limited lookahead and anytime characteristics.

Chapter 6

Handling unspecified fault models

In model-based diagnosis, a component model specifies the behavior of the component both when it is working normally and when it is in an abnormal mode. The specification of component behavior in an abnormal mode is called a *fault model*. For example, a gate in a digital circuit might have an abnormal mode where the output is always 0 regardless of the input (such a mode is usually called *stuck-at-zero*).

Fault models are often not available—the modeler might not know enough about the device physics to describe how a component behaves when it is broken. Alternatively, the modeler might list some known modes of failure but want to leave open the possibility that the component fails in some unforeseen way. It is thus very important that unspecified fault models be handled in some coherent way during diagnosis.

Current practice in model-based diagnosis approaches this problem by assuming that all possible system outputs are equally likely when a component is in an abnormal mode. This, in effect, corresponds to a choice of a particular (non-deterministic) fault model for the component. In this chapter, we first examine current practice and show that it can lead to incoherent results. We then suggest a change in modeling practice to solve this problem.

A larger problem with the current approach is that it is *ad hoc*. There is no semantic justification for choosing "equally probable outputs" when the fault model is not known. To address this problem, we go on to develop an alternative method of handling unspecified fault models. The goal of diagnosis is to compute inexpensive repair and maintenance plans. In the presence of unspecified fault models, one can at best compute bounds on repair costs. Our method assumes that unspecified fault models behave such that the repair cost is maximized. Thus, our method takes a conservative approach to cost estimation in the presence of incomplete information.

We now examine the problem of diagnosis in the presence of unspecified fault models in more detail. As we have seen in Chapter 3, diagnosis involves computing a posterior probability distribution over the states of the system given an observation.

Say we make an observation $\Omega = \langle \mathbf{I} = \mathbf{i}, \mathbf{O} = \mathbf{o} \rangle$ of a system S. Ω consists of readings of the set of system input variables $\mathbf{I}$ and the corresponding readings of some set of component outputs $\mathbf{O}$. Given a candidate δ, we compute the posterior probability $P(\delta \mid \Omega, S)$ (to within a constant factor K) as follows:

$$\begin{aligned} P(\delta \mid S, \Omega) &= P(\delta \mid S, \mathbf{i}, \mathbf{o}) \\ &= \frac{P(\mathbf{o} \mid \mathbf{i}, \delta, S) P(\delta)}{P(\mathbf{o} \mid \mathbf{i}, S)} \\ &= K \times P(\mathbf{o} \mid \mathbf{i}, \delta, S) P(\delta) \end{aligned} \tag{6.1}$$

The above equations account for the fact that the choice of the system model S, the distribution over the system input $\mathbf{I}$ and the distribution over the candidates δ are all mutually independent. The prior probability of a candidate δ is computed simply by multiplying the priors of the component modes in the candidate. When there are n observations, Ω_1, Ω_2, ..., Ω_n, the posterior probability can be calculated with:

$$\begin{aligned} P(\delta \mid S, \Omega_1, \Omega_2, \ldots, \Omega_n) = \; & K \times P(\delta) \times \\ & P(\mathbf{o}_1 \mid \mathbf{i}_1, \delta, S) \times \\ & P(\mathbf{o}_2 \mid \mathbf{i}_2, \delta, S) \times \\ & \ldots \\ & P(\mathbf{o}_n \mid \mathbf{i}_n, \delta, S) \end{aligned}$$

The above equation accounts for the fact that knowing the system state renders the observations independent.

Consider the case where *all fault models are fully specified.* Given δ and $\mathbf{i}$, we can simulate the system forward and see whether the simulated output $\mathbf{o}^{sim}$ and the observed output $\mathbf{o}$ are the same. If they are the same, we have, $\mathbf{i}, \delta, S \models \mathbf{o}$, or equivalently, $P(\mathbf{o} \mid \mathbf{i}, \delta, S) = 1$. If they are not the same, we have $\mathbf{i}, \delta, S \not\models \mathbf{o}$, or equivalently, $P(\mathbf{o} \mid \mathbf{i}, \delta, S) = 0$.

Now consider the situation where all fault models are *not* fully specified, i.e., some component $\mathcal{C}_j$ has a mode u_j for which the fault model is not specified. Such a mode is referred to as an *unknown* mode in model-based diagnosis. Let δ_u be a candidate in which some component is in an unknown mode. Say we simulate the input forward assuming the state of the system is δ_u. When we encounter a component which is in the unknown mode we note that we cannot compute its output since the fault model is not known. In that case, we set the output to be the value "unknown". In addition, we set the output to be "unknown" for any component which has at least one input having the value "unknown". When we are done with the simulation, we see that variables in $\mathbf{O}$ can be split into two subsets. The subset $\mathbf{O}_p$ consists of those variables for which the simulation predicts a value. The subset $\mathbf{O}_u$ consists of those variables for which the output is "unknown".

Let us refer to the subset of readings corresponding to $\mathbf{O}_p$ in the observed output $\mathbf{o}$ as $\mathbf{o}_p$. Let the subset of readings corresponding to $\mathbf{O}_u$ be $\mathbf{o}_u$. Similarly, for the simulated output $\mathbf{o}^{sim}$ we have $\mathbf{o}_p^{sim}$ and $\mathbf{o}_u^{sim}$. We note that if $\mathbf{o}_p \neq \mathbf{o}_p^{sim}$, we have $\mathbf{i}, \delta_u, S \not\models \mathbf{o}$, or equivalently, $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S) = 0$.

Now consider the case where $\mathbf{o}_p = \mathbf{o}_p^{sim}$. In this case, we see that we have no way of predicting $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S)$ in Equation 3.1. In this situation, any technique for doing model-based diagnosis with unspecified fault models has to choose a value for $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S)$ in some explicit or implicit way.

6.1 Current practice

The technique that is most widely used currently for handling unspecified models was first suggested by [de Kleer and Williams, 1989]. Say we have a candidate δ_u that contains an unknown mode and an observation $\Omega = \langle \mathbf{I} = \mathbf{i}, \mathbf{O} = \mathbf{o} \rangle$ such that we cannot compute $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S)$. Say the subset $\mathbf{O}_u$ of the output variables $\mathbf{O}$ has M_u possible joint states. If $\mathbf{O}_u$ consists of only one output variable, then $M_u = m$ where m is the number of states of this output variable. If $\mathbf{O}_u$ consists of more than one output variable, $M_u = \Pi_i m_i$ where i ranges over the output variables in $\mathbf{O}_u$. The technique suggested by DeKleer and Williams chooses $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S)$ in Equation 6.1 as:

$$P(\mathbf{o} \mid \mathbf{i}, \delta_u, S) = \frac{1}{M_u} \tag{6.2}$$

In other words, the technique assumes that all possible outputs are equally probable. We will call this the *equiprobable output* (EPO) assumption.

6.1.1 How this technique can go wrong

Applying the EPO assumption can yield incoherent results. Specifically, there can be situations where there is no way to choose fault models for the components such that the probabilities postulated by the EPO assumption are attained.

We explain by means of an example. Consider the digital circuit shown in Fig 6.1. B_1 and B_2 are buffers that feed into an AND gate. Each of the buffers has two modes. When a buffer is **ok**, its output is equal to its input. The other mode is an unknown mode u. We see from the observation shown in the figure that something is wrong—if both buffers were **ok**, the output should be 1. We will assume that the AND gate always works perfectly, i.e., it has only one mode and that is the **ok** mode. A candidate in this system consists of a mode for B_1 and a mode for B_2. The mode of B_1 is described by a state variable M_1. Similarly, M_2 describes the mode of B_2.

We now examine what the EPO assumption entails. Say we make the EPO

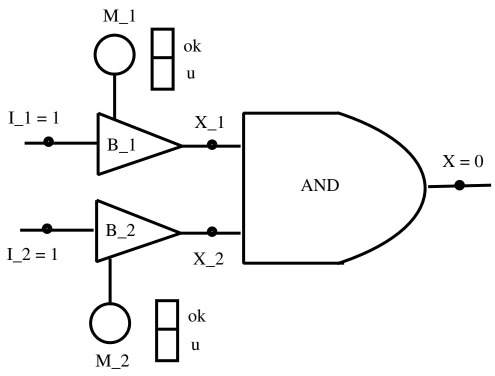


Figure 6.1: Incoherence of EPO assumption: An example.

assumption for the candidate $\delta_{(ok,u)} = \langle B_1 = \mathbf{ok}, B_2 = u \rangle$. The EPO assumption tells us that $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(ok,u)}) = \frac{1}{2}$. When $B_1 = \mathbf{ok}$ and $B_2 = u$, the only way we can have $X = 0$ is if $X_2 = 0$. This implies that the local fault model for B_2 behaves as $P(X_2 = 0 \mid I_2 = 1, M_2 = u) = \frac{1}{2}$. This implies $P(X_2 = 1 \mid I_2 = 1, M_2 = u) = 1 - \frac{1}{2} = \frac{1}{2}$.

Similarly, examining the EPO assumption for $\delta_{(u,ok)}$ gives us $P(X_1 = 0 \mid I_1 = 1, M_1 = u) = P(X_1 = 1 \mid I_1 = 1, M_1 = u) = \frac{1}{2}$.

We note that the fault model for the unknown mode of B_1 is now fully specified when the input to B_1 is 1. The fault model is not deterministic—it gives a probability distribution over the output of B_1 given the input is 1 and that B_1 is in the unknown mode. Similarly, the fault model for B_2 is fully specified when the input for B_2 is 1.

Now consider the candidate $\delta_{(u,u)}$. We can compute the probability $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)})$ from the fault models for B_1 and B_2 that were implied by the application of the EPO assumption to $\delta_{(ok,u)}$ and $\delta_{(u,ok)}$. We do this as follows:

$$
\begin{aligned}
& P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) = \\
& \quad\quad P(X_1 = 1, X_2 = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) + \\
& \quad\quad P(X_1 = 0, X_2 = 1 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) + \\
& \quad\quad P(X_1 = 0, X_2 = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) \\
& = \quad P(X_1 = 0 \mid I_1 = 1, M_1 = u) P(X_2 = 1 \mid I_2 = 1, M_2 = u) + \\
& \quad\quad P(X_1 = 1 \mid I_1 = 1, M_1 = u) P(X_2 = 0 \mid I_2 = 1, M_2 = u) + \\
& \quad\quad P(X_1 = 0 \mid I_1 = 1, M_1 = u) P(X_2 = 0 \mid I_2 = 1, M_2 = u) \\
& = \quad (\frac{1}{2} \times \frac{1}{2}) + (\frac{1}{2} \times \frac{1}{2}) + (\frac{1}{2} \times \frac{1}{2})
\end{aligned}
$$

$$= \frac{3}{4}$$

The above equation sums over mutually exclusive cases and takes advantage of the fact that the fault models of B_1 and B_2 behave independently.

Say that instead of computing $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)})$ as described above, we computed it by applying the EPO assumption *directly* to $\delta_{(u,u)}$. This would give us $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) = \frac{1}{2}$. Note that we have a contradiction. Our application of the EPO assumption to $\delta_{(u,ok)}$ and $\delta_{(ok,u)}$ implies that $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) = \frac{3}{4}$. However, applying the EPO assumption *directly* to $\delta_{(u,u)}$ tells us that $P(X = 0 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) = \frac{1}{2}$.

In summary, it is impossible to apply the EPO assumption to all the candidates δ_u for which we cannot compute $P(\mathbf{o} \mid \mathbf{i}, \delta_u, S)$ *while assuming that the fault models for each component are independent of each other*. The only way we can make the EPO assumption be coherent is to assume that the fault model of one component and the fault model of some other component are *not* independent.

An example of a fault model correlation that would fix the problem in this example would be to set $P(X_1 = x_1, X_2 = x_2 \mid I_1 = 1, I_2 = 1, \delta_{(u,u)}) = \frac{1}{6}$ for any values x_1 and x_2. However, this violates the modularity of components. Component modularity is one of the basic assumptions we make in model-based diagnosis (and in compositional modeling in general).

6.1.2 A fix for the problem

The EPO assumption's basic intuition is to assume all outputs are equally likely if a fault model is not available. Instead of applying it globally for the entire system, consider applying it locally to components instead.

Consider a component $\mathcal{C}$ whose mode variable is M. The component's inputs are the set of variables $\mathbf{I}$ and the output is X. We use x to refer to a state of X. We denote a (joint) state of the input variables as $\mathbf{i}$. Let m be the number of states of X. Applying the EPO assumption locally amounts to specifying that $P(X = x \mid \mathbf{I} = \mathbf{i}, M = u) = \frac{1}{m}$ for any x and any $\mathbf{i}$.

Basically, we are specifying a particular non-deterministic fault model for the mode u. This introduces a problem. Most model-based diagnosis systems can handle fully or partially specified deterministic fault models but cannot handle a probabilistic specification. We would like to make a fix which does not need a fundamental change in an existing logic-based model based diagnosis system.

One way around the problem is to transform the component description as follows. We split the mode u into m modes, one for each state of the output variable x. Let u_x denote the new mode corresponding to state x. The fault model of u_x is *stuck-at-x*. That is, if the component is in mode u_x the output is x regardless of the input. We set the prior of u_x to be $P(M = u) \times \frac{1}{m}$. The transformed description is basically equivalent to applying the EPO assumption locally. The only difference is that we do not have a single unknown mode—we have m "unknown" modes instead, each of which is deterministic. We now can apply any logic-based diagnosis method.

Applying the EPO assumption locally gives us coherence but comes at a computational cost. When we have a candidate with fully specified fault models we have seen before that we can compute the posterior probability with a simple forward simulation of the system (see Equation 6.1 and preceding discussion). If the candidate and observation are consistent, the posterior is simply the prior multiplied by a constant factor. If the candidate is inconsistent with the observations, the posterior is 0. The EPO assumption, as currently practiced, extends this desirable property to candidates which contain unknown modes. That is, we can compute the posterior probability with a single forward simulation. However, when applying the EPO assumption locally, we lose this property.

We demonstrate this with an example. Consider the digital circuit shown in Fig 6.1. Say we want to apply the EPO assumption locally and then compute the posterior probability (to within a constant factor) of $\delta_{(u,u)}$. Either explicitly or implicitly, we have to do this by summing over the posterior probabilities of $\delta_{(u_0,u_0)}$, $\delta_{(u_0,u_1)}$, $\delta_{(u_1,u_0)}$ and $\delta_{(u_1,u_1)}$. Each of these "subcandidates" have fully specified fault models and we can use the single forward simulation for each of these. Say that a candidate contains n components in the unknown state. Say each of these components has k outputs. In this situation, we will have to examine k^n "subcandidates" to compute the posterior probability of the candidate.

This problem can be solved by redefining our modeling approach to avoid this. Say that, instead of having a single unknown mode for a component, we model the component directly as having m "unknown" modes (where m is the number of possible values of the output of the component). In other words, we go directly with the results of the transformation approach as the original model. This would mean, for example, that we will consider $\delta_{(u_0,u_0)}$ and $\delta_{(u_0,u_1)}$ as distinct candidates of interest (this would not be the case if we had only one unknown mode u for each of these components).

Such a change in modeling approach may be permissible in some domains. For example, if one wants to be able to use a non-intermittency assumption (see [Raiman *et al.*, 1991], for example), it is necessary that all fault models be deterministic. This is to ensure that the output of the unknown mode for a given input is the same across multiple observations. In such a case, one might use this redefined modeling approach.

Though the local application of EPO assumption fixes the problem of incoherence, we still have a basic problem. The problem is that the EPO assumption is *ad hoc*. We do not understand what its semantics and ramifications are when doing diagnosis and repair.

6.2 A cost based approach to model completion

The goal of diagnosis is to compute cost effective repair and maintenance strategies for the system under diagnosis. A repair algorithm takes observations of the system as input and computes a decision tree of actions to be carried out. These actions may replace a component or perform testing (see Section 5.6). Given an observation, the corresponding decision tree has an expected cost of execution. This is the cost of repair conditioned on that observation.

In addition, a repair algorithm has an expected cost—the expected cost of the algorithm is the cost averaged over the long run of the system, i.e., the sum over all possible observations of the product of the expected cost of the decision tree for the observation multiplied by the probability of the observation. Say some of the components in a system have "unknown" modes. As we have seen before, we may not be able to compute posterior probabilities on candidates or on the component modes. As a result, we will not be able to compute the expected cost of a repair algorithm R. Hence, we have to make some choice for the behavior of the unknown modes if we want to estimate repair costs (as part of a search for a good repair strategy, for example).

Say we are able to make a choice U_h which will guarantee that the expected repair cost of any repair algorithm R is as high as possible. In other words, given a repair algorithm R, any other choice for the behavior of the unknown modes will result in a lower estimate for the expected repair cost of R than the choice U_h. We note that expected repair cost estimated after making the choice U_h for the unknown modes is a conservative bound. In other words, irrespective of the actual behavior of the unknown modes, the actual expected repair cost will be less than the estimated expected repair cost.

6.2.1 Fixing components

We now consider the *fix* actions which are part of the repair decision tree. A fix action operates on a component and guarantees that the component is in the **ok** state after completion. We will develop a particular model for how the fix actions are performed. We will then examine the contribution of fix actions to the cost of a repair algorithm.

Consider, for now, the situation where there are no unknown fault models. Say a component has a set of inputs I[1] and an output O. We fix the component as follows:

1. We observe the component's input and output. Say the component's mode variable is M. Observing the input (vector) $I = i$ and output $O = o$ gives us a

[1]To aid readability, henceforth we will use I (without bold font) to refer to the input variables of the component and i to refer to a generic joint state of these variables.

distribution $P(M|I = i, O = o)$.

2. Using the posterior $P(M|I = i, O = o)$ as a starting estimate of the mode, we perform tests of the component to discover which mode it actually is in. Say the mode is k.

3. We then repair the component at cost $Cost(k)$. So, for example, if k is an abnormal mode, then $Cost(k)$ is the actual cost taken to repair the component given it is in the mode. If k is the **ok** mode, then $Cost(k) = 0$.

We now consider the cost of fixing the component given that a particular input i and a particular output o have been observed. Say each of the tests we perform in Step 2 above are all of equal cost. Let this cost be $Cost_t$. Given a probability distribution $P(X|\epsilon)$, the entropy of the distribution $H(X|\epsilon) = -\Sigma_x P(X = x|\epsilon)\log(P(X = x|\epsilon))$ is a lower bound on the number of tests required to determine the actual state of X (see [Cover and Thomas, 1991]). Hence, a lower bound on the test cost in Step 2 is $Cost_t \times H(M|I = I, O = o)$. The total cost of fixing the component is then bounded below by:

$$\begin{aligned} LbCost_{fix}(I = I, O = o) &= Cost_t \times H(M|I = I, O = o) + \\ &\quad \Sigma_k Cost(k)P(M = k|I = I, O = o) \end{aligned}$$

Now consider the cost of fixing the component in the long run using the repair algorithm R. We compute the amount contributed to the expected cost of algorithm R by the action of fixing the component. When using the repair algorithm R, we will encounter some joint distribution of the inputs of the component and the output of the component $P(I, O)$. The long run cost of fixing the component is bounded below by:

$$\begin{aligned} LbCost^{avg}_{fix} &= \Sigma_i\Sigma_o LbCost_{fix}(I = I, O = o)P(I = I, O = o) \qquad (6.3) \\ &= Cost_t \times \Sigma_i\Sigma_o H(M|I = I, O = o)P(I = I, O = o) + \\ &\quad \Sigma_i\Sigma_o\Sigma_k Cost(k)P(M = k|I = I, O = o)P(I = I, O = o) \end{aligned}$$

We define $H(M|IO) = \Sigma_i\Sigma_o H(M|I = I, O = o)P(I = I, O = o)$. The quantity $H(M|IO)$ is called the conditional entropy of M given I and O (see [Cover and Thomas, 1991]). Simplifying Equation 6.3, we have:

$$LbCost^{avg}_{fix} = Cost_t \times H(M|IO) + \Sigma_k Cost(k)P(M = k) \qquad (6.4)$$

Now say a component has an unknown mode u. We define the behavior of u as the specification of $P(O|I, M = u)$. $P(O|I, M = u)$ is the conditional distribution over the output given each input, assuming the component mode is u. Note that we leave open the possibility that u behaves intermittently. The user has specified a marginal distribution over the modes of the component. However, he or she has not specified how the component behaves when it is in mode u.

For each component with an unknown mode, say we choose $P(O|I, M = u)$ such that $LbCost^{avg}_{fix}$ of the component is maximized. This corresponds to making the expected repair cost of strategy R as high as possible.

Looking at Equation 6.4, we see that the second term in the right hand side is just the decision theoretic minimum repair cost of the component and is a constant unaffected by the choice of $P(O|I, M = u)$. The first term, however, is affected by the choice of $P(O|I, M = u)$. In fact, it can be computed only if we make a choice $P(O|I, M = u)$. Say we choose $P(O|I, M = u)$ such that $H(M|I, O)$ is maximized. This would also maximize $LbCost^{avg}_{fix}$.

6.2.2 The maximum cost fault model

As we shall prove in Section 6.2.4, the following choice of $P(O = o|I = i, M = u)$ maximizes $H(M|I, O)$:

$$P(O = o|I = i, M = u) = \frac{1}{1 - P(M = u)} \Sigma_{k \neq u} P(O = o|I = i, M = k) P(M = k) \tag{6.5}$$

Thus, we can compute a description of the fault model for u from the descriptions of the behavior of the component in the other modes (i.e., $P(O = o|I = i, M = k)$ where $k \neq u$) and the prior distribution over the modes. If we use this choice of fault model when we do diagnosis, the expected cost estimates we make will be conservative estimates. We note that the fault model chosen by this method can be intermittent. That is, the probabilities $P(O = o|I = i, M = u)$ need not be 0 or 1.

One way of interpreting this result is as follows. Since $P(I)$ and $P(M)$ are independent a priori, we note the following:

$$\begin{aligned}
P(O = o|I = i) &= \Sigma_k P(O = o|I = i, M = k) P(M = k) \\
&= (\Sigma_{k \neq u} P(O = o|I = i, M = k) P(M = k)) + \\
&\quad P(O = o|I = i, M = u) P(M = u) \\
&= [1 - P(M = u)] P(O = o|I = i, M = u) + \quad \text{(Using Eqn 6.5)} \\
&\quad P(M = u) P(O = o|I = i, M = u) \\
&= P(O = o|I = i, M = u)
\end{aligned}$$

Let us examine the probability $P(M = u|I = i, O = o)$. We have:

$$\begin{aligned} P(M = u|I = i, O = o) &= \frac{P(O = o|I = i, M = u)P(M = u)}{P(O = o|I = i)} \\ &= P(M = u) \end{aligned}$$

Thus, observing an input and output has no affect on our estimate on the probability of the unknown mode, i.e., the posterior is equal to the prior. In effect, this choice for the behavior of u allows us to conclude nothing about the unknown mode based on observations. The observations only affect the posterior probabilities of the other modes.

Another way of looking at this is as follows—the right hand side of Equation 6.5 is exactly the probability $P(O = o|I = i, M = \neg u)$. The unknown mode thus behaves exactly like the component would (on the average) if the component was not in the unknown mode. Thus, we see that observations cannot distinguish between the situations "Component is in the unknown mode" and "Component is in some other mode". However, observations do help us distinguish between the known modes.

6.2.3 A special case

Consider the situation where the component has one **ok** mode, one unknown mode and no other modes. In this situation if we apply Equation 6.5 we find that:

$$\begin{aligned} P(O = o|I = i, M = u) &= \frac{1}{1 - P(M = u)} \Sigma_{k \neq u} P(O = o|I = i, M = k)P(M = k) \\ &= \frac{1}{P(M = \mathbf{ok})} P(O = o|I = i, M = \mathbf{ok})P(M = \mathbf{ok}) \\ &= P(O = o|I = i, M = \mathbf{ok}) \end{aligned}$$

That is, the unknown mode behaves exactly like the **ok** mode. As a result, no anomalous observation is possible (since the unknown mode is identical to the **ok** mode). If a diagnostic system using this model for the unknown mode was presented with an anomalous observation for the component, a contradiction would result since the model does not allow for such an observation.

In general, given a particular input i, we see from Equation 6.5 that the model for the unknown mode has non-zero probabilities for only those outputs which occur for at least one of the known modes. Since the intent of the unknown mode is to catch unforeseen situations, we would like the unknown mode to have a non-zero probability (however small) for every possible output regardless of what the input is.

We can achieve this result by using the following modeling practice: For every output x, introduce a *stuck-at-x* mode (if one does not already exist) in the component model with a very small prior probability. The probability can be vanishingly small. The perturbation of the original component model is hence very small. The behavior of the unknown mode $P(O = o|I = i, M = u)$ is calculated after these additional modes have been introduced.

In fact, the diagnostic system need not explicitly introduce these modes. We can achieve the same end result with the following procedure: A *stuck-at-x* mode is introduced only when an observation with output x has occurred such that none of the non-unknown modes is consistent with the observation. If this situation occurs, the posterior probability of the unknown mode is still $P(M = u)$. The posterior probabilities of all other modes are zero except for the newly introduced *stuck-at-x* mode whose posterior probability is $1 - P(M = u)$.

6.2.4 Deriving the maximum cost fault model

We will now prove that maximizing $H(M|I, O)$ leads to Equation 6.5 as the choice for the fault model for the unknown mode u.

Conditional entropy satisfies the following identities:

$$\begin{aligned} H(Y|X) + H(X) &= H(X|Y) + H(Y) \\ H(Y|X) &= H(Y) \quad \text{when } Y \text{ and } X \text{ are independent} \end{aligned}$$

These identities can be verified directly from the definition of conditional entropy [Cover and Thomas, 1991]. In this section, we will use the notation $P(x)$ to denote $P(X = x)$. The variable being referred to will be obvious from context.

Applying the first identity in two different sequences for the variables M, we have:

$$H(M|I, O) + H(O|I) + H(I) = H(O|I, M) + H(M|I) + H(I) \tag{6.6}$$

We note that the input distribution $P(I)$ over the component is marginally independent of the distribution over the mode $P(M)$. Hence $H(M|I) = H(M)$. From Equation 6.6 we therefore have:

$$H(M|I, O) = H(M) + H(O|I, M) - H(O|I) \tag{6.7}$$

We see from Equation 6.7 that when we have a choice for $P(O|I, M = u)$, we can maximize $H(M|I, O)$ by simply maximizing $H(O|I, M) - H(O|I)$. This is because $H(M)$ is independent of the choice of $P(O|I, M = u)$. $H(M)$ is determined solely by the prior distribution over the component modes.

Consider the term $H(O|I, M)$. We have:

$$H(O|I, M) = \Sigma_k \Sigma_i H(O|I = i, M = k) P(i) P(k)$$

$$\begin{aligned}
&= \Sigma_k H(O|I, M = k)P(k) \\
&= H(O|I, M = u)P(u) + \Sigma_{k \neq u} H(O|I, M = k)P(k)
\end{aligned}$$

Note that $\Sigma_{k\neq u} H(O|I, M = k)P(k)$ is a constant which is independent of the choice of $P(O|I, M = u)$. Hence, the maximization of $H(O|I, M) - H(O|I)$ reduces to the maximization of $F = [H(O|I, M = u)P(u) - H(O|I)]$. We simplify the expression for F:

$$\begin{aligned}
F &= \{\Sigma_i [\Sigma_o - P(o|i, u)\log(P(o|i, u))] P(i)P(u)\} - \\
&\quad \{\Sigma_i [\Sigma_o - P(o|i)\log(P(o|i))] P(i)\} \\
&= \Sigma_i \Sigma_o [P(o|i)\log(P(o|i)) - P(o|i, u)\log(P(o|i, u))P(u)] P(i)
\end{aligned}$$

We have to choose the value of $P(o|i, u)$ for every combination of i and o such that F is maximized. Consider the derivative of F with respect to $P(o|i, u)$. We have:

$$\begin{aligned}
\frac{dF}{dP(o|i, u)} &= P(i)[1 + \log(P(o|i))]\frac{dP(o|i)}{dP(o|i, u)} \\
&\quad - P(i)[1 + \log(P(o|i, u))]P(u)
\end{aligned}$$

$P(o|i)$ can be calculated as:

$$P(o|i) = \Sigma_{M=k} P(o|i, k)P(k) \tag{6.8}$$

Hence:

$$\frac{dP(o|i)}{dP(o|i, u)} = P(u)$$

Hence, we have:

$$\begin{aligned}
\frac{dF}{dP(o|i, u)} &= P(i)[1 + \log(P(o|i))]P(u) - P(i)[1 + \log(P(o|i, u))]P(u) \\
&= P(i)P(u)\log(\frac{P(o|i)}{P(o|i, u)})
\end{aligned}$$

Setting the derivative of F with respect to $P(o|i, u)$ to zero for each combination of o and i, we get:

$$P(o|i, u) = P(o|i)$$

Using Equation 6.8 for the right hand side of the above equation gives us:

$$P(o|i, u) = \Sigma_{M=k} P(o|i, k)P(k)$$

Hence, we have the answer discussed earlier in Equation 6.5 of Section 6.2.2:

$$P(o|i,u) = \frac{1}{1-P(u)}\Sigma_{k\neq u}P(o|i,k)P(k)$$

It can be verified that this is a maximum point for the function F.

6.3 Discussion

When performing model-based diagnosis in a deterministic framework (i.e., without probabilities), the diagnosis process consists of computing candidates consistent with the observations. When unknown modes are present, the consistency check can still be carried out without a model of the behavior of the unknown mode. However, since the unknown modes are unconstrained in their behavior, there are a large number of candidates consistent with any observation. Introducing fault models improves the predictive power of the system model. Hence, the discriminatory power of the diagnosis process is increased and the number of candidates consistent with observations becomes smaller.

As a result, there has been research on how deterministic fault models can be modeled and exploited during the diagnosis process. [Struss and Dressler, 1989] describe how one can integrate *predictive fault models* of a component into diagnosis. A predictive fault model specifies the input-output behavior for a component for a mode k which is neither the **ok** mode or the unknown mode. [Freidrich *et al.*, 1990] introduce the notion of physical impossibility to constrain the faulty behavior of a component. The idea here is that even though we might not have full knowledge of the behavior of a component when it is broken, we can rule out certain behaviors as impossible when it is broken. They show that this can be considered as equivalent to partially specifying fault models.

In [Raiman *et al.*, 1991], the authors take a different approach to the problem. They assume that no fault model is available, but employ a non-intermittency assumption to reduce the number of possible diagnoses when working with devices which have unknown modes. The basic assumption is that though the mode's behavior is not known, it is deterministic and so can be expected to give the same output for the same input across different observations.

When moving to probabilistic diagnosis, we need to compute posterior probabilities on the candidates. As we have seen, to compute the posterior we need to assume some model of behavior of the unknown mode. This problem was first recognized in [de Kleer and Williams, 1989] and the EPO assumption was introduced to handle it. This is the technique most widely used today. As we have seen, this technique can be incoherent. We provide a simple fix where we apply the same underlying intuition locally at the components instead of globally over the system. However, there still remains the problem that the ramifications of the EPO assumption are not well understood. To address this problem, we propose choosing the behavior of the unknown mode such that the costs estimated by a repair strategy are conservative, i.e., irrespective of the actual behavior of the unknown mode, the actual cost we will encounter are smaller than our estimated cost.

Chapter 7

Conclusions

In this thesis, we have addressed some crucial problems that are encountered in the practical application of probabilistic model-based diagnosis. We now present a summary of the contributions of this thesis. We then move on to discuss future work.

7.1 Summary and Contributions

As we saw in Chapter 1, some major problems facing model-based diagnosis are:

- How do we do diagnosis tractably in large systems?
- How do we assess the prior probability of failure of components?
- How do we tractably compute inexpensive repair actions using the results of our diagnoses?
- How do we do diagnosis in a reasonable way if fault models are not available?

This thesis addresses each of the above questions. To address the tractability issue, we have developed two techniques. The first technique introduces a notion of hierarchy in the system specification. Hierarchies are widely prevalent in engineering practice and hence, the notion of a hierarchical system specification is a very natural one. We have developed a method of translating a hierarchical system specification into a hierarchical Bayesian network. Using the hierarchical Bayesian network, we have developed a method of compiling "atomic" descriptions of components from the descriptions of subcomponents. The compiled descriptions are utilized to develop a hierarchical diagnosis algorithm. The hierarchical diagnosis algorithm allows us to focus the computation on the parts of the system we are interested in, and this focusing gives significant computational gains. For example, the computation of the posterior probability of a leaf level component can be done in time linear in the depth of the hierarchy tree. If the hierarchy is not utilized during the computation, the complexity is exponential in the depth of the hierarchy tree.

Our second technique for tractable diagnosis improves on search-based diagnosis algorithms. We have developed a suite of generate-and-test diagnosis algorithms that are extremely simple to implement. The algorithms are specified independently of any particular implementation scheme for model-based diagnosis (such as an ATMS). They come with provable guarantees on complexity and run extremely fast in practice.

The first algorithm in the suite makes the usual assumption that component failures are independent. The algorithm successively generates all possible candidates in decreasing order of prior probability. Each candidate is tested for consistency with the observation. Consistent candidates are valid diagnoses. The complexity of generating each successive candidate is linear in the number of components in the system. The second algorithm improves on the first algorithm by using the information from previous consistency checks to rule out some of the candidates which have not yet been generated. This algorithm has performance which compares with (and in some cases, is better than) the best systems fielded today. We then develop a third algorithm which extends the generate-and-test idea to systems with dependent component failures. Component dependencies are specified as a Bayesian network. The candidate generator that we have developed computes the k-th most probable candidate in $O(Bk)$ time where B quantifies the size of the Bayesian network describing the fault dependencies.

The specification of prior probabilities of failure is a crucial step in probabilistic model-based diagnosis. These probabilities are often very difficult to assess. One of the main reasons for this difficulty is an implicit notion of a time interval underlying the probability specification. We have used concepts from Reliability Theory to make this time interval explicit in the specification of the prior. The model we have developed automatically computes the failure prior from an empirical measure of reliability of a component (the Mean Time Between Failures) and the up-time of the component. Thus, when empirical reliability data is available, it can be directly plugged into the diagnosis system specification.

We go on to use the same framework to also quantify persistence of system state over time. This allows us to develop a method for computing diagnoses with multiple observations, each of which occurs at a different time. Our method explicitly allows for possible component failures in between observations.

The ultimate goal of model-based diagnosis is to recommend cost effective actions to repair or maintain the system. To address this objective, we have developed a general algorithm for computing the least cost repair strategy. However, this algorithm is not practical for large systems since it examines all possible strategies to determine the best possible strategy. To gain tractability, we make a reasonable assumption about the behavior of the system—we assume that if any component has failed there will be some observed anomaly in the system. That is, the system works correctly if and only if all the components are normal. Given this assumption, we have developed a polynomial algorithm to compute the optimal repair strategy. We have also extended this algorithm to hierarchical systems. This extension explicitly considers the choice between replacing entire subsystems as against dis-assembling them and replacing subcomponents. The implemented algorithm is able to compute repair strategies for systems with thousands of components in a few seconds.

When modeling a system, fault models for components are often not available. However, these fault models are necessary for the diagnosis computation. Currently, the wide spread practice in model-based diagnosis is to assume that all system outputs are equally likely when any component is in a mode which does not have a fault model. In this thesis, we have shown how this assumption can lead to incoherent results. A more serious problem is that this assumption is *ad hoc*—its ramifications are not well understood. We have developed an alternative method of assinging a behavior to unspecified fault models. In our method, we choose the behavior of fault models in a way that guarantees that estimates of system repair costs are conservative upper bounds of the actual repair costs.

7.2 Future work

There are a number of directions in which the work described in this thesis can be extended.

An immediate extension that is of interest is a tight integration of the two complementary methods of addressing the tractability problem that are developed in this thesis. Thus, ideas from the candidate generation schemes of Chapter 3 may be applicable to improve the performance of the hierarchical Bayesian network inference algorithm developed in Chapter 2. Such an extension would be in the spirit of [Poole, 1993]. Poole's work discusses how conflicts can be used to develop a search-based algorithm to compute posterior probabilities in Bayesian networks.

The work in this thesis, and in model-based diagnosis in general, has mostly been confined to discrete valued variables. An extension of the techniques in this thesis to system models which have continuous valued variables would be of great interest. However, allowing arbitrary continuous valued functions as component models may preclude development of general purpose diagnosis algorithms. The reason is as follows: The computation of posterior probabilities during the diagnosis process involves the application of Bayes' rule. During the application of Bayes' rule, one has to perform marginalization. With continuous variables, marginalization corresponds to integrating the product of two or more functions. If arbitrary forms are allowed for the continuous functions in the model, then closed form solutions of the integrals may not be possible. Hence, some restrictions on the class of functions allowed (for example, linear combinations of inputs) may be necessary. In this regard, extensions to Bayesian networks that handle continuous values [Shachter and Kenley, 1989; Driver and Morrell, 1995] would be of great interest.

All the results in this thesis are confined to systems which are essentially static. Our work on handling multiple observations does introduce a very restricted notion of dynamics—components may fail (i.e., change state) on-line. It would be very useful to include a full notion of dynamic systems, i.e., systems in which components can have internal state (other than the mode). The component model would describe how the internal state evolves as a function of the component input and the previous state. It may be possible to represent the evolution of a dynamic system as a dynamic network (as in Section 4.2.3). Diagnosis would then reduce to inference within this dynamic network.

As we have seen earlier, prior probabilities of failure are often hard to assess. We have suggested one solution in this thesis—instead of specifying the priors directly, we provide estimates of reliability like the MTBF. In extremely reliable systems, even such estimates may not be available. However, it is possible that an expert would be able to compare the reliability of different components and group them by their "class" of reliability (e.g., "very reliable", "very very reliable").

A plausible interpretation for the reliability "class" is provided by the qualitative probability calculus of [Goldszmidt, 1992]. This calculus assumes that all probabilities are extremal (i.e., approach 0 or 1). In terms of our intended application, the qualitative probability calculus sets the probability of a component being broken to $P(\textbf{broken} = \epsilon^n)$ where $\epsilon \rightarrow 0$. Thus, the probability that a component is broken is very small. The exponent n signifies an order of magnitude of how small it is. Thus, one could plausibly think of $n = 1$ being equivalent to "very reliable" and $n = 2$ being equivalent to "very very reliable" and so on.

The qualitative probability calculus has exact analogs of operations like Bayesian conditioning. Thus, a diagnosis algorithm developed originally for numerical probabilities can often be directly adapted to work with the order of magnitudes that characterize qualitative probabilities. In fact, the adapted algorithm is potentially more efficient than the original algorithm. Adapting the diagnosis algorithms from this thesis to work with qualitative probabilities is thus a topic of great interest.

The polynomial time repair algorithm we have developed in this thesis makes a strong assumption about how the system behaves, viz, that if a component fails, the system will certainly display an anomaly. If we do not want to make any such restrictions on the system model, we are forced to search a large space to find an optimal strategy. It would be useful to find a search method which has an anytime property such that the search can be interrupted at any time to yield a current best strategy. Further, the more time the search is given the better the result.

We sketch one possible way of organizing such a search. Consider a particular search method which only considers the various possible sequences in which the first k components of an n component system are replaced. After the first k components are replaced, say that it is assumed that the remaining $(n-k)$ are replaced in some predetermined order (for example, least cost first). If we are able to successively search for the best strategy with $k=1$ initially, followed by $k=2$ and so onwards, we will eventually find the best strategy (when $k=n$). A nice property of such a search would be that if it is interrupted at any time (say when $k=m$ where $m<n$), there would still be a current best answer (i.e., the current best strategy with $k=m$).

In this regard, [Srinivas and Horvitz, 1995] suggests a different technique for achieving the goal of computing optimal repair strategies tractably without restricting the system model. They develop a means of exploiting the system hierarchy to achieve tractability. In addition, the resulting algorithm is modified to have anytime characteristics.

Repair algorithms with anytime characteristics could be further extended to reason about computing more versus acting immediately [Horvitz, 1988]. The tradeoff here would be between the savings which would come from computing more and coming up with a better strategy and the cost of the time it takes to compute the better strategy (i.e., downtime cost of the system).

Finally, it would be interesting from the perspective of the work in this thesis, and model-based diagnosis in general, to find new domains in which model-based diagnosis may be applicable and provide impact. In this regard, the domain of discrete-event systems seems to be very promising.

Bibliography

[Balke and Pearl, 1994] Balke, A. and Pearl, J. 1994. Counterfactual probabilities: Computational bounds, methods and applications. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*. Morgan Kaufmann, San Mateo, Calif., 1994. 46–54.

[Brglez and Fujiwara, 1985] Brglez, F. and Fujiwara, H. 1985. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. Distributed at the Special Session on ATPG and Fault Simulation, *Int. Symp. on Circuits and Systems.*

[Buchanan and Shortliffe, 1984] Buchanan, B. and Shortliffe, E. H., editors 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley.

[Charniak, 1991] Charniak, E. 1991. Bayesian networks without tears. *AI Magazine* 12(4):50–63.

[Collins and DeCoste, 1991] Collins, J. W. and DeCoste, D. 1991. CATMS: An ATMS which avoids label explosions. In *Proc. National Conference on Artificial Intelligence (AAAI '91)*. AAAI Press, Menlo Park, Calif., 1991. 281–287.

[Cover and Thomas, 1991] Cover, T. M. and Thomas, J. A. 1991. *Elements of Information Theory.* Wiley-Interscience.

[Dagum *et al.*, 1992] Dagum, P.; Galper, A.; and Horvitz, E. 1992. Dynamic network models for forecasting. In *Proc. Eighth Annual Conference on Uncertainty Artificial Intelligence (UAI '92)*. Morgan Kaufmann, San Mateo, Calif., 1992. 41–48.

[Darwiche and Goldszmidt, 1994] Darwiche, A. and Goldszmidt, M. 1994. Action Networks: A framework for reasoning about actions and change under uncertainty. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*. Morgan Kaufmann, San Mateo, Calif., 1994. 136–144.

[Darwiche, 1995] Darwiche, A. 1995. Model-based diagnosis using causal networks. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*. 1995.

[Davis and Hamscher, 1988] Davis, R. and Hamscher, W. C. 1988. Model-based reasoning: Troubleshooting. In Shrobe, H. E., editor 1988, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence.* Morgan Kaufmann, San Mateo, Calif., 1988. 297–346. Also in [Hamscher *et al.*, 1992]. 3–24.

[de Dombal *et al.*, 1972] de Dombal, F. T.; Leaper, D. J.; Staniland, J. R.; McCann, A. P.; and Horrocks, J. C. 1972. Computer aided diagnosis of acute abdominal pain. *British Medical Journal* 2:9–13.

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

[de Kleer and Williams, 1989] de Kleer, J. and Williams, B. C. 1989. Diagnosis with behavioral modes. In *Proc. Eleventh International Joint Conference on Artificial Intelligence (IJCAI '89)*. 1989. 1324–1330.

[de Kleer, 1986] de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28(1):127–162.

[de Kleer, 1991] de Kleer, J. 1991. Focusing on probable diagnoses. In *Proc. National Conference on Artificial Intelligence (AAAI '91)*. AAAI Press, Menlo Park, Calif., 1991. 842–848.

[de Kleer, 1994] de Kleer, J. 1994. A hybrid truth maintenance system. Technical report, Xerox PARC, Palo Alto, CA.

[Dean and Kanazawa, 1989] Dean, T. and Kanazawa, K. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.

[Dressler and Farquhar, 1990] Dressler, O. and Farquhar, A. 1990. Putting the problem solver back in the driver's seat: Contextual control of the ATMS. In *Lecture Notes in Artificial Intelligence 515*. Springer-Verlag, 1990.

[Driver and Morrell, 1995] Driver, E. and Morrell, D. 1995. Continuous Bayesian networks. In *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95)*. Morgan Kaufmann, San Mateo, Calif., 1995.

[Forbus and de Kleer, 1988] Forbus, K. D. and de Kleer, J. 1988. Focusing the ATMS. In *Proc. National Conference on Artificial Intelligence (AAAI '88)*. AAAI Press, Menlo Park, Calif., 1988. 193–198.

[Freidrich *et al.*, 1990] Freidrich, G.; Gottlob, G.; and Nejdl, W. 1990. Physical impossibility instead of fault models. In *Proc. National Conference on Artificial Intelligence (AAAI '90)*. AAAI Press, Menlo Park, Calif., 1990. 331–336.

[Friedrich and Nejdl, 1992] Friedrich, G. and Nejdl, W. 1992. Choosing observations and actions in model-based diagnosis/repair systems. In *The Third International Workshop on Principles of Diagnosis*. October 1992.

[Geffner and Pearl, 1987] Geffner, H. and Pearl, J. 1987. Distributed diagnosis of systems with multiple faults. In *Proc. Third IEEE Conference on AI Applications*, Kissimmee, Fla. Also in [Hamscher *et al.*, 1992]. 453–459.

[Genesereth, 1984] Genesereth, M. R. 1984. The use of design descriptions in automated diagnosis. *Artificial Intelligence* 24(1):411–436. Also in [Hamscher *et al.*, 1992]. 328–340.

[Goldszmidt, 1992] Goldszmidt, M. 1992. *Qualititative Probabilities: A Normative Framework for Commonsense Reasoning*. Ph.D. Thesis. Technical Report R-190, Cognitive Systems Laboratory, University of California, Los Angeles, Calif.

[Gorry and Barnett, 1968] Gorry, G. A. and Barnett, G. O. 1968. Experience with a model of sequential diagnosis. *Computers and Biomedical Research* 1:409–507.

[Hamscher *et al.*, 1992] Hamscher, W.; Console, L.; and de Kleer, J., editors 1992. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, Calif.

[Hamscher, 1991] Hamscher, W. C. 1991. Modeling digital circuits for troubleshooting. *Artificial Intelligence* 51(1-3):223–271. Also in [Hamscher *et al.*, 1992]. 283–308.

[Heckerman and Shachter, 1994] Heckerman, D. and Shachter, R. 1994. A Decision-based view of causality. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*. Morgan Kaufmann, San Mateo, Calif., 1994. 302–310.

[Heckerman *et al.*, 1990] Heckerman, D.; Horvitz, E.; and Nathwani, B. 1990. Update on the Pathfinder project. In *Proc. Thirteenth Symposium on Computer Applications in Medical Care*, Silver Spring, MD. IEEE Computer Society Press. 203–207.

[Heckerman *et al.*, 1995a] Heckerman, D.; Breese, J.; and Rommelse, K. 1995a. Decision-theoretic troubleshooting. *Communications of the ACM* 38(3):49–57.

[Heckerman *et al.*, 1995b] Heckerman, D.; Mamdani, A.; and Wellman, M. P. 1995b. Real world applications of Bayesian networks. *Communications of the ACM* 38(3):24–30.

[Horvitz and Heckerman, 1986] Horvitz, E. and Heckerman, D. 1986. The inconsistent use of measures of certainty in Artificial Intelligence research. In Kanal, L. N. and Lemmer, J. F., editors 1986, *Uncertainty in Artificial Intelligence 2*. North Holland, Amsterdam, 1986. 137–151.

[Horvitz *et al.*, 1988] Horvitz, E.; Breese, J.; and Henrion, M. 1988. Decision Theory in Expert Systems and Artificial Intelligence. *International Journal of Approximate Reasoning* 2:247–302.

[Horvitz, 1988] Horvitz, E. 1988. Reasoning under varying and uncertain resource constraints. In *Proc. National Conference on Artificial Intelligence (AAAI '88)*. AAAI Press, Menlo Park, Calif., 1988. 111–116.

[Jensen *et al.*, 1990] Jensen, F. V.; Lauritzen, S. L.; and Olesen, K. G. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly* 4:269–282.

[Kalagnanam and Henrion, 1990] Kalagnanam, J. and Henrion, M. 1990. A comparison of decision analysis and expert rules for sequential diagnosis. In Shachter, R. D.; Levitt, T.; Kanal, L.; and Lemmer, J., editors 1990, *Uncertainty in Artificial Intelligence*, volume 4. North-Holland, 1990. 271–281.

[Lauritzen and Spiegelhalter, 1988] Lauritzen, S. L. and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their applications to expert systems. *J. R. Statist. Soc. B* 50(2):157–224.

[McAllester, 1980] McAllester, D. 1980. An outlook on truth maintenance. Technical Report AIM-551, Artificial Intelligence Laboratory, M. I. T., Cambridge, Mass.

[Mozetič, 1991] Mozetič, I. 1991. Hierarchical model-based diagnosis. *Int. J. of Man-Machine studies* 35(3):329–362. Also in [Hamscher *et al.*, 1992]. 354–372.

[Pearl, 1987] Pearl, J. 1987. Distributed revision of composite beliefs. *Artificial Intelligence* 33(2):173–215.

[Pearl, 1988] Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, Calif.

[Pearl, 1994] Pearl, J. 1994. A probabilistic calculus of actions. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*. Morgan Kaufmann, San Mateo, Calif., 1994. 454–462.

[Poole and Provan, 1991] Poole, D. and Provan, G. 1991. Use and granularity in consistency-based diagnosis. In *The Second International Workshop on Principles of Diagnosis*. September 1991.

[Poole, 1993] Poole, D. 1993. The use of conflicts in searching Bayesian networks. In *Proc. Ninth Conference on Uncertainty in Artificial Intelligence (UAI '93)*. Morgan Kaufmann, San Mateo, Calif., 1993. 359–367.

[Portinale, 1992] Portinale, L. 1992. Modeling uncertain temporal evolutions in model-based diagnosis. In *Proc. Eighth Annual Conference on Uncertainty Artificial Intelligence (UAI '92)*. Morgan Kaufmann, San Mateo, Calif., 1992. 244–251.

[Raiman *et al.*, 1991] Raiman, O.; de Kleer, J.; Saraswat, V.; and Shirley, M. H. 1991. Characterizing non-intermittent faults. In *Proc. National Conference on Artificial Intelligence (AAAI '91)*. AAAI Press, Menlo Park, Calif., 1991. 849–854. Also in [Hamscher *et al.*, 1992]. 170–175.

[Shachter and Kenley, 1989] Shachter, R. S. and Kenley, C. R. 1989. Gaussian Influence Diagrams. *Management Science* 35:527–550.

[Shachter, 1986] Shachter, R. D. 1986. Evaluating Influence Diagrams. *Operations Research* 34(6):871–882.

[Srinivas and Breese, 1990] Srinivas, S. and Breese, J. 1990. IDEAL: A software package for analysis of influence diagrams. In *Proc. Sixth Annual Conference on Uncertainty Artificial Intelligence (UAI '90)*. 1990. 212–219.

[Srinivas and Horvitz, 1995] Srinivas, S. and Horvitz, E. 1995. Exploiting system hierarchy to compute repair plans in probabilistic model-based diagnosis. In *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95)*. Morgan Kaufmann, San Mateo, Calif., 1995.

[Srinivas, 1994] Srinivas, S. 1994. A probabilistic approach to hierarchical model-based diagnosis. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*. Morgan Kaufmann, San Mateo, Calif., 1994. 538–545.

[Srinivas, 1995a] Srinivas, S. 1995a. Modeling failure priors and persistence in model-based diagnosis. In *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95)*. Morgan Kaufmann, San Mateo, Calif., 1995.

[Srinivas, 1995b] Srinivas, S. 1995b. A polynomial algorithm for computing the optimal repair strategy in a system with independent component failures. In *Proc. Eleventh Conference on Uncertainty in Artificial Intelligence (UAI '95)*. Morgan Kaufmann, San Mateo, Calif., 1995.

[Struss and Dressler, 1989] Struss, P. and Dressler, O. 1989. Physical negation: Integrating fault models into the General Diagnostic Engine. In *Proc. Eleventh International Joint Conference on Artificial Intelligence (IJCAI '89)*. 1989. 1318–1323.

[Sun and Weld, 1993] Sun, Y. and Weld, D. S. 1993. A Framework for Model-Based Repair. In *Proc. National Conference on Artificial Intelligence (AAAI '93)*. AAAI Press, Menlo Park, Calif., 1993. 182–187.

[Tarjan and Yannakakis, 1984] Tarjan, R. E. and Yannakakis, M. 1984. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce hypergraphs. *SIAM J. Computing* 13:566–579.

[Tsokos and Shimi, 1977] Tsokos, C. P. and Shimi, I. N., editors 1977. *The theory and applications of Reliability, with emphasis on Bayesian and Nonparametric methods*, volume 1. Academic Press.

[Warner *et al.*, 1961] Warner, H. R.; Toronto, A. F.; Veasy, L. G.; and Stephenson, R. 1961. A mathematical approach to medical diagnosis: application to congenital heart disease. *Journal of the American Medical Association* 177:177–183.

[Yuan, 1993] Yuan, S. 1993. Knowledge-based decision model construction for hierarchical diagnosis: A preliminary report. In *Proc. Ninth Conference on Uncertainty in Artificial Intelligence (UAI '93)*. Morgan Kaufmann, San Mateo, Calif., 1993. 274–281.