PH.D. THESIS

Intelligent Distributed Fault and Performance Management for Communication Networks

by Hongjun Li
Advisor: John S. Baras

CSHCN PhD 2002-2
(ISR PhD 2002-2)

INTELLIGENT DISTRIBUTED FAULT AND PERFORMANCE MANAGEMENT FOR COMMUNICATION NETWORKS

by

Hongjun Li

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2002

ABSTRACT

Title of Dissertation: INTELLIGENT DISTRIBUTED FAULT AND PERFORMANCE MANAGEMENT FOR COMMUNICATION NETWORKS

Hongjun Li, Doctor of Philosophy, 2002

Dissertation directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

This dissertation is devoted to the design of an intelligent, distributed fault and performance management system for communication networks. The architecture is based on a distributed agent paradigm, with belief networks as the framework for knowledge representation and evidence propagation.

The dissertation consists of four major parts. First, we choose the mobile code technology to help implement a distributed, extensible framework for supporting adaptive, dynamic network monitoring and control. The focus of our work is on three aspects. First, the design of the standard infrastructure, or Virtual Machine, based on which agents could be created, deployed, managed and initiated to run. Second, the collection API for our delegated agents to collect data from network elements. Third, the callback mechanism through which the functionality of the delegated agents or even the native software could be extended. We propose three system designs based on such ideas.

Second, we propose a distributed framework for intelligent fault management purpose. The managed network is divided into several domains and for each domain, there is an intelligent agent attached to it, which is responsible for this domain's fault management tasks. Belief network are embedded in such an agent as the probabilistic fault models, based on which evidence propagation and decision making processes are carried out.

Third, we address the problem of parameter learning for belief networks with fixed structure. Based on the idea of Expectation-Maximization (EM), we derive a uniform learning algorithm under incomplete observations. Further, we study the rate of convergence via the derivation of Jacobian matrices of our algorithm and provide a guideline for choosing step size. Our simulation results show that the learned values are relatively close to the true values. This algorithm is suitable for both batch and on-line mode.

Finally, when using belief networks as the fault models, we identify two fundamental questions: When can I say that I get the right diagnosis and stop? If right diagnosis has not been obtained yet, which test should I choose next? The first question is tackled by the notion of right diagnosis via intervention, and we solve the second problem based on a dynamic decision theoretic strategy. Simulation shows that our strategy works well for the diagnosis purpose. This framework is general, scalable, flexible and robust.

DEDICATION

To Miao and Flora

To my parents

ACKNOWLEDGEMENTS

I am deeply grateful to my advisor Professor John S. Baras for his invaluable encouragement, insightful guidance and enthusiastic support during the course of my Ph.D. study. Without his inspiration and support, the accomplishment of this dissertation would have been impossible.

I am also very grateful to Professor Carlos A. Berenstein, Professor Jerome A. Gansman, Professor Prakash Narayan, and Professor Nicholas Roussopoulos, for their kindly consenting to join the advisory committee and review this thesis.

I would also like to thank many of my friends who support me during my study at University of Maryland. They make my student life at Maryland warm, colorful and rewarding.

Special thanks to my colleague and friend Shah-An Yang, for our numerous talks and discussions. I am also thankful to Tina M. Vigil, Diane E. Hicks, and Althia Y. Kirlew for their technical help.

I am also grateful to the Center for Satellite and hybrid Communication Networks for the support of my research. This work was supported by NASA Glen NCC3528, ATIRP QK9931, and Hughes Network Systems MIPS 233113. I would also like to thank the Internet department at MCI/WORLDCOM for providing me the summer intern opportunity in 1998.

Special thanks to Mike Skeith and David Whitefield from Hughes Network Systems for their insightful suggestions to this work.

Last, but certainly not the least, I would like to acknowledge the immeasurable support from my family, my wife Miao, daughter Flora and my parents, for their understanding, sacrifices, and unconditional love. To them, with the deepest love, I would like to dedicate this dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTELLIGENT DISTRIBUTED FAULT
AND PERFORMANCE MANAGEMENT
FOR COMMUNICATION NETWORKS

Hongjun Li

February 15, 2002

Chapter 1

Introduction

Communication networks have become indispensable today and this trend will continue as more and more new technologies emerge. These will provide both opportunity and challenge. A network can be configured to use the latest technologies and be customized to the user's needs. At the same time, the risk or faults in such a heterogeneous system will increase [1]. To meet the needs of current and future communication environments, it is the responsibility of network management to maintain the network operation and service.

Conceptually, any system that is able to monitor and control the state of another centralized or distributed system at medium to long term time scales can be called a network management system, and it consists typically of an operator interface with a powerful but user-friendly set of commands for performing monitoring and control tasks (manager); and incremental hardware and software embedded in network elements, computers and managed resources in general that allow the manager to monitor and control the state of the equipment. Typically, a Network Management System (NMS) consists of the following five functional areas: **F**ault Management, **C**onfiguration Management, **A**ccounting Management, **P**erformance Management and **S**ecurity Management (FCAPS).

The role of fault management is to detect, isolate, diagnose and correct the possible faults during network operations. Therefore it is primarily fault management that helps to keep the normal operations and ensure the networks reliability and availability. Due to the growing number of networks that have served as the critical components in the infrastructure of many organizations, interest in fault management has increased during the past decade, both in academia and in industry [2, 3, 4, 5, 6, 7, 8].

In general, any fault diagnosis procedure can be interpreted in terms of search spaces and corresponding operations [9]. The search spaces are *data space, hypothesis space* and *repair space.* In data space, measured data, together with alarms and users reports, are mapped into some fault hypotheses. It may include operations like data gathering, data analysis (such as trend analysis and feature extraction) and hypothesis testing. In hypothesis space, the hypotheses generated in data space are mapped into some possible causes. Usually, there is a fault model in this space based on which the reasoning can be executed. In repair space, such causes are mapped into a set of possible actions to treat or repair the faulty components in some efficient way. Such a space-operation paradigm has been successfully adopted in many fault diagnosis applications in various areas like electric circuits and chemical industry. In communication networks fault management, we can also take this paradigm, and we base our work on three important assumptions [10]:

- The objective is to deal with malfunctions, not the design faults, of the system. So it is basically a **fault diagnosis** problem, rather than fault tolerant system design.

- Tests are more expensive than computations so it is more favorable to compute and infer the faults and their causes rather than brute-force tests.

- Mis-diagnosis is more expensive than tests. Thus, it is desirable that a fault management system can cover and diagnose as many fault scenarios as possible in a cost efficient manner.

The task of fault management is to detect, diagnose and correct the possible faults during network operations. Fault detection refers to the recognition of the symptoms of a fault, which include trouble reports issued from end users or personnel, and alarms issued from the monitoring processes. Fault detection can be thought of as an online process that gives indication of malfunctioning. To declare the existence of such malfunctioning, we need a model of "normal" behavior against which comparisons can be made. Such normal behavior could be specified as a finite state machine, as in the case of protocol or software testing [11, 12, 13]; or it could be a derived model according to operation status and/or statistical analysis, for example Auto Regressive (AR) model [14] and Generalized Likelihood Ratio (GLR) model [15]. A normal behavior could also be a service agreement between an end user and a service provider. So if the user senses service degradation, he/she may then file a trouble report. Typically, such indications of malfunctioning are manifested in the form of events, which should be correlated to facilitate fault diagnosis [16, 17, 18]. Fault diagnosis is the most difficult task of fault management, and refers to the process of identifying the most likely reason(s) for the symptoms based on some modeling of cause and effect relationships among the propositions of interest in the current problem domain. Inputs to the fault diagnosis task are the detected symptoms in terms of system reported alarms or user and personnel reported trouble reports. Finally, corrective actions are taken to restore the normal operations. In addition to the mentioned tasks, fault documentation is also important due to being fundamental for the enhanced functionalities. In [2], Dreo presented an overview of trouble ticket systems for such documentation tasks. In our work, we focus on fault diagnosis issues.

1.1 Motivations

In this section, we describe the motivations for our integrated, intelligent fault management system based on a critical evaluation of current research results and approaches.

Distributed Management Architecture

A conventional network management system consists of two classes of components: managers and agents. Applications in the management station assume the manager role; Agents are server processes running in each involved manageable network entity. These agents collect network device data, stores them in some format, and support a management protocol, e.g., Simple Network Management Protocol (SNMP) [19, 20]. Manager applications retrieve data from element agents by sending corresponding requests over the management protocol. There is no intelligence embedded near the network elements. What the agent does is to provide the manager with the desirable data only. It is the manager that performs all the statistics computation, threshold checking and other applications, e.g., the fault diagnosis steps.

Such a system favors a centralized framework and works well for small networks. But as the networks become larger, more complex, and heterogeneous (e.g. multimedia networks), the centralized paradigm will incur vast amounts of communication between manager and agent and thus occupy too much bandwidth inefficiently. Since not all the data are relevant and necessary for the manager to process, and there are many cases where the processing can be done on the spot, there is no need to centralize all the processing at the manager site.

In this regard, we borrow the idea of Management by Delegation (MbD) [21] and distribute some of the processing logic and responsibilities by embedding code within the network elements. This embedded code within the network element is called a *delegated agent*. In our work, such delegated agents are deployed to the network elements for the monitoring and control purpose.

In terms of fault management, we also propose that the faults should be dealt with locally if they are local. Only those that cannot be handled locally should draw global attention. To do this, we propose an intelligent, distributed fault management system where the managed network is divided into separate domains. It is the author's belief that it will be more efficient, both in time and bandwidth utilization, if faults were handled in this way.

Flexible Manager-Agent Interface

In conventional network monitoring systems, the set of services offered by the element agents is fixed and is accessible through interfaces that are statically defined and implemented, for example Remote Monitoring (RMON) [22]. Statically pre-assigning functionality implies that the decision of what functionality to delegate needs to be taken at the agents' design phase. But, not all possible management tasks can be predefined this way. Further, requirements in the dynamic network environment change very often, which means that new type of functionality might be required now and then. To this end, not only do we need to distribute intelligence, we also need to provide a dynamically extensible interface between such agents and the manager, such that the manager could change the parameter values, and extend the processing logic of the delegated agents dynamically.

Further, the functionality of the underlying native processes could also be dynamically extended via our callback mechanisms. The native processes, written in C/C++ for many cases, embody the processing logic viewed as necessary at the time of native software design and implementation. They may, however, lack consideration of some unanticipated cases. Such unanticipated cases, if they do occur, might lead to inconsistency in the processing followed. Thus we need to modify or extend the native software processing logic somehow to accommodate those unanticipated cases. Based on the observation that we would not like to re-code the C/C++ programs and recompile, reinstall, and reinstantiate the server processes, which usually incurs system down time, we need a flexible way such that the processing logic could be extended dynamically. Here, we respect the current processing logic and put on more processing capabilities to handle the unexpected cases. This is more like putting a "booster" rather than replacing the original logic.

Note that the awareness of such undesirability and thus the need for change of logic is from human, not from the native processes themselves. It is the human manager again, that determines the functionality of the added codes. Such added codes would work with the original codes to get the expected results for both regular and the unanticipated situations.

Automated System

In legacy communication networks, fault diagnosis is often not too difficult since the knowledge of the network manager combined with the alarms reported is usually enough to rapidly locate most failures. But in future communication networks, which are expected to be broadband, giant, heterogeneous and complex, things will not be that easy. As the size and speed of the networks grow, their dynamics become increasingly difficult to understand and control. For example, a single fault can generate a lot of alarms in a variety of domains, with many of them not helpful. Multiple faults will make things even worse. In such cases, it is almost impossible for the network manager, inundated in the ocean of alarms, to correlate the alarms and localize the faults rapidly and correctly just by his experience. On the other hand, more and more users, possibly with different or even competing requirements of quality of service (QoS), wish to benefit from the networks. These will pose significant problems on fault management and thus more advanced techniques are needed. Therefore, efficient fault management requires an appropriate level of *automation*.

Probabilistic Fault Models

Knowledge-based expert systems, as examples of automated systems, have been very appealing for complex system fault diagnosis [23]. Nevertheless, most of the developed expert systems were built in an *ad-hoc* and unstructured manner by simply transferring the human expert knowledge to an automated system. Usually, such systems are based on deterministic network models. A serious problem of using deterministic models is their inability to isolate primary sources of failures from uncoordinated network alarms, which makes automated fault identification a difficult task. Observing that the cause-and-effect relationship between symptoms and possible causes is inherently nondeterministic, *probabilistic* models can be considered to gain a more accurate representation for the networks. As a natural and efficient model for human inferential reasoning, belief networks have emerged as the general knowledge representation scheme under uncertainty and key technology for diagnosis [24, 25, 26, 27]. In our work, we use belief networks as the probabilistic fault models.

Integrated Fault Management

In previous research on fault management, the term "fault" was usually taken the same as "failure", which means component (hardware or software) malfunctions, e.g. sensor failures, broken links or software malfunctions [8]. Hardware faults are usually due to incorrect or incomplete logic design, damage, wear or expiry, etc. Software faults usually come from incorrect or incomplete design and implementation. We call such faults *hard* faults. In communication networks, however, there are still some other important kinds of faults that need to be considered. For example, the performance of a switch is degrading or there exists congestion on one of the links. Another example is to model faults as deviations from normal behavior [28]. Since there might not be a failure in any of the components, we call such faults *soft* faults. Hard faults can be solved by replacing hardware elements or software debugging. Such diagnosis is called re-active diagnosis in the sense that it consists of basically the reactions to the actual failures. Soft faults are in many cases indications of some serious problems and for this reason, the diagnosis of such faults is called *pro-active* diagnosis. By early attention and diagnosis, such pro-active management will sense and prevent disastrous failures and thus can increase the survivability and efficiency of the networks. Handling soft faults is typically part of the functionality of performance management [29, 30] and in the sequel, we use the term ”fault” to represent both hard and soft faults for convenience.

Decision-Theoretic Diagnosis Strategy

In communication networks fault management field, Hood and Ji [14] proposed a pro-active network fault detection scheme based on Auto Regressive (AR) models and belief networks. Selected Management Information Base (MIB) variables were monitored and their normal behaviors are learned via AR modeling. Belief networks were used to compute certain posterior probabilities, given some deviations from the normal behavior. However, their belief network model is over simplistic in that there is only one root node, which will explain whatever anomalies as detected by the AR modeling.

Obviously, a more general belief network model is needed if we should diagnose the symptoms and give out explanations (rather than detection) of the current deviation from the normal behavior. After observing symptoms, such initial evidence is propagated and the posterior probability of any possible candidates being faulty can be calculated. It would be ideal if we can locate the fault with efforts up to this. But most of the time, similar to what happens in medical diagnosis, we need more information to help pinpoint the fault. So naturally, we need to know what to do next and when to stop.

In [31], Huard and Lazar used a more general belief network model with multiple root nodes as the candidate faults. They also presented a dynamic programming (DP) formulation for the network troubleshooting problem. However, single fault assumption was made, which limits the applicability.

In our work, we develop a framework that supports fault diagnosis for communication networks. General belief network models with multiple root nodes are chosen as the knowledge representation scheme. We handle multiple faults and formulate the fault diagnosis procedure as a Partially Observable Markov Decision Processes (POMDP) problem with optimal stopping. To help solve the problem, we introduce the notion of right diagnosis for optimal stopping and provide a dynamic, heuristic strategy for test sequence generation.

1.2 Contributions

Based on the above motivations, this dissertation is devoted to the design of an automated, intelligent and distributed fault and performance management system for communication networks. The system architecture is based on a distributed and flexible agent paradigm, and we use Bayesian belief networks as the framework for knowledge representation and evidence propagation. Both hard and soft faults are integrated and we propose a dynamic heuristic strategy for test sequence generation.

Adaptive, Distributed Network Monitoring and Control

We choose the mobile code technology, in particular remote evaluation and code on demand paradigms, to help implement a distributed, extensible framework for supporting adaptive, dynamic network monitoring and control. To make it possible for agents to exist in heterogeneous environments, there needs to be a standard infrastructure on each system where they need to be hosted. Then agents may be developed as if they will be always on the same machine–the Virtual Machine, which could be but not limited to Java Virtual Machine (JVM). In our work, we use either JVM or C/C++ dynamic linkage technology to serve as the Virtual Machine under different situations, see our papers [32, 33, 34].

The focus of our work is on three aspects. First, the design of the standard infrastructure, or Virtual Machine, based on which agents could be created, deployed, managed and initiated to run. Second, the collection API for our delegated agents to collect data from network elements. Third, the callback mechanism through which the functionality of the delegated agents or even the native software could be extended. We propose three system designs based on such ideas.

Our first design uses full-blown JVM in both manager and network element site and assumes the presence of MIBs. It is a proof-of-concept design and is suitable for network elements equipped with powerful computing and memory capabilities, i.e. routers and ATM switches. Here we use the off-the-shelf JVM and we do not need to access the network element native software directly; instead, we need only to access the MIBs that store the raw monitoring data. Our prototype system works well, which encouraged us to research further into the Virtual Machines and collection API issues.

In our second design we consider the situations where there is no MIB embedded with network elements. We still use JVM but here our focus is on the network elements equipped with limited computing and memory capabilities. Specific versions of JVM are considered. For the delegated Java agents to access the native software, Java Native Interface (JNI) is exploited and a directory containing addresses of the native global variables and function pointers is set up. The processing logic of the delegated agents could be extended by creating new agents with the desirable functionality, followed by deploying them to the network elements to replace the old agents. To extend the native software functionality, we carry out function replacement by swapping the function pointers of the Java agents and the corresponding native code functions.

Further, in our third design we remove the convenient JVM for those network elements equipped with multiple processors and address spaces. The focus here is to use dynamic linkage technology to emulate the Virtual Machine concept. The delegated agents are dynamically linked to the native code by the C/C++ run-time environment. The collection API in this case is very thin since all that is needed is to access the native code directly. The extension of functionality is similar as the second design, with the difference that we do not need JNI in this case. This design is suitable for those resource limited network elements that run over a real-time operating system and will not use Java as the native code development.

A Framework for Fault and Performance Management

The managed network is divided into several domains [35] and for each domain, there is an intelligent agent attached to it, which is responsible for this domain's fault management. Each agent is called a "Domain Diagnostic Agent (DDA)" with the goals of monitoring the health of the domain and diagnosing the faults in a cost-efficient manner. Belief network models are embedded in such a DDA as the probabilistic fault models. Both hard and soft faults can be naturally incorporated in a belief network model. A domain is an abstract notion, for example it might be a subnet, a cluster, a host or a member of a functional partition. For those problems that none of the individual agent can solve, there is a mechanism by which the agents can report to the coordinator and share the information in order to get a global view and solve it cooperatively. So the whole system is, from the agent point of view, a distributed, cooperative multi-agent system. This framework is quite general and can be incorporated into many network management paradigms, like the traditional client-server (CS) based architecture, and the more recent mobile-code based framework as discussed above. Our previous publications on this work can be found in [36, 37].

Statistical Parameter Learning for Belief Networks

When building a belief network model, initially both the network structure and the associated CPTs can be provided by human experts as the prior information. In many applications, however, such information is not available. In addition, different experts may treat the systems in various ways and thus give different and sometimes conflicting assessments. In such cases, the network structure and corresponding CPTs can be estimated using empirical data and we refer to this process as learning. Even if such prior information does exist, it is still desirable to validate and improve the model using data.

In our work [38], we address the problem of parameter learning for belief networks with fixed structure. Both complete and incomplete (data) observations are included. Given complete data, we describe the simple problem of single parameter learning for intuition and then expand to belief networks under appropriate system decomposition. If the observations are incomplete, we adopt the idea of Expectation-Maximization (EM) and derive a uniform learning algorithm. Further, we study the rate of convergence via the derivation of

Jacobian matrices of our algorithm and provide a guideline for choosing step size. Our simulation results show that the learned values are relatively close to the true values. This algorithm is suitable for both batch and on-line mode for real applications.

Dynamic Test Generation with Optimal Stopping

When using belief networks as the knowledge representation scheme and inference engine for the problem domain, we identify in our recent paper [39] two fundamental questions: When can I say that I get the right diagnosis and stop? If right diagnosis has not been obtained yet, which test should I choose next? The first question is tackled by the notion of right diagnosis via intervention, and we solve the second problem based on a dynamic decision theoretic strategy. Simulation shows that our strategy works well for the diagnosis purpose.

This framework is quite **general**. Belief network models have very rich expressive capability and further, the belief network model and the associated decision making algorithm could exist at any management station in a network management system.

Due to the event correlation procedure prior to the diagnosis process, only a small fraction of the so many alarms generated in a big problem domain is chosen as input to a belief network model. Thus, the diagnosis based on such condensed events tackles much less symptoms, which makes our framework and algorithm **scalable** and run fast.

Moreover, our framework is **robust** to noise and incomplete data. By nature, belief network models handle the problem of uncertainty in the cause and effect relationship among propositions. In terms of observation noise, spurious alarms can be easily tackled in the event correlation phase; if the input events are not complete, i.e., one or more of the condensed events from the event correlation process is lost, the lost events can be easily demanded by our dynamic troubleshooting strategy if such lost events are calculated as relevant for further diagnosis. Further, we observe from our experience of statistical parameter learning that, in terms of fault diagnosis, the *true* and *learned* belief networks would give the *same* test sequences and average cost under most of the symptom patterns, and we conclude that such diagnostic belief network models are not so sensitive to the parameters.

After a test node is chosen, the observation for this test may take advantage of the traditional SNMP paradigm by polling appropriate MIB variables; or in our case, delegated (mobile) agents could be sent to the network elements to collect the data, as discussed above.

Note that as evidence accumulates, we may input them one by one followed by a propagation right after each evidence-input, or we may input them once altogether and do only one propagation. This provides us the flexibility for either on-line diagnosis or off-line diagnosis/analysis.

1.3 Organization

The rest of the dissertation is organized as follows. In chapter 2, we describe the distributed, extensible framework for dynamic network monitoring and control. In chapter 3, we give a brief introduction of what belief networks are as the background knowledge and discuss why we choose belief networks as the probabilistic fault model for our purpose. In chapter 4, we describe the system architecture and function definitions of our framework. We also include the discussion of event correlation as the preprocessing procedure before using belief networks, and outline the steps to construct such belief network models. We present our work of statistical learning for belief networks in chapter 5, and then in chapter 6 we discuss the fault diagnosis problems and solutions using belief networks. Finally, chapter 7 summarizes this dissertation and suggests future research.

Chapter 2

On System Designs for Adaptive, Distributed Network Monitoring and Control

The increasing complexity and importance of communication networks have given rise to a steadily high demand for advanced network management. Network management system handles problems related to the configurability, reliability, efficiency, security and accountability of the managed distributed computing environments. Accurate and effective monitoring and control is fundamental and critical for all network management functional areas. In this chapter, we present our distributed, extensible framework for supporting adaptive, dynamic network monitoring and control. We choose the mobile code technology, in particular remote evaluation and code on demand paradigms, to help implement such a framework.

2.1 Mobile Code Design Paradigms

To understand mobile code technology, we first present some basic concepts that are an abstraction of the entities that constitute a software system. In particular, we introduce three architectural concepts: *components, interactions,* and *sites* [40].

Components are the constituents of a software architecture. They can be further divided into *code components,* that encapsulate the know-how to perform a particular computation, *resource component,* that represent data or devices used during the computation, and *computational components,* that are active executors capable to carry out a computation, as specified by a corresponding know-how. *Interactions* are events that involve two or more components, e.g., a message exchanged among two computational components. *Sites* host components and support the execution of computational components. A site represents the intuitive notion of location. Interactions among components residing at the same site are considered less expensive than interactions taking place among components located in different sites. In addition, a computation can be actually carried out only when the know-how describing the computation, the resources used during the computation, and the computational components responsible for execution are located at the same site.

Design paradigms are described in terms of interaction patterns that define the relocation of and coordination among the components needed to perform a service. We will consider a scenario where a computational component A, located at site S_A needs the results of a service. We assume the existence of another site S_B, which will be involved in the accomplishment of the service. We identify three main design paradigms exploiting code mobility: *remote eval-*

Table 2.1: Mobile Code Design Paradigms

Paradigm	Before		After	
	S_A	S_B	S_A	S_B
Client_Server	*A*	K_h, Resource, *B*	*A*	K_h, Resource, **B**
Remote Evaluation	K_h, *A*	Resource, *B*	*A*	*K_h,* Resource, **B**
Code on Demand	Resource, *A*	K_h, *B*	Resource, *K_h,* **A**	*B*
Mobile Agent	K_h, *A*	R	—	*K_h,* Resource, **A**

uation, code on demand, and *mobile agent.* These paradigms are characterized by the location of components before and after the execution of the service, by the computational component which is responsible for execution of code, and by the location where the computation of the service actually take place. These paradigms are compared with the traditional client-server paradigm. Table 2.1 shows the location of the components before and after the service execution [40], where K_h stands for Know_how. For each paradigm, the computational component in bold face is the one that executes the code. Components in italics are those that have been moved. Also see figure 2.1, where NMS stands for network management station, Req stands for request, and Rep stands for Reply.

The client-server (CS) paradigm is well known and widely used. In this paradigm, a computational component B (the server) offering a set of services is placed at site S_B. Resources and know-how needed for service execution are hosted by site S_B as well. The client component A, located at site S_A, requests the execution of a service with an interaction with the server component B. As a response, B performs the requested service by executing the corresponding know-how and accessing the involved resources co-located with B. In general,

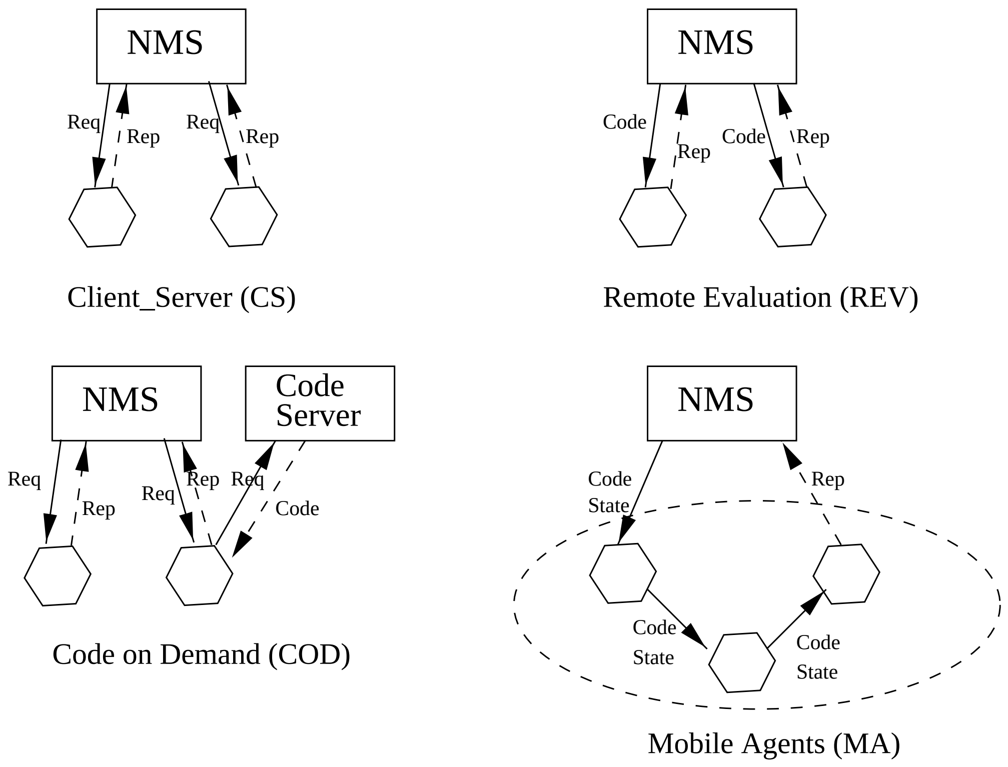

Figure 2.1: Design Paradigms

the service produces some sort of results that will be delivered back to the client with an additional interaction.

In the remote evaluation (REV) paradigm, a component A has the know-how necessary to perform the service but it lacks the resources required, which happen to be located at a remote site S_B. Consequently, A sends the service know-how to a computational component B located at the remote site. B, in turn, executes the code using the resources available there. An additional interaction delivers the results back to component A.

In the code on demand (COD) paradigm, component A is already able to access the resources it needs, which co-located with it at S_A. However, no information about how to manipulate such resources is available at site S_A.

Thus, A interacts with a component B at S_B by requesting the service know-how, which is located at S_B as well. A second interaction takes place when B delivers the know-how to A, that can subsequently execute it.

In the mobile agent (MA) paradigm, the service know-how is owned by A, which is initially hosted by S_A, but some of the required resources are located on S_B. Hence, A *migrates* to S_B carrying the know-how and possibly some intermediate results. After it has moved to S_B, A completes the service using the resources available there. The mobile agent paradigm is different from other mobile code paradigms since the associated interactions involve the mobility of an *existing* computational component. In other words, while in REV and COD the focus is on the transfer of code between components, in the MA paradigm a whole computational component is moved to a remote site, along with its state, the code it needs, and some resources required to perform the task.

The client-server paradigm is static with respect to code and location. Once created, components cannot change either their location or their code during their lifetime. Therefore, the type of interaction and its quality (local or remote) cannot change. Mobile code paradigms overcome these limits by providing component mobility. By changing their location, components may change dynamically the quality of interaction, reducing interaction costs. To this end, the REV and MA paradigms allow the execution of code on a remote site, encompassing local interactions with components located there. In addition, the COD paradigm enables computational components to retrieve code from other remote components, providing a flexible way to extend dynamically their behavior and the types of interactions they support.

2.2 Mobile Code for Network Management

The world of network management research can be split roughly in two worlds: management of IP networks, where the Simple Network Management Protocol (SNMP) [19, 20] proposed by IETF is the dominant protocol, and management of ISO networks, based on the Common Management Information Protocol (CMIP) [19]. Both protocols are based on a CS paradigm where a network management station (the client) polls information from agents (the servers) residing on the network devices. Each agent[1] is in charge of managing a management information base (MIB)[2], a hierarchical information base that stores the relevant parameters for the corresponding device. In this setting, all the computation related to management, e.g., statistics, is demanded to the management station. Polling is performed using very low level primitives—basically **get** and **set** of atomic values in the MIB. This fine-grained CS interaction leads to the generation of intense traffic and computational overload on the management station. This centralized architecture is particularly inefficient during periods of heavy congestion, when management becoming important. In fact, during these periods the management station increases its interactions with the devices and possibly uploads configuration changes, thus increasing congestion. In turn, congestion, as an abnormal status, is likely to trigger notifications to the management station, which worsen network overload. Due to this situation, access

[1]Despite the name, management agents are conventional programs that cannot move and in general do not exhibit a great deal of ingelligence

[2]MIB is actually the term used for information base in SNMP only. CMIP uses the term management information tree (MIT) database instead. Hereafter, we will ignore the difference for the sake of simplicity.

to devices in the congested area becomes difficult and slow.

These problems have been addressed by IETF and ISO with modifications of their management architecture. For instance, SNMPv2 introduced hierarchical decentralization through the concept of *proxy* agents. A proxy agent is responsible for the management of a pool of devices (towards which it acts as a client) on behalf of the network management station (towards which it acts as a server). Another protocol derived from SNMP, called Remote MONitoring (RMON) [22], assumes the existence of stand-alone dedicated devices called *probes*. Each probe hosts an agent able to monitor "global" information flowing through links rather than information "local" to a device. Although these decentralization features improve the situation, experimentation showed that they do not provide the desired level of decentralization needed to cope with large networks. In the sequel, we analyze if and how mobile code design paradigms can provide a suitable alternative to the CS paradigm fostered by SNMP. Let us begin with the CS paradigm.

The rationale for the management architecture proposed in SNMP and CMIP, which provides very low-granularity primitives, is to keep the agents on the device small and easily implementable, keeping all the complexity on the management station. Nevertheless, this is going to dramatically increase congestion and decrease performance. For instance, tables are often used to store information into devices. To search a value in a table using a CS approach, either the table has to be transferred to the management station and searched there for the desired value, or the agent has to be modified to provide a new search service. Neither solution is desirable. The former leads to bandwidth waste for large tables. The second increases the size of the agent as a large number of routines are implemented, maybe without a substantial payoff if the routines are used only now and then.

The REV paradigm could be used to pack together the set of primitive operations describing the search and send them on the device holding the table for local interaction[3]. After execution, only the target value should be sent back—thus performing semantic compression of data. This solution is likely to save bandwidth at least for big tables and small routines. As an aside, this solution provides a desirable side effect: it raises the level of abstraction of the operations available to the network manager. One could envision a scenario where the manager builds her own management procedure upon lower level primitives, stores them on the management station, and invokes their remote evaluation on the appropriate device whenever needed.

On the other hand, the capability to retain the state across several hops implicit in an MA design adds a new dimension to the benefits achievable through an REV design: autonomy. In the REV design, each remote evaluation on a device must be initiated explicitly by the management station. In the MA paradigm, the management station can exploit the capability of a mobile component to retain its state and demand to it the retrieval of information from a specified pool of devices. Thus it can delegate to it the decision about when and where to migrate, according to its current state. Whether this is actually improving traffic load is still unclear at this point, because the state of the mobile component is likely to grow from hop to hop.

The COD paradigm provides the capability to extend dynamically the set of

[3]We assume the presence of a run-time support for mobile code on network devices. This run-time support is just a kind of Virtual Machine.

services offered by a device. This is convenient if many identical queries have to be performed on a device: once the code to perform the primitive queries locally is installed, it can be remotely invoked by the management station. On the other hand, if few different queries have to be performed, COD does not help that much: either a REV or MA paradigm needs to be exploited. In our work, we choose REV and COD paradigms.

In the next section, we discuss some related work in terms of decentralized network management. Some of them fall into the category of mobile code, while some of them do not. But they are still discussed here to help us obtain a better picture of the relative positioning of different technologies.

2.3 Related Work in Network Management

Management by Delegation

Management by Delegation (MbD) [21] is one of the earliest efforts towards decentralization and increased flexibility of management functionality, and it greatly influenced later research and exploration along this direction [41, 42]. MbD identifies an architecture for the dynamic uploading of management scripts on network devices using a combination of REV and CS paradigms. The main advantage of this approach is that it is language independent. However, the proof-of-concept MbD system was implemented with a proprietary server environment and we hardly see any working systems that are built upon this proprietary environment. Also, the MbD server environment is so comprehensive and complicated that it can turn out to be an "overkill" in most real-world applications. Still, we must give credit to MbD because it can be considered a precursor of the ideas discussed in this chapter. The major difference is that we have adopted the standard Java or C/C++ platform and, from the very beginning, aimed to build a portable, simple, yet powerful framework that can be easily understood, implemented and enhanced.

Flexible Agents and AgentX

In [43], Mountzia discussed temporal aspects of the delegation process and analyzed many issues concerning the application of the delegation concept in integrated network management. This framework is close to our system designs and it provided some helpful tips for our work. However, we also need to tackle the problem of extending the functionality of the native processes, which incurs many other issues, such as native collection API, callback mechanisms, etc.. In Internet community, RFC 2741 [44] defines a standardized framework for extensible SNMP agents. It defines processing entities called master agents and subagents, a protocol (AgentX) for the communication between them, and the elements of procedure by which the extensible agent processes SNMP protocol messages. RFC 2742 [45] defines the associated Management Information Base (MIB) that uses the AgentX protocol. In our work, however, we need to face such situations that there is no MIB embedded in network elements, i.e. small satellite terminals, and again, we tackle the problem of native process extension.

Mobile Agents

Another approach that enables dynamic downloading of functionality is provided by mobile agents [46, 47, 48], as discussed above. Languages that are used to develop mobile agents include Java [49], Tcl/Tk [50], and Telescript [51], among others, and using mobile agents in decentralized and intelligent network management is a great leap from client-server based management pattern. Our system exploits the idea of REV and COD rather than mobile agents (in the sense of existence of an itinerary), and assumes a management server in each device concerned. Comparing with their mobile agent counterparts, the behaviors of our agents are much easier to understand and anticipate. Since our agents could also be implemented via native code, they are less straightforward, but more powerful.

Web-based Network Management

We are by no means the first people thinking of using Java technology in network management [52, 53]. Web-based Network Management is a well-justified idea that attempts to provide uniform management services through such common client-side interface as Web browsers. Java Management API Architecture (JMAPI) provides a single device-specific, platform independent management applet, written in Java, that allows a network administrator to manage a network device with Remote Method Invocation (RMI). In this architecture, vendors save the cost of supporting many add-ons on multiple platforms, and the loss of revenue incurred by scrapping add-ons is covered by selling embedded HyperText Transfer Protocol (HTTP) servers on a per-device basis. In our work, we have used Java for a totally different purpose, which is not to facilitate client-side presentation or Web integration, but to use Java's native support for distributed computing, remote class downloading and object serialization to implement dynamic and intelligent network monitoring. REV and COD paradigms are used. However, it makes perfect sense to include Web-based front-ends in our systems.

CORBA

The Object Management Group (OMG), faced with the issue of interoperability in the object-oriented world, addressed it by standardizing the Object Management Architecture, often referred to by its main component: Common Object Request Broker Architecture (CORBA). Since OSI is object-oriented (and so is CMIP) and SNMP entities map easily on objects, it took little time for researchers to start integrating CORBA with existing network management environments. In principle, the extension of a management agent can be achieved with traditional technology based on CS paradigm, like in some CORBA-based approaches. This solution presents some relevant drawbacks. First, one could argue that the reason why the primitives offered by management agents are so poor is that agents must be lightweight, since they can be embedded in network devices equipped with limited computational resources. In this setting, adding management primitives to agents is just not desirable. Even if adequate computational resources are available, another relevant issue is the frequency of invocation of primitives. A function may be used only now and then, e.g., because its execution is needed only during periods of heavy congestion, or because its code may change slightly according to operation conditions. In these cases, hard-wiring the function into the agent just makes it bigger, wasting device resources without appreciable gain. In general, the dynamic nature of management operations demands for dynamic customizability of agent primitives, and mobile code provides the technology needed to enhance network management with the degree of flexibility needed to cope such problems.

2.4 Architecture

In our work here, we exploit the REV and COD paradigm, delegate some agents to the network elements, and manage these delegated agents. The focus of the research is the design of a dynamic adaptable distributed architecture for the network monitoring system, with communications between the adaptable network element management agents and the manager-coordinator for all the required interactions. The architecture will provide all the necessary mechanisms, languages and protocols for the dynamic definition of both the manager-to-agent network management view definition and the internal agent-to-network-element collection method definition. We also define an internal network element API, which will be used for the collection of the management information. The adaptable network element management agent will use that API, so that internal collection methods can be changed dynamically by the manager-coordinator while the whole system is in operation. Figure 2.2 depicts the specific components of our network monitoring system. These are the Network Elements, the Adaptable Network Element Management Agents and the Manager-Coordinator.

The network elements consist of the managed objects. The adaptable network element management agent provides the network Manager-Coordinator with an information view of the supported network element management information. Such an agent possesses an API, called Collection API, by which the agent can dynamically change the way the information is collected from the network element. Specific real resources can be selected for monitoring and also the frequency can be accordingly adapted, according to the current monitored status of the network and the management applications requirements.

The gathered monitored information is processed by the network Manager-

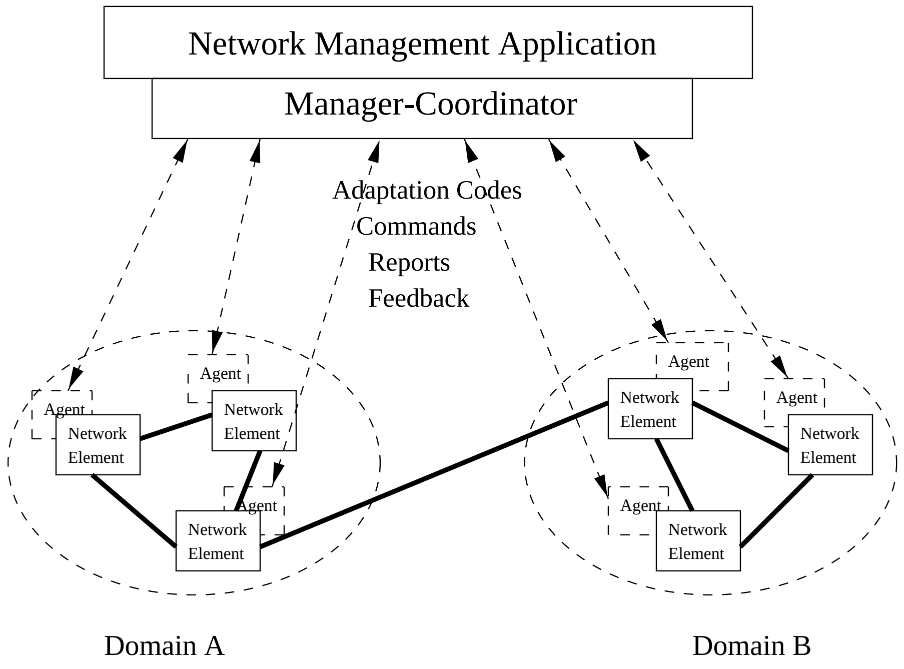

Figure 2.2: Components of network monitoring system

Coordinator and an even higher-level view is offered to network fault and performance management applications. The network Manager-Coordinator coordinates the dynamic update of the information views, provided by the agents, by specifying the specific filtering expressions and the various threshold values. When a threshold crossing is detected an asynchronous notification will be forwarded to the Manager-Coordinator. This event-based paradigm for network monitoring results in huge reduction of monitoring traffic, since statistics are no more transferred periodically from agents to manager.

The dynamic control of the monitoring system will be based on decision making and knowledge embedded in the network manager-coordinator. This will enable the network manager-coordinator to take decisions that: send to the network elements agent programs that can re-direct the data collection of the elements, change the logic of the processing within the elements, and even direct the elements to execute tests or collect new types of data. Similar levels of intelligence will be embedded in the fault and performance management applications. We will use belief networks as the basic structure for implementing this intelligence and the associated learning.

2.5 Design Considerations

This section discusses the design considerations for this adaptive distributed network monitoring system. Here we focus on the system architecture, API, the way agents could be created, deployed, and managed, and so on.

The design should consist of a set of facilities that allow a remote manager to collect data from the network element in such a way that changes to the collection method can be made at run-time. Additionally, it allows run-time extension of the network element behavior at statically defined points within the network element code through callbacks. The embedded environment requires the delegated agent(s) to function with limited computational and memory resources at high data rates.

Agent Management

From the point of view of the manager, it is important to be able to manage the delegated agents. Management of the agents includes deploying and terminating them. It is also essential that the agents be able to send messages back to the manager. The manager should be able to send commands to the agents as well, perhaps to adjust some parameter of the agents. Figure 2.5 identifies the agent management functions.

- *Distribute Logic as Agents:* Sends agent collection logic across the network from the Manager-Coordinator to the network element. Must support dynamic linkage onto the network element.

- *Terminate Agent:* For an agent that has been deployed by invoking the Distribute Logic as Agents use case and still exists in the system, causes the Agent to be removed from the system.

- *Control Agents:* For agents that have been deployed with the Distribute Logic as Agents use case, send a command to the Agent instructing it to perform a generic action. The actual command to be performed will be determined by the specific command and Agent implementations.

- *Provide Feedback from Agents:* For an agent that has been deployed by invoking the Distribute Logic as Agents use case and still exists in the system, causes feedback to be sent from the agent back to the manager that created it.

Agent Functions

From the point of view of the deployed agents, being able to read values and write values from and to the network element is critical. Also, it is necessary for the agents to be able to collect data across arbitrary data structures. Navigating these data types, such as queues and hashtables, could usually be performed through some native API associated with the abstract data type in the native processes. For this purpose, we shall include a facility to allow agents to call

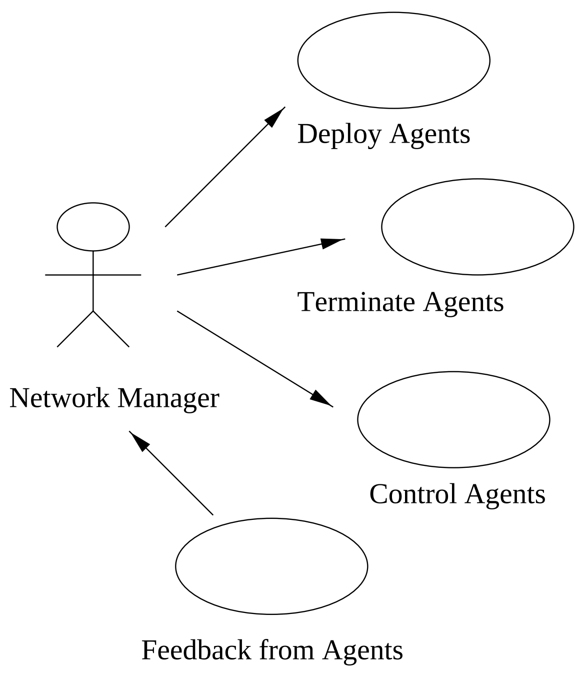

Figure 2.3: Agent Management

functions defined in the network element itself. Figure 2.5 identifies the main use cases that the agent performs.

- *Read Values:* For an agent, which has been deployed using the Distribute Logic as Agents use case and has the address of what data it is looking to read, read values from the address space of the Network Element.
- *Write Values:* For an agent, which has been deployed using the Distribute Logic as Agents use case and has the address of what data it is looking to write, writes values into the memory space of the Network Element.
- *Call Functions on the Network Element:* Suppose an agent that is going to do the actual function call has been deployed. Invoke a native C/C++ function defined in the Network Element.

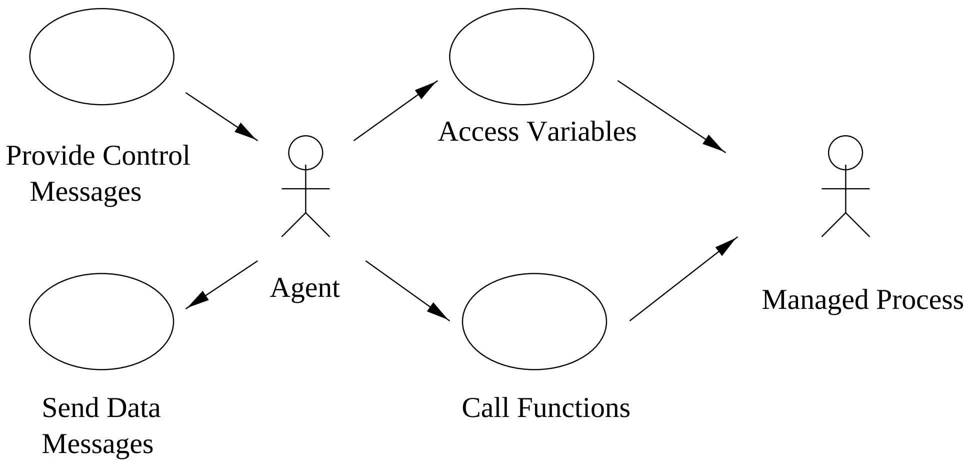

Figure 2.4: Agent Functions

Collection API

To enable the above functions, we need an interface between the network element native software and the delegated agent, with which the agent can define the specific set of resources that are considered useful to be monitored. The collection API must be able to support a range of data structures like queues and hash-tables.

If we know the address of any variable, we can read or assign its value. Of course, this implies that we know the type of variable we are dealing with, which probably requires the source code to be available. To access variables in such a way requires that we can find the addresses of the variables of interest. This is accomplished by examining the symbol table of the compiled code.

For the symbol table examination to work, we require that there is some way to extract addresses from the compiled code. For example, Solaris UNIX provides an 'nm' (name mangle) utility that allows the listing of symbols in an executable. We could use such utilities to create a directory of variables. A directory service will provide variable lookup by name; it also includes addresses of functions and function pointers. This address extraction is also possible with standard dynamic linkage mechanisms as provided by VxWorks [54] and Solaris.

On Extensions and Callbacks

As we stated in chapter 1, we could dynamically extend the functionality of the delegated agents or even the native software. The awareness of the need for extension is from human, not from the delegated agents or native processes themselves. It is the administrator again that determines what functionality to add.

For the delegated agents, since the manager has full control of the agents' lifecycle, it is always a good option for the manager to create new agents with appropriately added functionality to replace the old ones by killing the old agents and deploying new agents. All the development is at the manager site and there is no need to recompile any code at the network element site.

For native software, on the other hand, we don't have the luxury to put extra code off-line onto the original code at the manager site and deploy to network elements as a whole piece, without any recompilation. Here, the native code was already compiled and fixed; what we can do at the manager site is only to design and deploy the added code in its own fragment. We need a way to make sure that this added code could cooperate with the original code to have expected performance.

First, callback hooks are defined at certain places in the native code. Such hooks could be the places where the developer is reasonably suspicious that additional functionality may be needed later, but what he/she does not yet know at the time of software development. Defining a hook could mean putting an empty function at a certain place in the native code. But theoretically, all functions in the native code could be thought of as suspects, even though we would usually not be so suspicious. Such hooks represent the possible places to add new functionality and they are the only locations where additional functionality could possibly be integrated.

If callbacks are determined as needed by administrator, the defined callbacks will be deployed and dynamically linked into the network element code by replacing the corresponding empty function at appropriate hook. Obviously, we need access to the function pointers in order to achieve the function replacement. Such function pointers could be obtained by the directory service or dynamic linkage mechanisms as described above. Then, in the native software, this added code would be executed the next time this particular hook is encountered. Figure 2.5 illustrates these ideas.

- *Declare Callback:* The Network Element declares the hooks in its native processes for the callbacks to hook up.
- *Define Callback Operation:* Given that the Network Element has invoked the Declare Callback use case, the Manger-Coordinator defines a procedure to be used whenever a particular callback is invoked. Such a procedure is delegated to the corresponding Network Element by invoking the Distribute Logic as Agents use case
- *Perform Callback:* Causes a procedure defined by the Manager-Coordinator and deployed as an agent to be called by the Network Element.

To better understand callback mechanisms, we model software processes us-

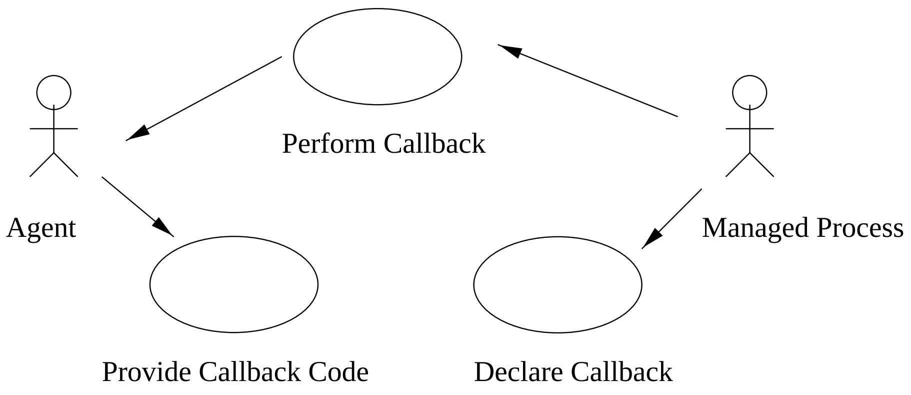

Figure 2.5: Network Element Callbacks

ing communicating finite state machines [55, 56]. A finite state machine (FSM) consists of a finite set of states with one initial state, input set of the machine, and transition function, which is a partial function from the states and inputs to its states. An extended finite state machine (EFSM) introduces state variables in addition to the explicit states. These variables can take on a number of values themselves and they are treated as *implicit* states. The complete set of the states, or *global state,* of a process instance is now the union of the explicit and implicit states. When performing analysis, however, our focus is mainly on *system state,* which consists of explicit states, plus the status of enabled transitions from those states. The problem of state explosion is avoided as such.

Each native process is modeled as an EFSM. Each process instance has its own memory space that is under its own control. No other process instances are allowed to change the values of its variables. Such variables are called local variables. Local variables are further categorized as either state variables or temporary variables. State variables are those implicit states that represent the information accumulated and retained by a process. State variables are persistent, meaning that from the perspective of each process, they retain their value over time. One example is the variable that captures some certain statistics of interest, e.g. number of packets in the queue. Temporary variables are used to store information that does not require persistence. For example, an integer variable used as the index of a loop is typically treated as a temporary variable.

Apart from local variables, we also define some variables that are visible to all process models and they are called shared variables. Shared variables could be typically implemented via header files, in a C/C++ programming environment, and they are used as a communication mechanism among the process instances. Another way to model inter-process communication is message passing via input buffers. The management procedure of the input queue could also be modeled as an EFSM and to this end, we claim that our communications between EFSMs are all through shared variables.

There are no clear rules on the use of explicit states and state variables, as this is often a matter of design and depend very much on particular application. In our software process modeling, we use explicit states to represent the top-level modes or stages that a process can enter. Such a mode could be any waiting or inactive status, or it could be a decision place that leads to different situations. To facilitate callbacks, we also identify those places where some unanticipated situations might happen and where we might later put added functionality. We call these places *callback hooks*. Specifically, we associate each callback hook with two states: one is called pre-callback state and the other post-callback state. There is a transition from pre-callback state to post-callback state, with TRUE as the predicate and the callback function as the *action* associated with this transition. At the time of software process design, the callback functions could be just empty functions, and the pre-callback state and post-callback state look identical in the sense that all the accessible local/shared variables are the same. Right after each post-callback state, we put a decision state to accommodate the possible multiple branches the process may lead to. The different branches defined over local/shared variables are mutual exclusive and exhaustive.

The parameters in the declaration of such empty functions are visible to an external entity. Symbol table examination via directory service or dynamic linkage mechanisms, as discussed above, provides a scheme to access the shared variables and the stack. The external callback function will use these local/shared variables to fulfill some added logic, and probably, some changes will be made on them since the function call is by reference. After this added external function is executed, the post-callback state will usually be different from that before this external callback function execution, in terms of the enabling branches from the decision state that follows. Such a mutual exclusive and exhaustive decision state ensures that there will always be a valid progress route for the process to move ahead, with or without addled logic.

Many callback places in a process model may use the same function call, like DoCallbacks (CallbackID, shared_variables, state_variables, temporary_variables), and we assign for each callback hook a unique CallbackID for identification. In our second and third system designs to be discussed in the next section, we proof-tested this idea of *extension via callbacks* using function replacement. Essentially, the callbacks extended the processing logic of the original state machine.

Conceptually, our callback mechanism is similar to the idea of protocol boosters [57]. It is a supporting agent and by itself is not a process or protocol. Beyond protocol boosters, it handles some unanticipated situations. To make sure that the augmented system work well for both original and new conditions, we need to investigate some certain syntactic and semantic properties using system state analysis, based on the identification of states as discussed above. Such issues are beyond the scope of this thesis.

2.6 Three System Designs

2.6.1 Java-based System Design with MIB

Taken individually, the characteristics of Java can be found in a variety of software development platforms. What's completely new is the manner in which Java technology and its runtime environment have combined them to produce a flexible and powerful programming system.

- *Code on Demand:* The dynamic feature of our framework requires support for code-on-demand (CoD) paradigm, i.e., the network element management agent can download and link on-the-fly the code from some class server to perform a given task. Traditionally, CoD is only supported by mobile code languages (MCL), such as Telescript [51]and Agent Tcl [50], for mobile agent programming. With the dynamic class loading and linking ability, Java is actually a weak MCL that directly supports CoD.

- *Distributed Computing:* The interactions between manager applications and network element management agents require distributed computing support from the programming language. Java provides native distributed programming API through Remote Method Invocation (RMI). Compared with other distributed programming platforms such as DCOM or CORBA, Java RMI is the easiest to learn and use.

- *Cross-platform Compatibility:* Without a common platform, we have to write code with the same or similar functions for each of these platforms separately, which is very costly, time-consuming and error-prone. Java's architecture-neutral and portable ability makes it an ideal platform base for a hybrid system like our network-monitoring framework.

In this design, agents are Java codes created at the Manager-Coordinator site and deployed to the network elements via RMI. We assume that JVM is available both at Manager-Coordinator site and network element site.

In our first design here, we focus on legacy systems where SNMP is used. The SNMP agents collect raw data from network elements and store those data into Management Information Base (MIB). Our work here is to design a Java-based Extensible Management Server (JEMS) that runs as a server process at the network elements to host the delegated Java agents, which in turn, could access the MIBs and carry out their predetermined functionality. Figure 2.6.1 depicts the architecture of JEMS.

Remote Delegation Interface (RDI) is the interface through which management applications can remotely delegate Java objects, exchange information with these objects and control their execution. Management Information Tree (MIT) is a container that holds Java objects in a tree structure. Two kinds of objects are stored in the MIT. Intelligent Management Objects (IMO) are delegated by remote managers to the JEMS; they perform monitoring and control functions, and interact with remote managers via the RDI. INFormation Objects (INFOs) store management information in an object-oriented way; they are used by IMOs to implement management functions. Delegation service Provider (DSP) is an RMI server object that uses the MIT and the class loader to implement the RDI;

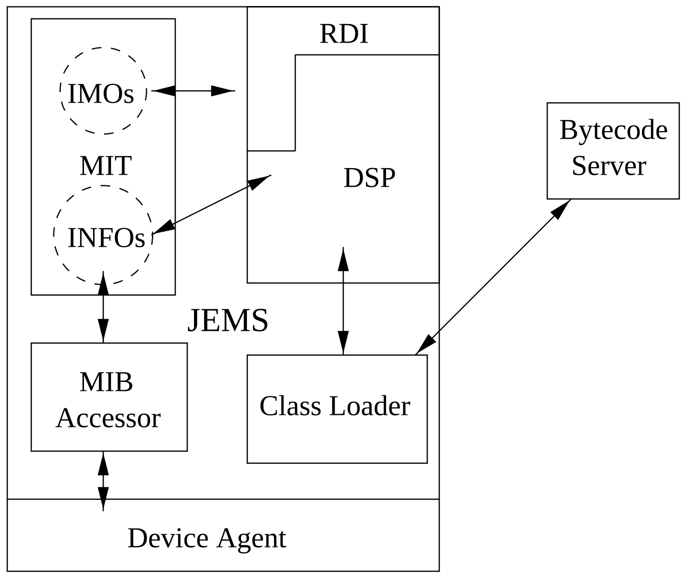

Figure 2.6: The JEMS Architecture

it provides the delegation service needed by remote managers to delegate, control and communicate with IMOs. Class loader is an internal Java object used by the DSP to load, either locally or from some class server (a.k.a. bytecode server) on the network, those classes that are needed to instantiate corresponding delegated objects. MIB Accessor is another internal Java object used by INFOs to exchange low-level management information with the local MIBs. For more information about this system design, we refer to Xi's master thesis [58].

2.6.2 Java-based System Design without MIB

The previous design is suitable for network elements equipped with MIBs, i.e. routers and switches. In many cases, however, network elements are not equipped with such MIBs. Even worse, these network elements may be equipped with only minimum amount of memory and computing resources, e.g. VSAT terminals for satellite communication networks. This Java-based design and the next native code based design are for such situations. One important issue of using Java in this case is that the network element side JVM has to be lightweight. We simply could not assume that we could ship the whole suite of the standard JVM down there. Specific versions of JVM are needed. For example, if the real-time operating system of the network elements were VxWorks [54], Personal Java suite was ported onto VxWorks. Another alternative is the so-called KVM [59], or Kilobyte Virtual Machine, that encapsulates only the core JVM and APIs. Such a KVM suite would typically require about 150KB memory, which is not stringent.

At the manager site, on the other hand, we could install as much sophisticated a suite of Java system as we need since the manager station would typically possess quite enough memory and computing abilities. Yet we have to make sure that the functionality of all the Java agents created here can be supported by the network element side embedded Java system.

As to the collection API, Java Native Interface (JNI) could be utilized to help the communication between Java agents and the native C/C++ processes embedded in network elements. With addresses of functions and function pointers, we can provide callbacks and function replacement. For a function that is accessed through a function pointer, we can replace the function, by simply redirecting the pointer to our own replacement code. In the same way, we can provide a callback service. A function pointer can be declared in a native process, which calls this function pointer whenever it wants a callback. The function pointer in the beginning points to a null operation. Only when the Java code replaces the function will the callback be ready. Finally, we come full circle with the communication between Java and the target process by allowing a Java client to call functions from the target process. As mentioned previously, the directory service also has addresses of functions and the Java client can call these functions using JNI and the function addresses.

In summary, we use global variables as shared memory. So any variables of interest to the Java client should be global, and any functions to be replaced should be accessed through a global function pointer. Function pointers should be declared as a hook for us to create callbacks. Finally, we assume there is some way of extracting memory addresses from the symbol table similar to the name mangle utility in Unix.

2.6.3 System Design based on Native Code Technology

The last design based on Java technology assumes that the global variables and the processes are in a common address space. If, however, the network elements have multiple processors and separate address spaces, we have a different situation. Simply delegating Java agents to each address space would incur little extra work, but if we wish to change the native processes' logic and do the callbacks, things get worse. In this situation, the only way for a native C/C++ code to perform a callback onto Java is for a JVM to be running on each processor. This does not seem practical. Also, in order for Java to interact with native C/C++ processes, it is necessary to use JNI, which incurs a great deal of overhead. This interface is quite limited and not particularly easy to use.

A native C/C++ implementation has neither the multiple JVM burden, nor the overhead and difficulty of a JNI implementation. And here we use native C/C++ codes in the system design. In order to read, write or call functions, the ordinary dereferencing of the pointers or function pointers will suffice, assuming that the Agent has the correct addresses in the memory of the network element.

This design requires Inter Process Communication (IPC) in two distinct places. One is between the network element and the management site, and the other is between the processes running on the network element in different address spaces. For the former, it seems sensible to use Remote Procedure Call (RPC) or just an ordinary TCP/IP socket. The IPC between processes on the network element however, should not use sockets, which have much more overhead than required. We use shared memory based message queues with some semaphores to handle the IPC on the network element.

The network element code links the Agent code dynamically, so the network element resolves Agent symbols dynamically through ordinary dynamic linkage mechanisms. However, the Agent code does not have a dynamic mechanism to support the lookup of symbols in the target process. The solution to this problem is to look at the statically defined symbol table of the process residing in the executable code. As stated above, Solaris UNIX provides an 'nm' (name mangle) utility that allows the listing of symbols in an executable. Since the Agent code shall be defined in C/C++, we can provide a feature that takes the output of 'nm' and construct a directory based on it. Once again, this could be done via dynamic linkage mechanisms. Extensions are realized via function replacement, as in the previous design.

2.7 Tradeoffs between System Designs

During the processes of our systems designs, we encountered many issues that we need to balance between various design options. Choosing Java or native code technology is one of the most important considerations. Here we list the advantages and disadvantages of using Java or native code for the system designs.

Table 2.2: Java System Design

Advantages	**Disadvantages**
• Distributed computing via RMI	• One memory space assumption
• Dynamic class loading	• JVM memory footprint
• JVM availability and portability	• Callbacks are cumbersome
• Mobile computing support	• Overhead of JNI
• Exceptions handling	• Inflexible low-level synchronization

Table 2.3: Native Code System Design

Advantages	**Disadvantages**
• Allows multiple processors	• No RMI support
• Clear and neat design	• Handles IPC explicitly
• Callbacks via dynamic linkage	• No outsource dynamic class loading
• No JNI overhead	• No portability
• Flexible low-level synchronization	• Explicit exceptions handling

2.8 Traffic Analysis of the Proposed System

In this section, we derive a model for the traffic generated by network management. It is rather general, and has to be adapted to the particular management scenario where the manager operates. This scenario must take into account the actual management protocols in place, as well as the technology actually used to implement a given paradigm. This section borrows many statements from the paper by Baldi and Picco [60].

Let N denote the number of managed devices. The complexity of the management task is taken into account by the number Q of queries performed on the uniform data access interface, e.g., MIB. The transmission overhead introduced by protocol encapsulation, and possibly traffic control or connection setup, is taken into account down to the network layer. If a chunk of data of size X is to be transmitted at the application layer, we represent the actual amount of data exchanged at the network layer as $X' = \eta(X)X$ where $\eta(X) > 1$. $\eta(X)$ is called the overhead function since it accounts for the control information (protocol overhead) added to X by the network and the above layers. In the following, we write ηX in place of $\eta(X)X$ in order to simplify formulae.

2.8.1 Model for the Overall Traffic

We derive here a model for the traffic generated by a management task that involves retrieving a set of data from managed devices. We assume that the same management operations are executed on each device. This is not necessarily the case for every management task, but allows for a simpler notation without compromising the generality and significance of what can be inferred from the model.

With the traditional client-server paradigm, the NMS requests an operation to the management agent by sending a request message to it. We assume that such information is of size I_q. In order to perform the management task, Q request messages must be sent to each of the N managed devices. Device n

answers the q^{th} request with a reply whose size is R_{qn}. The overall traffic is then

$$T_{CS} = \sum_{n=1}^{N} \sum_{q=1}^{Q} (\eta_{CS} I_q + \tilde{\eta}_{CS} R_{qn}). \tag{2.1}$$

If the REV paradigm is exploited, the Q requests are embedded in a code fragment of size C_{REV} sent to managed device n. Remote evaluation of the code produces the Q results R_{qn} which are sent back collectively to the NMS. This pairwise interaction has to take place for each of the N managed devices. The overall traffic generated is then given by

$$T_{REV} = \sum_{n=1}^{N} (\eta_{REV} C_{REV} + \tilde{\eta}_{REV} \sum_{q=1}^{Q} R_{qn}). \tag{2.2}$$

In the MA paradigm, the NMS unleashes a mobile component that visits each of the N nodes and collects information locally. We model the code and the portion of the state needed for its execution (C_{MA}) as separate from the portion of the state relevant to the application. The latter grows as long as this agent travels from node to node. In fact, if we denote with $S_{MA,n}$ the size of the state of the agent during the trip towards node n, then

$$S_{MA,n} = \begin{cases} 0 & \text{if } n = 1 \\ \sum_{m=1}^{n-1} \sum_{q=1}^{Q} R_{qm} & \text{if } n > 1 \end{cases} \tag{2.3}$$

After information on the last node has been collected, the mobile agent sends back to the NMS all the results collected. Thus the overall traffic generated is given by

$$T_{MA} = \sum_{n=1}^{N} \eta_{MA} (C_{REV} + S_{MA,n}) + \tilde{\eta}_{MA} \sum_{n=1}^{N} \sum_{q=1}^{Q} R_{qn}. \tag{2.4}$$

With the COD paradigm, the NMS requests an operation by sending a message that contains the operation signature, I_{COD}. If the operation has already been installed on the managed node, a reply is sent which contains the result of the Q queries, like in a REV implementation. On the other hand, if the code for the operation has not been installed yet, the agent replies with a message of size I_{fetch} requesting the dynamic download. The code, of size C_{COD}, is transferred and linked on the agent device, and it becomes available for future invocations, and the corresponding operation is performed. Consequently, the expression of generated traffic at equilibrium is:

$$T_{COD,stable} = \sum_{n=1}^{N} (\eta_{COD} I_{COD} + \tilde{\eta}_{COD} \sum_{q=1}^{Q} R_{qn}). \tag{2.5}$$

During the setup phase, the generated traffic is:

$$T_{COD,setup} = \sum_{n=1}^{N} (\tilde{\eta}_{COD} I_{fetch} + \eta_{COD} C_{COD}). \tag{2.6}$$

The overall traffic with a setup phase is simply $T_{COD} = T_{COD,stable} + T_{COD,setup}$.

2.8.2 Evaluations on the Overall Traffic

Let us first compare CS and REV paradigms. The REV paradigm is preferable than CS paradigm only if $T_{CS} \geq T_{REV}$. After elaboration of (2.1) and (2.2) and let $\bar{I} = \frac{1}{Q}\sum_{q=1}^{Q} I_q$, and $\bar{R} = \frac{1}{Q}\sum_{q=1}^{Q} R_{qn}$, we obtain

$$NQ\eta_{CS}\bar{I} + NQ\tilde{\eta}_{CS}\bar{R} \geq N(\eta_{REV} C_{REV} + \tilde{\eta}_{REV} Q\bar{R}).$$

It is likely that $Q\tilde{\eta}_{CS}\bar{R} \geq \tilde{\eta}_{REV} Q\bar{R}$ since usually a fixed overhead is associated to each packet and thus, the longer the message being segmented, the smaller the relative overhead. Hence, if more results $\bar{R}$ can be transmitted together in a single message, the overhead is likely to be smaller, although depending on the protocol used to implement communications in CS and REV. We call the difference in the overhead introduced to send the results of the queries $\Delta O_{CS,REV} \geq 0$, and REV is more convenient than CS if

$$\eta_{REV} C_{REV} \leq \Delta O_{CS,REV} + Q\eta_{CS}\bar{I}, \tag{2.7}$$

that is, if the size of the message containing the code to be evaluated remotely is smaller than the overall size of the message requests needed in a CS paradigm plus the difference in overhead introduced when transmitting the result back. Clearly, REV is convenient when the number of instructions Q needed to perform a query is high and C_{REV} effectively compacts the representation of the local interactions I_q within the code, e.g., using loop control structures.

By applying the same reasoning and assumptions, we see that a MA implementation always generate more traffic than a REV one. From (2.2) and (2.4) we obtain

$$N\eta_{MA}C_{MA} + \tilde{\eta}_{MA}QN\bar{R} + \sum_{n=1}^{N} \eta_{MA}Q(n-1)\bar{R} \geq N(\eta_{REV}C_{REV} + \tilde{\eta}_{REV}Q\bar{R}).$$

If Q is sufficiently large and $\tilde{\eta}_{REV} = \tilde{\eta}_{MA}$, we have $\tilde{\eta}_{MA}QN\bar{R} \approx N\tilde{\eta}_{REV}Q\bar{R}$. Without loss of generality, let $\eta_{MA} = \eta_{REV}$. Since usually $C_{MA} > C_{REV}$, we could easily see that the above inequality always holds. That is, REV is always more convenient than MA, because the latter must carry the state which is growing at every hop.

The application of COD paradigm depends on the frequency of invocation which has not been considered yet. So far, we have given the expression of the traffic generated for a single execution of a management task. However, in general it may be interesting to consider how this varies over U different invocations. For the other paradigms, this additional parameter does not affect the expression of the traffic. So for $p \in \{CS, REV, MA\}$, $T_p(U) = UT_p$. And the traffic generated by a COD paradigm is

$$T_{COD}(U) = T_{COD,setup} + UT_{COD,stable}.$$

Calculation of $T_{REV}(U) \geq T_{COD}(U)$ under the likely assumptions that $\bar{I} \approx$

$I_{COD} \approx I_{fetch}$ and $\tilde{\eta}_{MA} Q \bar{R} \approx \tilde{\eta}_{REV} Q \bar{R}$ yields

$$\eta_{REV} C_{REV} \geq \frac{U+1}{U} \eta_{COD} \bar{I} + \frac{1}{U} \eta_{COD} C_{COD}.$$

Clearly, if U is large, i.e. the primitive is invoked many times before being upgraded or discarded, the inequality above can be approximated by $\eta_{REV} C_{REV} \geq \eta_{COD} \bar{I}$, which is always satisfied, the threshold being a REV code composed by a single instruction. So if a function is used many times, caching its code saves bandwidth. That is why we have chosen to combine REV and COD paradigms and ship some of the processing intelligence and make them resident to the network elements in order to save the bandwidth. For those functions that are used only now and then, REV paradigm suffices.

2.8.3 Application to Satellite Communication Networks

Now let us look at a more realistic case for our hub-based satellite communication network. Suppose we want to manage only the terminals. Terminals communicate with each other through an up-link and a down-link. A natural way to relate network traffic with the communication cost associated to the links is to assign to each link a weighting cost coefficient $0 \leq \lambda_l \leq 1$. For each terminal n, let λ_{nh} denote the cost coefficient of the up-link to the hub and λ_{hn} denote the cost coefficient of down-link. The value of the cost coefficients have to be determined by the administrator according to the notion of cost associated to the link, and may be actually a combination of several factors. For instance, a high cost may be due to high latency or low-bandwidth on the link, or to the fact that a link connected to the NMS should be kept as unloaded as possible, or to security considerations. As discussed before, MA paradigm is not considered here, and we list the cost related to the overall traffic generated by the other three paradigms as follows:

$$K_{CS} = \sum_{n=1}^{N} \lambda_{hn} \sum_{q=1}^{Q} \eta_{CS} I_q + \sum_{n=1}^{N} \lambda_{nh} \sum_{q=1}^{Q} \tilde{\eta}_{CS} R_{qn} \tag{2.8}$$

$$K_{REV} = \sum_{n=1}^{N} \lambda_{hn} \eta_{REV} C_{REV} + \sum_{n=1}^{N} \lambda_{nh} \tilde{\eta}_{REV} \sum_{q=1}^{Q} R_{qn} \tag{2.9}$$

$$K_{COD,stable} = \sum_{n=1}^{N} \lambda_{hn} \eta_{COD} I_{COD} + \sum_{n=1}^{N} \lambda_{nh} \tilde{\eta}_{COD} \sum_{q=1}^{Q} R_{qn} \tag{2.10}$$

$$K_{COD,setup} = \sum_{n=1}^{N} \lambda_{nh} \tilde{\eta}_{COD} I_{fetch} + \sum_{n=1}^{N} \lambda_{hn} \eta_{COD} C_{COD}. \tag{2.11}$$

To compare $K_{CS} \geq K_{REV}$, we elaborate (2.8) (2.9) and obtain

$$\sum_{n=1}^{N} \lambda_{hn} (\eta_{REV} C_{REV} - \eta_{CS} Q \bar{I}) \leq \sum_{n=1}^{N} \lambda_{nh} (Q \tilde{\eta}_{CS} \bar{R} - \tilde{\eta}_{REV} Q \bar{R}),$$

which says that the down-link weighted difference between the size of the code to be evaluated in REV and the overall size of the message requests in CS is smaller than the up-link weighted difference in overhead introduced when transmitting the results back. The right hand side of the inequality is positive, as discussed before. Clearly, REV is convenient when the number of instructions Q needed to perform a query is high and C_{REV} effectively compacts the representation of the local interactions I_q within the code.

2.9 Conclusions

In this chapter, we presented a distributed, extensible framework for supporting adaptive, dynamic network monitoring and control. After researching the major design paradigms and alternative technologies, we have chosen the mobile code technology, in particular REV and COD paradigms, to help implement such a framework. The focus of our work has been on three aspects. First, the design of the standard infrastructure, or Virtual Machine, based on which agents could be created, deployed, managed and initiated to run. Second, the collection API for our delegated agents to collect data from network elements. Third, the communicating finite state machine based callback mechanism through which the functionality of the delegated agents or even the native software could be extended. We proposed three system designs based on such ideas.

Our first design, as discussed in section 2.6.1, uses full-blown JVM in both manager and network element site and assumes the presence of MIBs. It is a proof-of-concept design and is suitable for network elements equipped with powerful computing and memory capabilities, i.e. routers and ATM switches. Here we use the off-the-shelf JVM and we do not need to access the network element native software directly; instead, we need only to access the MIBs that store the raw monitoring data. Our prototype system works well, which encouraged us to research further into the Virtual Machines and collection API issues.

In our second design, as shown in section 2.6.2, we consider the situations where there is no MIB embedded with network elements. We still use JVM but here our focus is on the network elements equipped with limited computing and memory capabilities. Specific versions of JVM are considered. For the delegated Java agents to access the native software, Java Native Interface (JNI) is exploited and a directory containing addresses of the native global variables and function pointers is set up. The processing logic of the delegated agents could be extended by creating new agents with the desirable functionality, followed by deploying them to the network elements to replace the old agents. To extend the native software functionality, we carry out function replacement by swapping the function pointers of the Java agents and the corresponding native code functions.

Further, in our third design discussed in section 2.6.3, we remove the convenient JVM for those network elements equipped with multiple processors and address spaces. The focus here is to use dynamic linkage technology to emulate the Virtual Machine concept. The delegated agents are dynamically linked to the native code by the C/C++ run-time environment. The collection API in this case is very thin since all that is needed is to access the native code directly. The extension of functionality is similar as the second design, with the difference that we do not need JNI in this case. It is a neat design with respect to a pure C/C++ environment, but without JVM, it loses Java's portability. This design is suitable for those resource limited network elements that run over a real-time operating system and will not use Java as the native code development. An important advantage of this design is that large amount of data can be processed quickly via this native code callback mechanism, as compared with Java-based designs.

Our system designs could be easily adapted to accommodate the possible hierarchical networks and, consequently, hierarchical network management architecture. We could simply treat the manager-coordinator, shown in figure 2.2, as a mid-level manager that reports to a higher-level manager. This mid-level manager acts as agent role for the higher-level manager, and as manager role for the managed network domains.

Now that we have the framework for adaptive, distributed network monitoring and control, our next step will be focused on the intelligence part of network management. In particular, we are interested in fault and performance management using such a framework.

Chapter 3

Belief Network as the Probabilistic Fault Model

In this chapter, we first give a brief introduction to belief networks, including the definition, semantics and inference algorithm. Then, we discuss the reason why we choose belief networks as the probabilistic fault model for our fault diagnosis purpose.

3.1 A Brief Introduction to Belief Networks

3.1.1 Definition

A belief network, also called a Bayesian network or a causal network, is a graphical representation of cause-and-effect relationships within a problem domain. More formally [26],

Definition 3.1.1 *A belief network* $\mathcal{B}=(V, L, P)$ *is a Directed Acyclic Graph (DAG) in which the nodes* V *represent variables of interest (propositions), the set of directed links* L *represent the causal influence among the variables and the* *parents of a node are all those nodes with arrows pointing to it, and the strength of an influence is represented by conditional probabilities tables (CPTs) attached to each cluster of parent-child nodes in the network.*

Let us look at one example. Suppose you have a new burglar alarm installed at home. It is fairly reliable in detecting a burglary, but also responds on occasion to minor earthquakes. You also have two neighbors, John and Mary, who have promised to call you at work if they hear the alarm. John always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, rather likes loud music and sometimes misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary. This simple scenario is illustrated in figure 3.1 [61]. The letters B, E, A, J, and M stand for Burglary, Earthquake, Alarm, JohnCall, and MaryCall, respectively. Node B and E do not have any parents, and the tables associated with them simply represent the prior probabilities of Burglary or Earthquake. Note that we omit the entries $P(\bar{B})$ and $P(\bar{E})$ in the tables since they can be trivially obtained from $P(B)$ and $P(E)$. Node A has two parents, B and E, and the conditional probability table represents the probabilities of alarm going off under different configurations of parent nodes. When there is both burglary and earthquake, with 0.95 probability the alarm will go off. If there is burglary there, but no earthquake, the probability is still 0.95. This reflects our understanding that, if in a geographical area where severe earthquake is very rare (surely not Los Angeles), earthquakes are usually mild and they do not increase the chance for the alarm to go off as long as there is burglary inside the house. However, earthquake will play a role under the situation of no burglary, and we see that the chance for the alarm to go off is 0.29, considerably larger than 0, but far not as comparable as the influence from burglary. Finally, the alarm will rarely go off if there is neither burglary or earthquake, with a probability of 0.001.

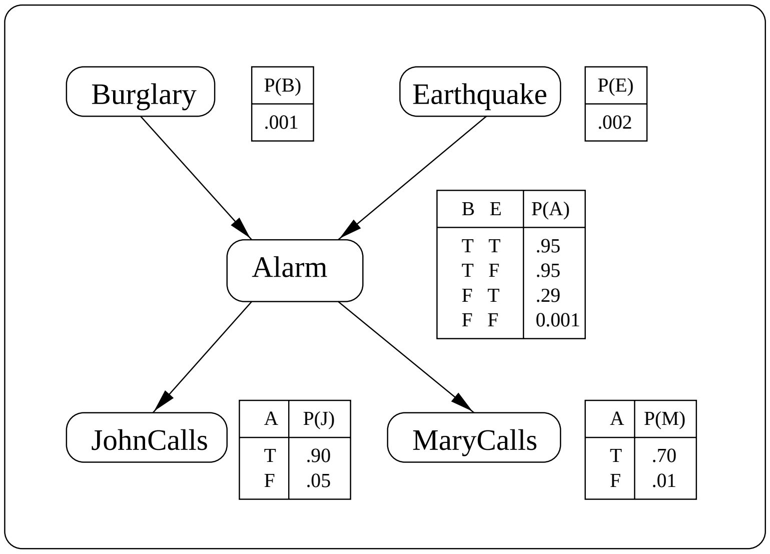

Figure 3.1: An example of Belief Network

3.1.2 The Semantics of Belief Networks

Representation of Joint Probabilities

A belief network for problem domain $\{x_1, ..., x_n\}$ represents the joint probability distribution (JPD) over those random variables. Based on the chain rule of probability, we have

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}). \tag{3.1}$$

Given a DAG G and a JPD P over a set $\mathbf{x} = \{\mathrm{x}_1, ..., \mathrm{x}_\mathrm{n}\}$ of discrete variables, we say that G *represents* P if there is a one-to-one correspondence between the variables in $\mathbf{x}$ and the nodes of G, such that

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i|\Pi_i). \tag{3.2}$$

If we order the nodes in such a way that the order of a node is larger than those of its parents and smaller than those of its children (the so-called *topological ordering*), we have

$$p(x_i|x_1, ..., x_{i-1}) = p(x_i|\Pi_i), \tag{3.3}$$

which means given its parent set $\Pi_i \subseteq \{x_1, ..., x_{i-1}\}$, the set of variables that render x_i, each variable x_i is conditionally independent of all its other predecessors $\{x_1, ..., x_{i-1}\}\backslash\Pi_i$. In other words, for any node in the DAG, given its parents, that node is conditionally independent of any other node that is not its descendent. This conditional independence makes a belief network model a compact representation of the joint probability distribution P over the interested variables.

Representation of Conditional Independence Relations

We have described above the conditional independence of a node and its predecessors, given its parents. But, is this the only and general case of conditional independence? In other words, given a set of evidence nodes E, is it possible to "read off" whether a set of nodes in X is independent of another set Y, where X and Y are not necessarily parents and children? Fortunately, the answer is yes and the methods are provided by the notion of **d-separation**, which means direction-dependent separation [26]. It is an important notion in designing inference algorithms.

Definition 3.1.2 *Let X, Y and Z be three disjoint subsets of nodes in a DAG G, then Z is said to* **d-separate** *X and Y, iff along every undirected path from* *each node in X to each node in Y there is an intermediate node A such that either (1) A is a head-to-head node (with converging arrows) in the path, and neither A nor its descendents are in Z, or (2) A is in Z and A is not a head-to-head node.*

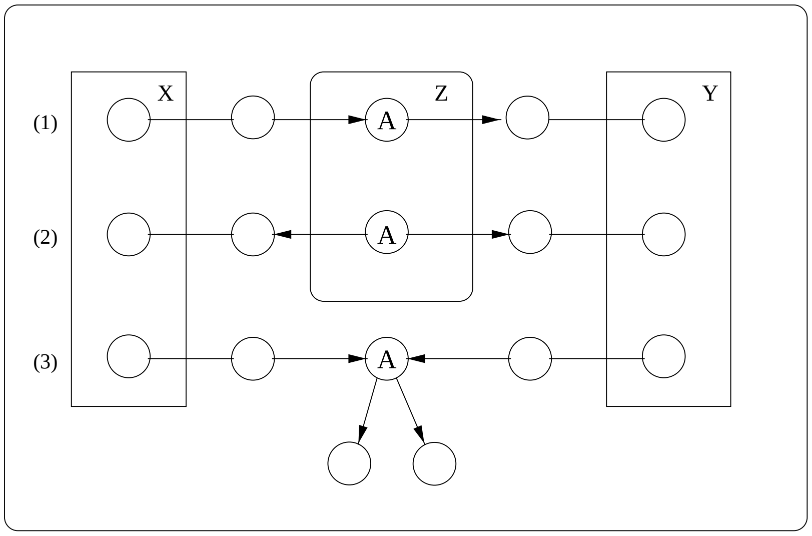

Figure 3.2: An Illustration of d-separation

The importance of d-separation is that, in a belief network, X and Y are conditionally independent given Z if and only if Z d-separates X from Y. To better understand this notion, we enumerate all of the possible cases, as shown in figure 3.2. In case (1), or the so-called serial connections case, we say evidence A will block node X and Y if it is initiated. In case (2), or the so-called diverging connections case, evidence A will block the communications between its children, node X and Y in this case, if it is initiated. In case (3), node A is a converging node and the communications between its parents will be open if it is initiated.

3.1.3 Inference in Belief Networks

By now we can say that a belief network simulates the mechanism that operate in the environment and thus allows for various kinds of inferences (also called **evidence propagation**). The question here is: How can one infer the (probabilities of) values of one or more network variables, given observed values of others? Or, in mathematics, we want to find $P(Q = q|E = e)$, where Q is the query variable set and E is the set of evidence variables. Based on the choice of Q and E, there are four distinct kinds of inference patterns, see figure 3.3.

- **Diagnosis inferences**: From effects to causes, also called backward inference. For example: "What is the most probable explanations for the given set of evidence?"
- **Causal inferences**: From causes to effects, also called forward inferences. For example: "Having observed A, what is the expectation of B?"
- **Inter-causal inferences**: Between causes of a common effect. For example: "If C's parents are A and B, then what is the expectation of B given both A and C?" Namely, what is the belief of the occurrence of one cause on the effect given that the other cause is true? The answer is that the presence of one makes the other less likely (explaining away).
- **Mixed inferences**: combining two or more of the above.

One of the most widely accepted exact propagation algorithm is based on a secondary structure called junction tree. Such an algorithm is quite general and can be used for both a polytree[1] and a multiply-connected network. It is used

[1]We call a tree as **polytree** (also singly connected network) if every node in the tree may have more than one parents.

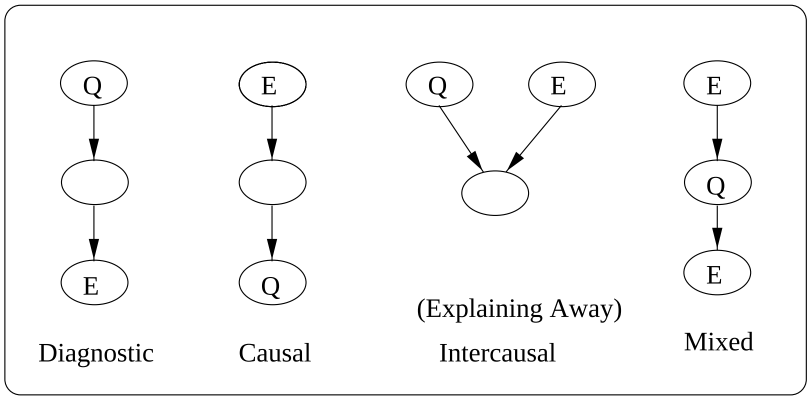

Figure 3.3: An Illustration of four inference patterns

in HUGIN [62], the expert system shell we adopted in this thesis. We give a brief introduction of this algorithm based on [63] in the next section. For more information, we refer to [63, 25, 64].

3.2 Exact Evidence Propagation by Junction Trees

In this section we give a brief introduction of the object-oriented version of the computational scheme proposed by Lauritzen and Spiegelhalter [64]. The essential idea of this method is local computation on a secondary structure, called junction tree, derived from the original probabilistic graphical model.

3.2.1 Introduction

A belief network (BN) is constructed over a *universe* U, consisting of a set of nodes each node having a finite set of *states*. The set of parents of A is denoted by $pa(A)$ and $fm(A)$ denotes the family $pa(A) \bigcup \{A\}$.

Let $V \subseteq U$. The *space* of V is the Cartesian product of the state sets of the nodes of V and is denoted by $Sp(V)$. For later notational convenience we think of the probability tables as functions, and denote them by the greek letters ϕ and ψ. If $A \in U$ then $\phi_A = P(A|pa(A))$ maps $Sp(fm(A))$ into the unit interval $[0, 1]$. Later it becomes convenient to consider functions that are not normalized and therefore take on arbitrary non-negative values. So in the sequel, ϕ and ψ denote such functions.

Let U be the universe of nodes for a BN. We define the (a priori) joint probability function ϕ_U as the product

$$\phi_U = \prod_{A \in U} \phi_A.$$

Evidence

Let V be a set of nodes. By *evidence* on V we mean a function $E_v : Sp(V) \to \mathbf{R}_0$. Thus evidence is represented as a likelihood function, giving relative weights to elements in the space of V. In the particular case, where E_V is a function into $\{0, 1\}$ it represents a statement that some elements of $Sp(V)$ are impossible. In that case we call E_V a *finding.* Typically a finding is a statement that a certain node is in one particular state.

If the prior joint distribution function for the BN is ϕ then the posterior joint probability function is defined to be $\mu(\phi * E_V)$ where μ is a normalizing constant.

The Calculation Problem

Given a BN with universe U, a set of (pieces of) evidence, and let $A \in U$. What is the probability distribution for A given the evidence?

In principle it is possible to calculate ϕ_U, multiply it with the evidence functions, and then to marginalize the resulting function to A. However, this calculation is linear in the cardinality of $Sp(U)$ and in practice intractable even for small universe. We therefore have to exploit the local structure of the network.

3.2.2 Trees of Belief Universe

The aim of the implementation of efficient methods for solving the calculation problem is to have a set of objects which can send messages to each other and can perform actions as results of received messages. In order to avoid a global control structure it is convenient to have the objects organized in a tree such that messages only can be passed between neighbors in the tree. Then a global operation can be started in any object and successive message passing to neighboring objects will spread the operation to the entire tree and stop by itself when this is done. Another effect would be that the objects can perform the tasks in parallel and thus enjoy the advantages of parallel distributed computing.

Basic Notations

A *tree of belief universes* consists of a collection $\mathcal{C}$ of sets of nodes organized in a tree. The intersection of neighbors in the tree are called *separators*. The collection of separators is called $\mathcal{S}$. Both the universes and the separators have belief potentials ϕ_W attached to them, where ϕ_W maps $Sp(W)$ to $\mathbf{R}_0$. The *joint system belief* ϕ_U is defined as a function on $Sp(U)$ given by

$$\phi_U = \frac{\prod_{V \in \mathcal{C}} \phi_V}{\prod_{S \in \mathcal{S}} \phi_S}.$$

A belief potential ϕ_W is said to be *normalized* if $\sum_W \phi_W = 1$. A *normalized tree* of belief universes is one where all belief potentials are normalized.

Construction

Now we are ready to construct the tree structures representing the same joint probability function as BNs. Let N be a BN with universe U and probability functions

$$\phi_A : Sp(fm(A)) \rightarrow \mathbf{R}_0, \quad A \in U.$$

Let further T be a tree of belief universes with collection $\mathcal{C}$ and separators $\mathcal{S}$ constructed such that

1. to each $A \in U$ we assign a $V \in \mathcal{C}$ with $fm(A) \subseteq V$;
2. for $V \in \mathcal{C}$ let $A, \ldots, B$ be the nodes to which V is assigned. Let

$$\phi_V = \phi_A * \ldots * \phi_B;$$

3. for all $S \in \mathcal{S}$ let ϕ_S be any constant positive function.

Note that to each $A \in U$ there might be several $V \in \mathcal{C}$ such that $fm(A) \subseteq V$ but we only assign one of them to A. From the construction it is easy to see that the joint system belief for T is proportional to the joint probability function for N and that the quotient between them is the product of the values for the potentials of the separators. Now the basic structure is established. The belief universes are the objects and the separators are the communication channels. Next we define the basic operations for the objects.

Absorption

Let T be a tree of belief universes with collection $\mathcal{C}$ and separators $\mathcal{S}$. Let $V \in \mathcal{C}$ and let $W_1, \ldots, W_m$ be neighbors of V with separators $S_1, \ldots, S_m$ respectively.

The universe V is said to *absorb* from $W_1, \ldots, W_m$ if the belief potentials ϕ_{S_i} and ϕ_V are changed to ϕ'_{S_i} and ϕ'_V, where

$$\begin{aligned}\phi'_{S_i} &= \sum_{W_i \backslash V} \phi_{W_i}, \quad i = 1, \ldots, m \\ \phi'_V &= \phi_V * (\phi'_{S_1}/\phi_{S_1}) * \ldots * (\phi'_{S_m}/\phi_{S_m}).\end{aligned}$$

After an absorption the belief potential for S_i is the marginal of W_i with respect to S_i. We also have that

$$\phi_V/(\phi_{S_1} * \ldots * \phi_{S_m}) = \phi'_V/(\phi'_{S_1} * \ldots * \phi'_{S_m})$$

and hence the joint system belief is invariant under absorption.

Entering Evidence

Let T be a tree of belief universes and $V \subseteq U$. Let E_V be an evidence function. This can be entered to T if there is a $W \in \mathcal{C}$ such that $V \subseteq W$. This is done simply by multiplying ϕ_W by E_V. If T is constructed from a belief network N as specified above then the posterior joint system belief for T is proportional to the posterior joint probability function for N.

Collecting Evidence

Each $V \in \mathcal{C}$ is given the action `COLLECTEVIDENCE`: When `COLLECTEVIDENCE` in V is called from a neighbor W then V calls `COLLECTEVIDENCE` in all its other neighbors and when they have finished their `COLLECTEVIDENCE`, V absorbs from them. The process takes the depth first search (DFS) paradigm in graphs [65].

Distributing Evidence

Each $V \in \mathcal{C}$ is given the action `DISTRIBUTEEVIDENCE`: When `DISTRIBUTEEVIDENCE` is called in V from a neighbor W then V absorbs from W and calls `DISTRIBUTEEVIDENCE` in all its other neighbors. This procedure also takes the depth first search paradigm (DFS) in graphs [65].

Local Consistency

A tree of belief universes is said to be locally consistent if whenever V and W are neighbors with separator S then

$$\sum_{V \setminus S} \phi_V \propto \phi_S \propto \sum_{W \setminus S} \phi_W.$$

An important prerequisite for the methods is the following:

Lemma 3.2.1 *Let V be any belief universe in a tree of belief universes. If first `COLLECTEVIDENCE` is evoked from V and then `DISTRIBUTEEVIDENCE` is evoked from V, the resulting tree of belief universes will be locally consistent.*

3.2.3 Junction Trees

What we aim for is a tree of belief universes such that the probability distributions can be directly inferred from the belief potentials without having to calculate the joint system belief. That is: If $W \subseteq V$ then $\sum_{V \setminus W} \phi_V$ is proportional to the probability distribution for W. In order to ensure this, local consistency is not sufficient. For example, if $W \subseteq V_1$ and $W \subseteq V_2$, then local consistency does not automatically ensure that

$$\sum_{V_1 \setminus W} \phi_{V_1} \propto \sum_{V_2 \setminus W} \phi_{V_2}$$

unless V_1 and V_2 are neighbors in the tree. We therefore define a tree of belief universes to be globally *consistent* if for each $V, W \in \mathcal{C}$

$$\sum_{V \setminus W} \phi_V = \sum_{W \setminus V} \phi_W$$

that is, ϕ_V and ϕ_W coincide on $V \cap W$. Clearly, a consistent tree will always be locally consistent but the converse is false in general.

Call a tree of belief universes a *junction tree* if for any $V, W \in \mathcal{C}$ and for any separator S on the path between V and W we have $V \cap W \subseteq S$. The important fact is that the junction tree property ensures that the converse to hold. In other words, *a locally consistent junction tree is consistent.* Now we have the key result of the method:

Theorem 3.2.2 *Let T be a consistent junction tree of belief universes with collection $\mathcal{C}$. Let ϕ_U be the joint system belief for T and let $V \in \mathcal{C}$. Then*

$$\sum_{U \setminus V} \phi_U = \phi_V.$$

At this point we have overcome the calculation problem stated in section 3.2.1. The theorem shows that it is not necessary to calculate and marginalize ϕ_U in order to find the belief in a particular node. When we have a consistent junction tree the joint probability function is for each universe V proportional to the belief potential for V. We can now find the belief in any node $A \in V$ by marginalizing ϕ_V to A and then normalize the result.

When evidence arrives to the BN, it is entered to the junction tree as described in section 3.2.2. Then the junction tree is made consistent by means of the operations `COLLECTEVIDENCE` and `DISTRIBUTEEVIDENCE` and posterior probabilities can be found by the above procedure.

The above discussion adds a constraint on the transformation of a BN to a tree of universes: The tree must be a junction tree. We brief the transformation procedure in the next section.

3.2.4 From BN to Junction Tree

When a BN model is completed, a junction tree T can be constructed to facilitate the efficient inference. This involves a moralization of the graph: for each node, links are added between all of its parents (if they are not connected already) and directions are removed.

The moral graph is then triangulated: Links are added until every cycle of length more than three has a chord. Based on this triangulation the junction tree is formed: The collection $\mathcal{C}$ is the set of cliques in the triangulated graph (a clique is a maximal set of nodes all of which are pairwise linked). Given the cliques the junction tree can be found through a maximal spanning tree algorithm [25].

The size of the cliques determines the runtime behavior of the system, so the triangulation is the single most important step in the transformation. In order to optimize this, heuristic methods have been developed to obtain a small total clique size [66].

The moralization of the graph ensures that for each node A a set $V \in \mathcal{C}$ exists such that $fm(A) \subseteq V$. Hence the construction in section 3.2.2 can be used and the joint system belief for T will be proportional to the joint probability function for the BN.

For more information about this junction tree based inference algorithm, we refer to [67, 63, 25, 64].

3.2.5 Remarks

It is shown in [68] that exact evidence propagation in an arbitrary belief network is NP-hard. Even approximate inference (using *Monte Carlo simulation*) is also NP-hard if treated in general [69]. For many applications, however, the networks are small enough (or can be simplified sufficiently) so that these complexity results are not fatal. For applications where the usual inference methods are impractical, we usually develop techniques customer-tailored to particular network topologies, or particular inference queries. So specifying efficiently and accurately the structure as well as CPT for belief networks entails both keen engineering insights of the problem domain and the indispensable good sense of simplification to obtain the appropriate trading-off. It is still somewhat an art.

3.3 Why Belief Networks?

3.3.1 Efficient Uncertainty Management

The attempts to model humans inferential reasoning, namely the mechanism by which people integrate data from multiple sources and come up with a coherent interpretation of the data, have motivated the popular approach that involves constructing an "intelligent agent" that functions as a narrowly focused expert for a particular problem domain. While the past decades have seen some important contributions of expert systems in medical diagnosis and engineering applications, problematic expert system design issues still remain. Dealing with uncertainty is among the most important since uncertainty is the rule, not the exception, in most practical applications. This is based on two observations:

- The concrete knowledge, or the observed evidence from which reasoning will begin, is not accurate.

- The abstract knowledge, namely the knowledge stored in the expert systems as the model of human reasoning, is probabilistic rather than deterministic.

Therefore a natural starting point would be to cast the reasoning process in the framework of probability theory and statistics. However, cautions must be taken if this casting is interpreted in a textbook view of probability theory [70]. For example, if we assume that human knowledge is represented by a JPD, $p(x_1, ..., x_n)$, on a set of random variables (propositions), $x_1, ..., x_n$, then the task of reasoning given evidence $e_1, ..., e_k$ is nothing but computing the probability of a small set of hypotheses $p(H_1, ..., H_m|e_1, ..., e_k)$—the **belief** of the hypotheses given the set of evidence. So one may conclude that given JPD, such kind of computing is merely arithmetic labor.

However, this view turns out to be a rather distorted picture of human reasoning and computing queries in this way is cumbersome at least, if not intractable at all. For example, if we are to encode explicitly for binary variables $x_1, ..., x_n$ an arbitrary JPD $p(x_1, ..., x_n)$ on a computer, we will have to build up a table with 2^n entries. Even if there is some economical way to compact this table, there still remains the problem of manipulating it to obtain queries on propositions of interest.

Human reasoning, on the contrary, acts differently in that probabilistic inference on a small set of propositions is executed swiftly and reliably while judging the likelihood of the conjunction of a large number of propositions turns out to be difficult. This suggests that the elementary building blocks of human knowledge are not the entries of a JPD table but, rather, the low-order marginal probabilities and conditional probabilities defined "locally" over some small set of propositions. It is further observed that an expert will feel more at ease to identify the dependence relationship between propositions than to give the numerical estimate of the conditional probability. This suggests that the dependence structure is more essential to human reasoning than the actual value. Noting also that the nature of dependence relationships between propositions resemble in many aspects that of connectivity in graphs, we can naturally represent such kind of relationship via more explicit graph approaches, which leads to belief networks.

As mentioned above, for any node in the DAG, given its parents, that node is conditionally independent of any other node that is not its descendent. Suppose each node in a belief network model $\mathcal{B}$ has not more than k parents, then the total number of table entries needed to represent the JPD is only $n2^k$, rather than 2^n, where n is the total number of variables in $\mathcal{B}$. When the belief network model is fairly sparse with small k, this is a huge saving.

3.3.2 General Expressiveness

A belief network model is very general and powerful in terms of expressive capability. Many of the current modeling techniques used for fault diagnosis purposes in various application domains can be easily adapted to belief network models. This is no wonder: people usually need to model the causal relationships among propositions to help infer and diagnose the problems at hand, and such causal relationships are typically represented using a directed graph. A belief network model contains such a directed graph, and the conditional probability tables make it more general.

In electric circuit fault diagnosis, for example, a component is comprised of many logic gates such as AND, OR, XOR, etc. These simple logics can be easily modeled using a belief network model for diagnosis purposes: each input and output is represented by a node (variable); the output variables depend on the input variables, and thus the input variables are parents for them in the belief network; the CPT associated with each output variable is simply the truth table as indicated by the logic. The cascade/combination of such simple logic gates can be similarly modeled this way, see for example [71].

Now let us look at one network element in a communication network environment. Such a network element could be a router, switch, hub, workstation, or a satellite terminal. Such an equipment is typically comprised of many hardware and/or software components that relate to each other in some way. The right functioning of this equipment depends on the correct cooperation of the related components. One component failure may incur a chain effect and lead the equipment to malfunction. These relationships among the components can also be naturally modeled using a belief network model.

We can further treat the network elements, linked via communication links[2], as comprising components and we go to a higher abstraction level—a networking scenario. As mentioned before, the task of fault management is to detect, diagnosis and correct the possible faults during network operations, and our focus is on fault diagnosis issues. Fault diagnosis entails modeling of the events[3] and possible causes. One important class of causal analysis is to study the dependent relationship among the propositions and thus the so-called *dependency graph* is utilized for the fault diagnosis [72][5]. A dependency graph models one aspect

[2]We do not usually look inside a communication link for causal analysis.

[3]We assume here that such events are condensed ones after the correlation procedure as defined in section 4.3.

of the managed system: Its nodes (objects) reflect the managed objects (MOs) of the system. Its edges reflect the functional dependencies between MOs. “An object A depends on an object B” means that a failure in B *can* cause a failure in A. There are no attributes or methods or other relations than dependency. A dependency graph can also be easily adapted to a belief network model: nodes in a dependency graph correspond to nodes in a belief network, links in a dependency graph correspond to those in a belief network with the semantics that a child is dependent on its parents. The belief network model is more general in that it can capture the probabilistic relationships easily, and the manipulation of the belief network model is efficient. Another graph-based model for causal analysis is the so-called *causal graph* presented by Kliger et.al. in a widely cited paper [17]. Such a model can also be converted to a belief network model in a similar way.

Dreo presented in [2] some methodologies for the correlation of trouble tickets and she studied the fault diagnosis problem from the service-oriented point of view. A service, used by an end user, is described with a set of subservices which themselves may be represented with a set of subservices. A subservice in layer N provides its functionality to the service on layer $N-1$, and uses some subservices on layer $N+1$. Relations between the services and subservices are represented in a *service graph* . Vertices in the service graph represent services and subservices, while the directed edges represent the relation that a service “uses” the contained subservices for the provision of its functionality. Due to the recursive nature of the service description, and the existing service hierarchies, the service graph has to be layered. The root vertices of the service graph represent services as used by end users, and are placed on the refinement layer 0. When a user senses some service problem, either no service or service quality degradation, he/she then sends out a trouble report to the service provider. After some preliminary correlation[4], the fault diagnosis procedure essentially starts from the reported service and inspects the subsequent subservices on some particular branches as indicated in the service graph, until a leaf vertex is reached. Again, a belief network model can be easily obtained from a service graph: vertices keep the same; the directed edges in a service graph is adapted by reversing the arrow direction; and CPTs are provided for each vertex to represent quantitatively the dependent relation. The CPTs make the belief network model more general in terms of expressiveness. Moreover, the service graph model assumes that the entity represented by each vertex can be tested in a straightforward way, which limits its applicability on many other situations where there exist some unobservable issues needed to be modeled also. As we will see in chapter 6, our belief network model can easily handle such a problem.

In Telecommunications Management Network (TMN) specifications [73], the management functionality in general can be divided into four logical management layers: Network Element Management Layer, Network Management Layer, Service Management Layer, and Business Management Layer. These four layers apply to fault management and we can see from the above discussion that belief network models can cover the modeling needs for the fault diagnosis purposes of all the first three layers. The belief network models are general and very expressive, and since the inception in the mid 1980s, they have become the standard schemes for knowledge representation under uncertainty in many fault diagnosis

[4]Such event correlation basically classify the trouble reports into several groups according to their possible common subservices.

applications [74, 24, 75, 14, 26, 76, 71, 77].

3.4 Conclusions

In summary, we can say that a belief network is a good framework to integrate experts' prior knowledge and statistical data and it constitutes a model of the *environment* rather than, as in many other knowledge representation schemes, a model of the reasoning process. Actually, it simulates the mechanism that operate in the environment and thus allows for various kinds of inferences. The contributions of belief networks can be summarized as follows [78]:

- Natural and key technology for diagnosis
- Foundation for better, more coherent expert systems
- Supporting new approaches to planning and action modeling
    - planning using Markov decision processes
    - new framework for reinforcement learning
- New techniques for learning models from data. See also chapter 5 and [79].

In our work, we assume that our belief network contain only discrete variables, and we will see some belief network examples modeling fault diagnosis scenarios in communication networks in the later chapters.

Note that the introduction above is by no means complete or exhaustive. It is supposed to provide background knowledge on what a belief network is, what it can do and how. For more information in this area, we refer to [80, 81, 25, 82, 26].

Chapter 4

A Framework for Intelligent, Distributed Fault and Performance Management

In this chapter, we first describe the system architecture of our intelligent, distributed fault and performance management system and give the function definitions for each system component. We then discuss some issues to use belief networks as the probabilistic fault models in one of the components.

4.1 System Architecture

Figure 4.1 shows the architecture of our intelligent fault and performance management system. The managed network is divided into several domains [35] and for each domain, there is an intelligent agent attached to it, which is responsible for this domain's fault management. A domain is an abstract notion, for example it might be a subnet, a cluster, a host or a member of a functional partition. For those problems that none of the individual agent can solve, there is a mechanism by which the agents can report to the coordinator and share the information in order to get a wider view and solve it cooperatively. Such a coordinator could be the topmost management entity closely watched by an operator, or it could also be another higher layer DDA that models the problem domain in a more abstract level. So the whole system is, from the agent point of view, a distributed, hierarchical cooperative multi-agent system.

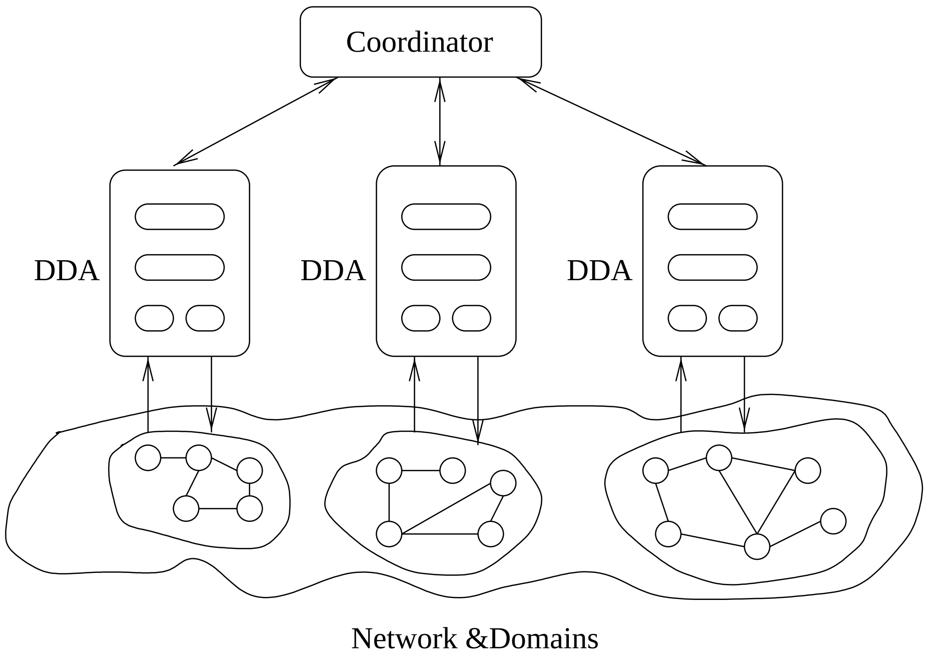

Figure 4.1: Architecture of Integrated, Intelligent Fault Management

Each agent is called a “Domain Diagnostic Agent (DDA)” with the goals of monitoring the health of the domain and diagnosing the faults in a cost-efficient manner. The percepts (inputs) of a DDA are the measured data, alarms or user reports[1] while the action it can take is to report the domain's health, possible causes and suggestive test sequence. The environment it faces is discrete and

[1]The term *user* refers to customer of a network provider, than of service provider to be discussed in chapter 6.

stochastic in nature. A DDA consists of the following components, as shown in Figure 4.2.

- Intelligent Monitoring and Detection Assistant (IMDA)— The role of the IMDA is to monitor and analyze the measured data, alarms and user trouble reports.

- Intelligent Domain Trouble-shooting Assistant (IDTA)—The role of the IDTA is to, based on the symptoms reported by IMDA, find the most possible causes and come up with the suggestive test sequence.

- Intelligent Communication Assistant (ICA)—The role of the ICA is to help the DDA to send messages in cases of global problems.

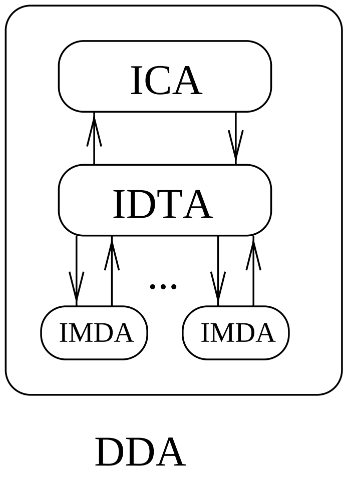

Figure 4.2: Components of a DDA

4.2 Function Definitions

4.2.1 Intelligent Monitoring and Detection Assistant

In a DDA, an IMDA is in the lowest level and serves to interface with Network Element Agents (NEA, as defined by SNMP or CMIP and supposed to provide operation information, for example) and user reports. There may be many such IMDAs delegated and distributed around the network environment, as discussed in chapter 2, and each one provides symptoms[2] information to the IDTA, as described below and illustrated in Figure 4.3. TTS stands for Trouble Ticket System.

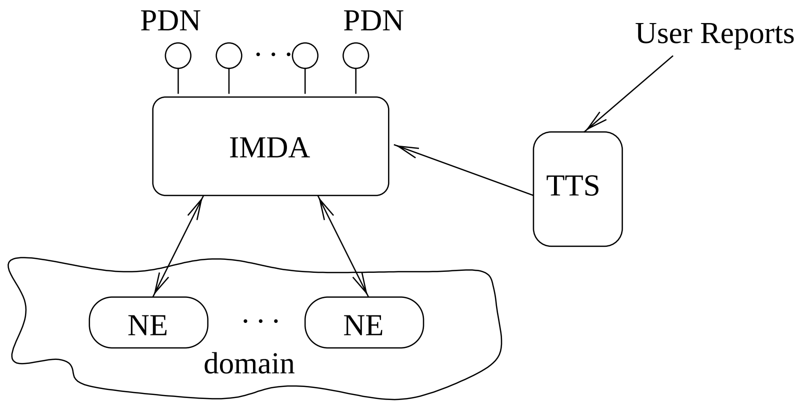

Figure 4.3: Illustration of an IMDA

Input

The input of an IMDA may include measured data from network elements, including IMDA's periodic polling and the alarms sent by the NEA, and the user trouble reports. So we integrate the information from both the network and the

[2]Such symptoms are the condensed events as described in section 4.3.

user point of view.

Output

The output of an IMDA is the activation status of the output nodes, each of which is called a Problem Definition Node (PDN) and acts to represent a certain type of symptoms. Again, such symptoms are the condensed events from the preprocessing procedure as discussed in section 4.3. The PDNs will in turn serve as the input for IDTA. We define five activation levels for each output node in order to reflect the severity of such a symptom type. The five severity levels are "alarm", "major", "minor", "warning" and "normal", respectively.

Functions

The function of an IMDA is to monitor the assigned managed object, analyze the measured data, correlate the alarms sent by the NEA and the user trouble reports, and output the condensed events. An IMDA will periodically poll the network element for operation information and execute on-line data analysis, which may include, but not limited to, threshold checking, ratio computation, change detection, trend analysis. Such analysis will generate alarms which are in turn correlated together with the alarms generated by the managed network element itself and user reports to generate the condensed events. In the analysis procedure of an IMDA, there must be some representation of *normal* behavior for the monitored variables in order to decide whether to issue an alarm or not. Such representations are usually called system (behavior) models and they can be set up in various ways, such as Auto Regressive (AR) modeling, Finite State Machines (FSM), or neural networks.

4.2.2 Intelligent Domain Trouble-shooting Assistant

The IDTA is located above the delegated IMDAs and acts as the trouble-shooter for the symptoms reported from the IMDAs. It includes a probabilistic expert system, which is basically a belief network database. Based on the activation status of the PDNs, a **sub-belief network** is extracted from the database and then the inference and trouble-shooting begin, as described below and shown in Figure 4.4.

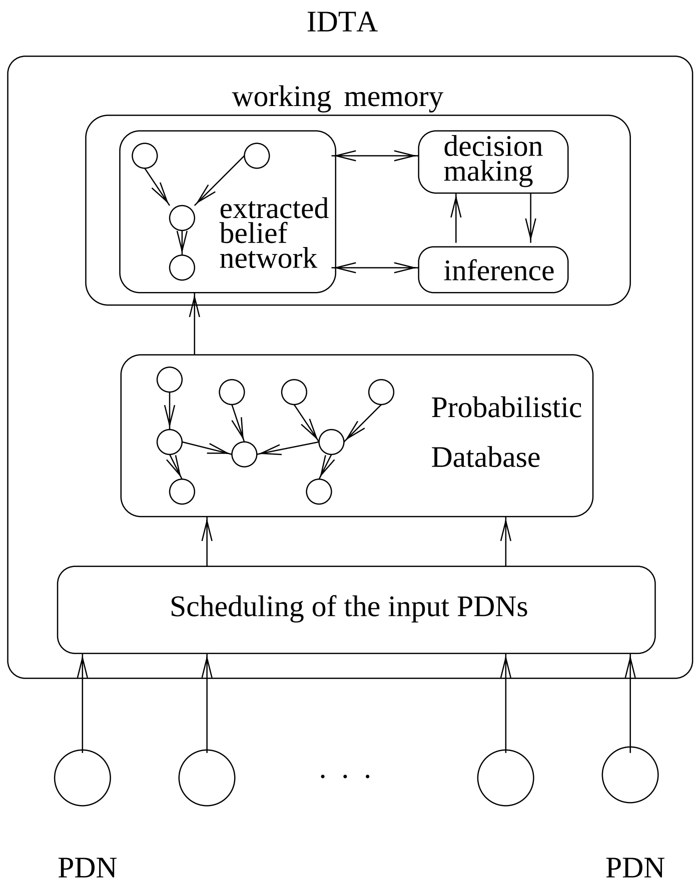

Figure 4.4: Illustration of an IDTA

Input

The input to an IDTA is the activation status of the PDNs as reported from the delegated IMDAs. Such inputs are the results of the event correlation process embedded in the IMDAs that condense the original alarms/events and user reports.

Output

The output of an IDTA consists of primary causes and the suggested test sequence. We tackles the fault diagnosis problems in chapter 6.

Functions

The IDTA functions include scheduling of the PDNs, extraction of the sub-belief network (model construction), inference and trouble-shooting.

- Scheduling of the PDNs: At the same time, there might be more than one PDNs that are not in the "normal" state. As described before, there are five severity levels for each PDN's value. The alarms are to be considered with highest priority and the warnings the least (the "normal" status incurs no diagnosis at all). So there should be a mechanism to discriminate the severity levels and determine the PDNs for which the sub-belief network will be extracted. For example, in a case where PDN one is in alarm status and PDN two is in minor status, it might be more desirable to take care of PDN one only instead of considering both of them (let alone PDN two only). The scheduling algorithm will be studied elsewhere. Note that this should be a quick and easy one since the purpose of IDFM is not scheduling anyway.

- Belief network extraction (model construction): For the selected PDNs, a sub-belief network can be extracted into the working memory. This can be done using the idea of d-separation as introduced in chapter 3. One such example can be found in [83].

- Inference and trouble-shooting: Given the extracted belief network (constructed model) B, the beliefs of any non-PDN nodes to be faulty can be calculated through backward inference based on which static or dynamic trouble-shooting strategies can be adopted to generate the test sequence. See Chapter 6 for details.

Re-action and Pro-action

Re-actions are embodied in the handling of the alarms. For pro-actions, however, we have two implications. First, since the "abnormal" PDNs with status other than "alarm" can also be dealt with, the diagnosis afterwards is actually **proactive** in the sense that it is dealing with something before it really goes wrong. Second, the belief network nodes are not restricted to be physical entities, they can also be "logical" or performance nodes, such as "link congestion", so that "soft" faults can also be included.

The rest of this dissertation is focused on fault diagnosis issues using belief networks.

4.2.3 Intelligent Communication Assistant

When the problems cannot be solved by any of the individual DDAs, it is the role of the ICA to report the problems to an upper layer, where correlation and coordination can be done and a conclusion can be drawn from a global point of view. ICA is an optional component and in many cases we could use IDTA directly to report to an upper entity. The ICA is illustrated in Figure 4.5.

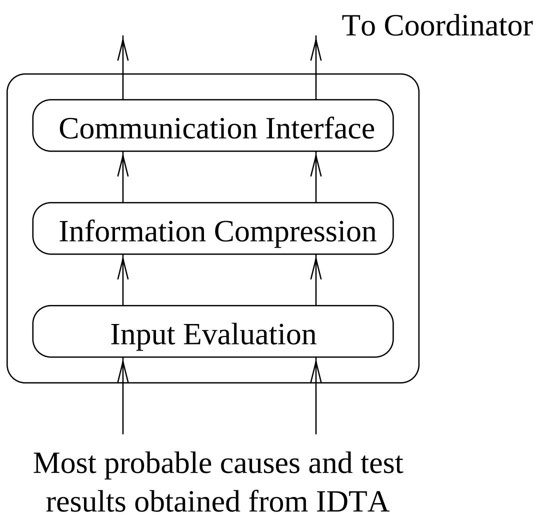

Figure 4.5: Illustration of an ICA

Input

Results of belief computations (the most probable causes) for various extracted belief networks and results from test sequences.

Output

Compressed versions of symptom statistics and of the results given as inputs. The output is then transmitted to a coordinator in the upper layer via some communication links.

Functions

The ICA functions include assessment of the value of the results from belief computations and test sequences (Input Evaluation). This evaluation will decide to what extent it is worthwhile to transmit these results to an upper layer. Only the most relevant results will be transmitted. In addition, the ICA will have a function to select features and compress the data describing valuable inputs (Information Compression). The evaluation and compression will help reduce the amount of data to be transmitted and thus reduce the bandwidth overhead for such communications. Finally, the ICA must include a function which will decide where to send the compressed descriptions and how to communicate with minimum overhead with the upper layer (Communication Interface). To understand such selected and compressed information (encoded data), the coordinator receiving such information must share with ICA the same encoding-decoding protocol.

We do not address ICAs in this dissertation.

4.3 Event Correlations in IMDA before using Belief Networks

In the context of network management, a fault is defined as a cause for malfunctioning. Faults are responsible for making it difficult or preventing the normal functioning of a system and they manifest themselves through errors, that is, deviations in relation to the normal operation of the system. An alarm consists of a notification of the occurrence of a specific event, which may or may not represent an error. In the sequel, we use event and alarm interchangeably. As mentioned in chapter 1, a single fault can generate a lot of alarms in a variety of domains, and multiple faults will make things even worse. Several factors contribute to this situation [4]:

- A device may generate several alarms due to a single fault.
- A fault may be intrinsically intermittent, which implies in the sending of a notification at each new occurrence.
- The fault of a component may result in the sending of an alarm notification each time the service supplied by this component if invoked.
- A single fault may be detected by multiple network components, each one of them emitting an alarm notification.
- The fault of a given component may affect several other components, causing the fault's propagation.

Not all these alarms are useful and we can not afford to input all these alarms to our belief network models for fault diagnosis. Therefore it is highly desirable to utilize some preprocessing on these original alarms before we start the real diagnosis procedure. We call this preprocessing event correlation, which is a *conceptual interpretation* of multiple alarms such that new meanings are assigned to these alarms [16]. The output events from such correlation processing are called *condensed* events. Such correlation procedures are executed in the delegated Intelligent Monitoring and Detection Assistants (IMDA), as described in section 4.1, and the condensed events are just the output status of the Problem Definition Nodes (PDN) described there. Several types of correlation may be identified [84], according to the operations executed on the alarms. The most important of these operations are detailed as follows.

Compression

Event compression is the task of reducing multiple occurrence of identical events into a single representative of the events. No number of occurrences of the event is taken into account. The meaning of the compression correlation is almost identical to the single event a, except that additional contextual information is assigned to the event to indicate that this event happened more than once.

Filtering

Event (alarm) filtering is the most widely used operation to reduce the number of alarms presented to the operator. If some parameter $p(a)$ of alarm a, e.g., priority, type, location of the network element, time stamp, etc., does not fall into the set of predefined legitimate values H, then alarm a is simply discarded or sent into a log file. The decision to filter alarm a out or not is based solely on the specific characteristics of alarm a. In more sophisticated cases, set H could be dynamic and depend on user-specific criterion or criterion calculated by the system. For an example, see the adaptive threshold in [15].

Suppression

Event suppression is a context-sensitive process in which event a is temporarily inhibited depending on the dynamic operational context C of the network management process. The context C is determined by the presence of other event(s), network management resources, management priorities, or other external requirements. The change in the operational context could lead to exhibition of the suppressed event. Temporary suppression of multiple events and control of the order of their exhibition is a basis for dynamic focus monitoring of the network management process.

Counting

Counting consists of generating a new alarm each time the number of occurrences of a given type of event surpass a previously established threshold.

Generalization

Event generalization is a correlation in which event a is replaced by its super class b. Event generalization has a potentially high utility for network management. It allows one to deviate from a low-level perspective of network events and view situations from a higher level. As an example, in the simultaneous occurrence of the alarm corresponding to all the routes that utilize a certain cable as a physical media, each one of the original alarms may be replaced by an alarm indicating a defect in the cable; next, through a compression operation, all the repeated alarms may be replaced by a single alarm.

Specialization

Event specification is an opposite procedure to event generalization. It substitutes an event with a more specific subclass of this event. This operation does not add any new information besides the ones that were already implicitly present in the original alarms and in the configuration database, but it is useful in making evident the consequences that an event in a given management layer may cause in the higher management layer. As an example of possible specification, the correlation system may generate, whenever a determined path is interrupted, an alarm for each one of the services affected by the interruption. Thus, through specification, the consequences of the telecommunications services management layer will be made evident.

Temporal Relation

Temporal relations T between events a and b allow them to be correlated depending on the order and time of their arrival. Several temporal relationships may be identified, utilizing concepts like AFTER, FOLLOW, BEFORE, PRECEDE, DURING, START, FINISH, COINCIDE, OVERLAP .

Clustering

Event clustering allows the creation of complex correlation patterns using logical operators $\wedge$ (and), $\vee$ (or), and $\neg$ (not) over component terms. The terms in the pattern could be primary network events, previously defined correlations, or tests of network connectivity. Clustering consists of generating a new alarm based on the detection of an established complex correlation patterns on the received alarms. The clustering operation may also take into account the result of other correlations and the result of tests carried out in the network.

In summary, event correlation supports the following network management tasks [16, 84]:

- Reduction of the information load presented to the network operations staff by dynamic focus monitoring and context-sensitive event suppression (filtering).

- Increasing the semantic content of information presented to the network operations staff by aggregation and generalization of events.

- Real-time network fault isolation, causal fault diagnosis, and suggestion of corrective actions.

- Analysis of the ramification of events, prediction of network behavior, and trend analysis.

- Long-term correlation of historic event log files and network behavior trend analysis.

4.4 Construction of a Belief Network Model

With the condensed events at hand, we would like to build a belief network model for the diagnosis purpose. This entails setting up both the DAG structure and the associated CPTs.

First, we identify the set of random variables that will be used to represent in the belief network model the causes and symptoms of the current managed problem domain. The condensed events are the manifestations or symptoms of the hidden faults. From the causal semantics of the directed links of a belief network model, there should be directed paths from the vertices corresponding to the faults to those corresponding to the condensed events. For a node representing a condensed event, if its parent is another condensed event, we call this child event a derived event and we just delete this node from the model. After this pruning, all of our event nodes are leaf nodes. For a fault node, if its parents are some other fault nodes, we call it a derived fault node. We call a fault node without any parent[3] an ultimate fault node. In our work, we only consider the ultimate fault nodes. Besides root nodes and leaf nodes, we also

[3]A root node in graph theory terminology

have some intermediate nodes that help mediating the modeling, see chapter 6 for more information.

The granularity of the belief network is decided by the system designer, and depends also on the requirement on how precise the fault diagnosis should be. A higher granularity is requested if it is necessary to localize the fault more precisely.

The parameters in the CPTs associated with each node can be initially assigned by human experts. Since human will usually feel less at ease to give the numerical estimate of the conditional probabilities, we can provide a table mapping from frequency-describing words to numerical values for the human experts to choose from. One example of such a table is shown in Table 4.1. Such initial setups can be further validated and improved using statistical data, see chapter 5 for details.

Table 4.1: From frequency-describing words to numerical values

Always	1
Almost Always	0.9
Quite Often	0.8
Often	0.7
Likely	0.6
Half_and_half	0.5
Less Likely	0.4
Sometimes	0.3
Occasionally	0.2
Seldom	0.1
Never	0

In summary, the construction of a belief network model consists of the following steps [61]:

- Choose the set of random variables that describe the domain
- Give order numbers to the random variables using topological ordering
- While there are still variables left:
  - Pick a random variable and add a node representing it
  - Choose parents for it as the minimal set of nodes already in the network such that (3.3) is satisfied
  - Specify the CPT for it

Since each node is only connected to earlier nodes, this construction method guarantees that the network is acyclic. Furthermore, it is both complete and consistent [70].

4.5 Conclusions

The framework proposed in this chapter is quite general and can be incorporated into many network management paradigms, like the traditional client-server (CS) based architecture, and the more recent mobile-code based framework.

Integration with a Client-Server based Management Platform

A conventional network management system consists of two classes of components: managers and agents. Applications in the management station assume the manager role; Agents are server processes running in each involved manageable network entity. These agents collect network device data, store them in some format, and support a management protocol, e.g., SNMP. Manager applications retrieve data from element agents by sending corresponding requests over the management protocol. Most of the current commercial network and system management platforms take this paradigm.

In this setup, our components all reside above the management platform as management applications. The IMDAs are not delegated agents to the network elements; they are simply data handling processes. The IDTAs serve to find the right diagnosis and the notion of domain is still valid. We do not usually need to use ICAs in such situations. Figure 4.6 illustrates such ideas. Note that we also integrate the trouble ticket system here. From architectural point of view, a TTS is a management application upon a management platform. The gateway is an application between platform and a TTS that collects alarm information issued from the platform and generate trouble tickets. It should be noted that an end user has no access to the management platform. He/she can only submit trouble reports, and be informed about the progress of fault recovery.

Integration with a Mobile Code based Management Framework

In chapter 2, we discuss three mobile code based design paradigms, i.e., remote evaluation, code on demand and mobile agent, in the development of distributed applications. In particular, for network management purposes, we combine the remote evaluation and code on demand paradigms and propose the adaptive distributed network monitoring and control framework for supporting fault and

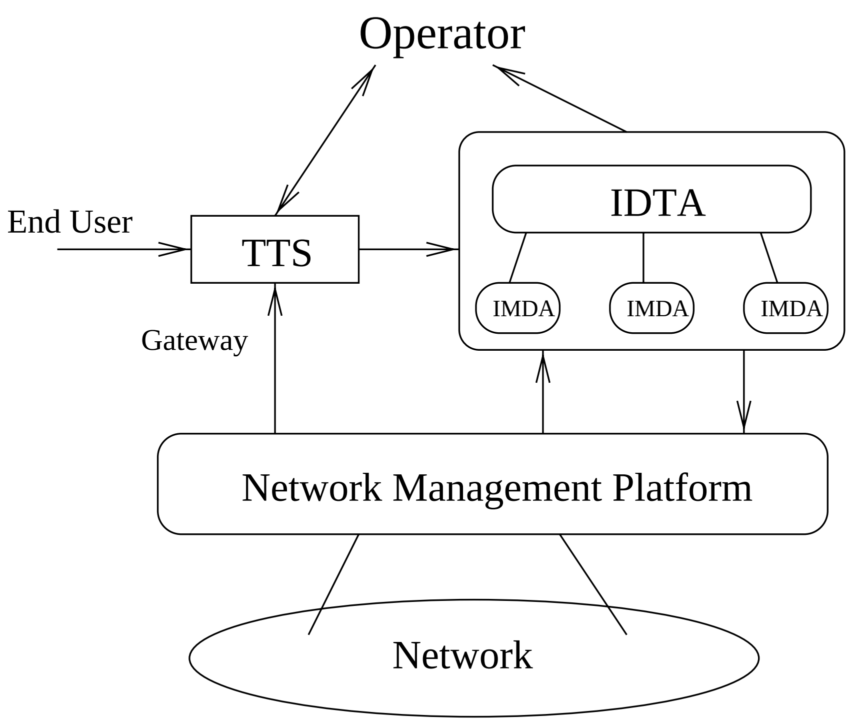

Figure 4.6: Integration with a traditional management platform

performance management. In this setup, we distribute some monitoring intelligence via our delegated agents to the network elements for some local computation. The computed results are reported back to the manager-coordinator for higher level processing.

Our components discussed in this chapter fit naturally into such a framework. In fact, the IMDAs are just the delegated agents in this framework, and IDTAs usually reside on the manager-coordinator site.

The rest of this dissertation focuses on the functionality of an IDTA.

Chapter 5

Statistical Parameter Learning for Belief Networks based on Expectation Maximization Method

5.1 Introduction

In previous discussions we assumed that both the network structure and the associated CPTs can be provided by human experts as the prior information. In many applications, however, such information is not available. In addition, different experts may treat the systems in various ways and thus give different and sometimes conflicting assessments. In such cases, the network structure and corresponding CPTs can be estimated using empirical data and we refer to this process as *learning* [85, 79, 86]. Even if such prior information does exist, it is still desirable to validate and improve the model using data.

In this chapter, we address the problem of parameter learning under fixed structure. Both complete and incomplete (data) observations are included. Given complete data, we describe the simple problem of single parameter learning for intuition and then expand to belief networks under appropriate system decomposition. If the observations are incomplete, we first estimate the missing observations and treat them as though they are "real" observations, based on which the parameter learning can be executed as in complete data case. We derive a uniform algorithm based on this idea for incomplete data case and present the convergence and optimality properties.

Before we proceed to the learning problems, we give the following definitions which are used throughout this chapter. First, we repeat the definition of belief network for convenient reference.

Definition 5.1.1 *A belief network is a Directed Acyclic Graph (DAG) $G = (X, E, P)$ in which: The nodes X represent variables of interest (propositions); The set of directed links or arrows E represent the causal influence among the variables; The strength of an influence is represented by conditional probabilities attached to each cluster of parent-child nodes in the network.*

We let $X = [X_1, \ldots, X_n]$, where each random variable X_i takes values from finite alphabet A_i. $X_i = x_i$ means random variable X_i assumes the value x_i. If we know that X_i assumes its j^{th} value from A_i, we write $X_i = x_i^j$.

Definition 5.1.2 *An observation is an instantiation $x = [x_1, \ldots, x_n]$ of X, with $x \in R^n$ and assume values in $A \overset{\triangle}{=} A_1 \times, \ldots, \times A_n$.*

Definition 5.1.3 *If every node in X is observed, we call x a complete observation, or a complete data set. Each such an x is called a configuration.*

Definition 5.1.4 *If there are some random variables in X that are not instantiated, such an observation is called an incomplete data.*

Definition 5.1.5 *The instantiated set of nodes $S \subseteq X$ forms the evidence set, while $N = X \backslash S$ is called the non-evidential set.*

In cases of full observation, $S = X$; but in practice, mostly $S \subset X$, which means incomplete data.

Definition 5.1.6 *If we mark the nodes in X that belongs to S with $*$ and those in N with $?$, then we form the observation schema S^+. Each instantiation of the schema is called an evidence under schema S^+.*

For example, in a 5 node belief network which takes only binary values, such a schema is $S = (*, ?, ?, *, *)$ and one possible instantiation is $(0, ?, ?, 1, 0)^T$.

Definition 5.1.7 *Suppose we have a batch of observations $D = [D_1, \ldots, D_L]$ with each $D_i \in R^n$ complying with schema S_i^+. If $S_i^+ \equiv S_j^+$, $\forall i, j = 1, \ldots, L$, we say D is a uniform batch of observations. If, on the other hand, S_i^+ may or may not be the same as S_j^+ for $i \neq j$, we say D is a hybrid-schema observation set.*

5.2 Parameter Learning under Complete Data

In this section, we begin with the simple one-parameter learning case, then we move to multi-variable parameter learning in belief networks.

5.2.1 Simple Parameter Learning

Imagine we have a (not necessarily fair) coin, and we conduct experiments whereby we flip the coin in the air, and it comes to land as either head or tail. We assume that different tosses are independent and that, in each toss, the probability of the coin landing heads is some underlying unknown real number θ. Our goal here is to estimate θ based on the outcomes of the experiments.

Define the *likelihood* function as $P_\theta(D) = \theta^h(1-\theta)^t$, which is the probability with which we get a particular data set D with h heads and t tails given that the probability θ has a certain value. It is straightforward to verify that the value of θ which maximizes $P_\theta(D)$ is $\frac{h}{h+t}$. This is called *maximum likelihood (ML)* estimate for θ.

ML estimate is the expectation of Bayesian estimate and ML estimate approaches to Bayesian estimate when sample size becomes unboundedly large (which means prior belief becomes less and less important as the data accumulate). Given ML estimate, it is straightforward to obtain the hyper-parameters for Dirichlet distribution in Bayesian analysis.

The usefulness of the coin flipping here lies in the fact that: If, by some decomposition mechanism, we can break a complex problem into multiple independent single parameter learning problems, then we can use the flip coin techniques here for each of them; And, such computations would be possibly done in a distributed manner.

5.2.2 Parameter Learning for a Belief Network

System Decomposition

Suppose data $D = [D_1, \ldots, D_L]$ are generated independently from some underlying distribution, with each $D_i = [x_1[i], \ldots, x_n[i]]^T$. The problem here is to find the CPT parameters θ that best model the data. The parameter θ is actually a 3-dimensional matrix, with its element θ_{ijk} defined as the probability that variable X_i takes on its j^{th} possible value assignment given its parents Π_i takes on their k^{th} possible value assignment, or

$$\theta_{ijk} = P(X_i = x_i^j | \Pi_i = \pi_i^k). \tag{5.1}$$

We assume here that $\theta_{ijk} > 0$, $\forall\, i, j, k$, and $P_\theta(D)$ continuous with θ. If we define $L(\theta; D) = P_\theta(D)$ as the likelihood function given some parameter θ, the Maximum Log-Likelihood formulation for the problem is:

$$\begin{cases} \quad \max_\theta \log L(\theta; D) \\ s.t. \;\; \sum_j \theta_{ijk} = 1, and \\ \qquad \theta_{ijk} \in [0, 1] \end{cases} \tag{5.2}$$

Moreover, we have:

$$\begin{aligned} L(\theta; D) &= \prod_{l=1}^{L} P_\theta(x_1[l], \ldots, x_n[l]) \\ &= \prod_{l=1}^{L} \prod_{i=1}^{n} P_{\theta_i}(x_i[l] | \Pi_i[l]) \\ &= \prod_{i=1}^{n} \prod_{l=1}^{L} P_{\theta_i}(x_i[l] | \Pi_i[l]) \\ &= \prod_{i=1}^{n} L_i(\theta_i; D), \end{aligned} \tag{5.3}$$

where $L_i(\theta_i; D) \triangleq \prod_{l=1}^{L} P_{\theta_i}(x_i[l] | \Pi_i[l])$. The first equality comes from the fact that D consists of independent observations and the second equality follows (3.2). $\theta_{\mathbf{i}}$ is a 2-dimensional matrix, with rows occupied by its possible values and columns defined according to its parents' status. From (5.3) we get

$$\log L(\theta; D) = \sum_{i=1}^{n} \log L_i(\theta_i; D). \tag{5.4}$$

The above derivation gives us a decomposition of the belief networks learning problem based on independent observations. Namely, the maximum likelihood solutions of (5.2) are just those achieved by the sum of the solutions from the following independent estimation problems:

For each $X_i \in X$:

$$\begin{cases} \max_{\theta_i} \log L_i(\theta_i; D) \\ s.t.\ \sum_j \theta_{ijk} = 1, and \\ \qquad \theta_{ijk} \in [0,1] \end{cases} \tag{5.5}$$

For each $L_i(\theta_i; D)$, we make further decomposition as follows.

Definition 5.2.1 *The set of nodes $S_i = (X_i, \Pi_i)$ forms the extracted schema for node X_i. X_i assumes values from A_i. Π_i takes values from $\prod_{X_j \in \Pi_i} A_j$. Let $K = |\prod_{X_j \in \Pi_i} A_j|$.*

Definition 5.2.2 *Group G_{ik} is the set of instantiations of S_i in D with $\Pi_i = \pi_i^k$. The number of elements in G_{ik} is $N_{ik} = |G_{ik}| = \sum_{l=1}^{L} I_{\{\pi_i^k | D_l\}}$, where $I_{\{\pi_i^k | D_l\}}$ is the indicator function defined as:*

$$I_{\{z|D_l\}} = \begin{cases} 1 & \textit{if } z \textit{ occurs in } D_l \\ 0 & \textit{otherwise} \end{cases} \tag{5.6}$$

and $\sum_k N_{ik} = L$.

Definition 5.2.3 *In G_{ik}, G_{ijk} is the set of instantiations where $X_i = x_i^j$, and $N_{ijk} = |G_{ijk}| = \sum_{l=1}^{L} I_{\{x_i^j, \pi_i^k | D_l\}}$, number of elements in G_{ijk} with $\sum_{j=1}^{A_i} N_{ijk} = N_{ik}$.*

So the decomposition is:

$$L_i(\theta_i; D) \quad = \quad \prod_{l=1}^{L} P_{\theta_i}(x_i[l] | \Pi_i[l])$$

$$
\begin{aligned}
&= \prod_{k=1,\{\Pi_i=\pi_i^k\}}^{K} \prod_{j=1,\{X_i=x_i^j|\Pi_i=\pi_i^k\}}^{|A_i|} \theta_{ijk}^{N_{ijk}} \\
&= \prod_{k=1,\{\Pi_i=\pi_i^k\}}^{K} L_i^k(\theta_i; D), \qquad (5.7)
\end{aligned}
$$

or

$$
\log L_i(\theta_i; D) = \sum_{k=1,\{\Pi_i=\pi_i^k\}}^{K} \log L_i^k(\theta_i; D), \qquad (5.8)
$$

where $L_i^k(\theta_i; D) = \prod_{j=1,\{X_i=x_i^j|\Pi_i=\pi_i^k\}}^{|A_i|} \theta_{ijk}^{N_{ijk}}$ is just the likelihood function of multinomial distribution under parameter θ.

Then the ML solutions of (5.5) are again those achieved by the sum of the solutions from the following independent estimation problems:

For each $\Pi_i = \pi_i^k$,

$$
\begin{cases}
\max_{\theta_i} \log L_i^k(\theta_i; D) \\
s.t. \ \sum_j \theta_{ijk} = 1, and \\
\theta_{ijk} \in [0, 1]
\end{cases} \qquad (5.9)
$$

Or equivalently, by defining $r_j = \theta_{ijk}$, $y_j = N_{ijk}$, and $\tilde{L}(r; D) = \prod_{j=1}^{|A_i|} r_j^{y_j}$, we consider the maximum log-likelihood problem,

For each $\Pi_i = \pi_i^k$,

$$
\begin{cases}
\max_r \log \tilde{L}(r; D) \\
s.t. \ \sum_j r_j = 1, and \\
r_j \in [0, 1]
\end{cases} \qquad (5.10)
$$

which is just a generalization of the simple parameter learning problem discussed in section 5.2.1.

The decomposition $\log L(\theta; D) = \sum_{i=1}^{n} \sum_{k=1,\{\Pi_i=\pi_i^k\}}^{K} \log L_i^k(\theta_i; D)$ first exploits the conditional independence structure embedded in belief networks and helps to reduce the problem to n independent ML estimate problems; then, each such problem is further decomposed as shown in (5.7). We can thus do the learning in a "local" and distributed manner.

Parameter Estimation

In this section, we derive the ML estimates for problem (5.10) and provide the optimality results. For some concepts of estimation theory, we refer to [87].

Lemma 5.2.4 *Given complete data set D, $\log \tilde{L}(r; D)$ is negative and strictly concave in $r = [r_1, \ldots, r_{|A_i|}]$.*

Proof. Let $J = |A_i|$ and define function $f : R^J \to R^1$ as

$$\begin{aligned} f(r) &= \log \tilde{L}(r; D) \\ &= \sum_{j=1}^{J-1} y_j \log r_j + y_J \log(1 - \sum_{j=1}^{J-1} r_j). \end{aligned} \tag{5.11}$$

Obviously, $\log \tilde{L}(r; D) < 0$.

Let $r = \lambda\alpha + (1-\lambda)\beta$, where α and β satisfy the constraints in (5.10) and $0 < \lambda < 1$. Since $\log(x)$ is strictly concave in x, we have

$$\begin{aligned} f(\lambda\alpha + (1-\lambda)\beta) &> \lambda \sum_{j=1}^{J-1} y_j \log \alpha_j + (1-\lambda) \sum_{j=1}^{J-1} y_j \log \beta_j \\ &\quad + \lambda y_J \log(1 - \sum_{j=1}^{J-1} \alpha_j) + (1-\lambda) y_J \log(1 - \sum_{j=1}^{J-1} \beta_j) \\ &= \lambda f(\alpha) + (1-\alpha) f(\beta). \end{aligned} \tag{5.12}$$

$\Rightarrow$ Strictly concave. $\square$

Lemma 5.2.5 *Given complete data set D, $\log L(\theta; D)$ is negative and strictly concave in $\theta = \{\theta_{ijk}\}$.*

Proof.

$$\log L(\theta; D) = \sum_{i=1}^{n} \sum_{k=1,\{\Pi_i=\pi_i^k\}}^{K} \log L_i^k(\theta_i; D),$$

where each $\log L_i^k(\theta_i; D)$ is negative and strictly concave in θ_{ijk}. We conclude that the sum of those negative (and thus non-cancelling), strictly concave functions is also negative and strictly concave in θ_{ijk}, $\forall\, i, j, k$, or, concave in θ. □

Lemma 5.2.6 *The maximum likelihood solution for problem (5.10) is*

$$r_j = y_j / \sum_{j=1}^{J} y_j, \tag{5.13}$$

where $y_j = N_{ijk}$, as defined in definition 5.2.3.

Proof. Use the f notation as above and obtain the Likelihood Equations

$$\frac{\partial f}{\partial r_j} = 0, \;\; j = 1, \ldots, J. \tag{5.14}$$

From (5.11) it is straightforward to get $J-1$ independent equations:

$$\frac{y_j}{r_j} = \frac{y_J}{1 - \sum_{i=1}^{J-1} r_i}, \;\; \forall j = 1, \ldots, J-1, \tag{5.15}$$

from which it is easy to obtain that $r_j = y_j / \sum_{j=1}^{J} y_j$, and we can check that $r_j \in [0, 1]$ and $\sum_j^J r_j = 1$. By lemma 5.2.4, we know such stationary points are global maxima (because $\log \tilde{L}(r; D)$ is strictly concave). □

Lemma 5.2.7 *The ML estimates for problem (5.10) are minimum variance unbiased estimators (MVUE).*

Proof. Given data set D, suppose X_i's J possible values are $[y_1, \ldots, y_J]^T$, if we look at node X_i under $\Pi_i = \pi_i^k$. The multinomial distribution is

$$p_\psi(y) = \frac{(\sum_{i=1}^{J} y_i)!}{\prod_{i=1}^{J} y_i!} \prod_{i=1}^{J} \psi_i^{y_i} \tag{5.16}$$

Then for an underlying set of ψ, we have for estimator $\hat{\psi}_j(y) = y_j / \sum_{i=1}^{J} y_i$

$$\begin{aligned}
E_\psi\{\hat{\psi}_j(y)\} &= \sum_{y_j=0}^{N} \frac{y_j}{N}\frac{N!}{y_j!}\psi_j^{y_j} \sum_{\substack{y_i \\ (\sum_{i\neq j} y_i)=N-y_j}} \frac{1}{\prod_{i\neq j} y_i!}\prod_{i\neq j}\psi_i^{y_i} \qquad (N \triangleq \sum_{i=1}^{J} y_i) \\
&= \frac{1}{N}\sum_{y_j=0}^{N} y_j \binom{N}{y_j}\psi_j^{y_j}(1-\psi_j)^{N-y_j} \\
&= \frac{1}{N}N\psi_j = \psi_j. \qquad (5.17)
\end{aligned}$$

From (5.16), we have

$$\frac{\partial \log p_\psi(y)}{\partial \psi_j} = \frac{y_j}{\psi_j} - \frac{N-y_j}{1-\psi_j}. \qquad (5.18)$$

Then the Fisher information and variance are,

$$\begin{aligned}
\mathcal{I}_\psi &= -E_\psi\{\frac{\partial^2 \log p_\psi(y)}{\partial^2 \psi_j}\} \\
&= E_\psi\{\frac{y_j}{\psi_j^2} + \frac{N-y_j}{(1-\psi_j)^2}\} \\
&= N/\psi_j(1-\psi_j), \qquad (5.19)
\end{aligned}$$

$$\begin{aligned}
Var_\psi\{y_j/N\} &= \frac{1}{N^2}E_\psi\{y_j^2\} - E^2\{y_j/N\} \\
&= \psi_j(1-\psi_j)/N = 1/\mathcal{I}_\psi. \qquad (5.20)
\end{aligned}$$

$\Rightarrow$ MVUE (achieves Cramer Rao Lower Bound, CRLB). $\square$

5.3 Parameter Learning under Incomplete Data using EM Algorithm

In this section, we first brief the idea of EM algorithm, followed by the derivation for belief networks and the discussion of convergence and optimality properties.

5.3.1 A Brief Description of EM Algorithm

EM algorithm is broadly applicable for computing maximum likelihood estimates from incomplete data. Each iteration consists of an expectation step followed by a maximization step, and hence the name [88, 89].

Suppose we have two sample spaces $\mathcal{X}$ and $\mathcal{Y}$ with many-to-one mapping $\mathcal{X} \to \mathcal{Y}$, where $\mathbf{x}$ can not be observed directly, but instead, only through $\mathbf{y}$. Let the family of sampling densities depending on Φ for $\mathcal{X}$ and $\mathcal{Y}$ are $f_\Phi(\mathbf{x})$ and $g_\Phi(\mathbf{y})$, respectively. We call $f_\Phi(\mathbf{x})$ as the complete data specification and $g_\Phi(\mathbf{y})$ as the incomplete data specification, with the following relation:

$$g_\Phi(\mathbf{y}) = \int_{\mathcal{X}(\mathbf{y})} f_\Phi(\mathbf{x}) dx, \tag{5.21}$$

where $\mathcal{X}(\mathbf{y})$ is the set of $\mathbf{x} \in \mathcal{X}$ that corresponds to $\mathbf{y}$. EM algorithm aims to find a value of Φ that maximizes $g_\Phi(\mathbf{y})$ given an observed $\mathbf{y}$, but does so by making essential use of $f_\Phi(\mathbf{x})$. When $f_\Phi(\mathbf{x})$ belongs to an exponential family, we have the EM algorithm:

Let $\Phi^{(p)}$ denotes the current value of Φ after p cycles, then for the next cycle:

E-step: Estimate the complete-data sufficient statistics $t(\mathbf{x})$ by

$$t^{(p)} = E_{\Phi^{(p)}}\{t(\mathbf{x})|\mathbf{y}\} \tag{5.22}$$

M-step: Determine $\Phi^{(p+1)}$ as the solution of the equations

$$E_\Phi\{t(\mathbf{x})\} = t^{(p)} \tag{5.23}$$

5.3.2 EM Algorithm for Belief Networks

Suppose we have a batch of observations $D = [D_1, \ldots, D_L]$ with each $D_i \in R^n$ complying with schema S_i^+. D may or may not be uniform. The objective is to find the most likely underlying parameter θ that can best model the incomplete observations, namely

$$\begin{cases} \max_\theta \log P_\theta(D) \\ s.t. \;\; \sum_j \theta_{ijk} = 1, and \\ \theta_{ijk} \in [0,1] \end{cases} \tag{5.24}$$

and we wish to do this via the maximization of an associated complete data problem. The idea is that we first estimate and "fill" the missing values based on the evidence and current guess of the parameters, after which we treat them as the real data and apply the ML principle to do the parameter learning. The estimation of the missing values are called Expectation-step (or E-step) and the parameter learning step is called Maximization-step (or M-step). It is straightforward to see that a multinomial distribution $p_\psi(y) = \frac{(\sum_{j=1}^J y_j)!}{\prod_{j=1}^J y_j!} \prod_{j=1}^J \psi_j^{y_j}$ belongs to the exponential family and the sufficient statistics is just the set $\{y_j\}$, $j = 1, \ldots, J$.

E-step: For each sample D_l, we want to estimate the values for those nodes corresponding to "?" mark in schema S_l^+ and thus get the augmented data set $C(D_l)$. Let N_l be the set of nodes marked as "?" in schema S_l^+, then given D_l, there are all together $\prod_{m \in N_l} |A_m|$ cases in the augmented data set $C(D_l)$. For any entry $D_l^+(q) \in C(D_l)$, $q = 1, \ldots, \prod_{m \in N_l} |A_m|$, the evidential nodes are "clamped" as in observation D_l, while the non-evidential nodes take the q^{th} combination from the $\prod_{m \in N_l} |A_m|$ choices. We use Q to denote $\prod_{m \in N_l} |A_m|$ in the sequel for convenience. Then under the current guess $\tilde{\theta}$, $P_{\tilde{\theta}}(D_l^+(q))$ can be computed using (3.2), and the probability of $D_l^+(q)$ given D_l is

$$P_{\tilde{\theta}}(D_l^+(q)|D_l) = \frac{P_{\tilde{\theta}}(D_l^+(q))}{\sum_q P_{\tilde{\theta}}(D_l^+(q))}. \tag{5.25}$$

M-step: For augmented data set $C(D_l)$ where each entry is a complete observation, we can either average over those entries or find the most-probable entry to serve as the complete data set for the ML estimator. At this time, we consider the averaging method, by which we weigh each $D_l^+(q)$ according to (5.25) within each D_l.

So the associated problem is: for complete data set $C(D) = [C(D_1), \ldots, C(D_L)]$,

$$\begin{cases} \max_\theta \log \bar{P}_\theta(C(D)) \\ s.t. \ \sum_j \theta_{ijk} = 1, and \\ \quad \theta_{ijk} \in [0,1] \end{cases} \tag{5.26}$$

where

$$\begin{aligned} \log \bar{P}_\theta(C(D)) &= \sum_{l=1}^{L} \log \bar{P}_\theta(C(D_l)) \\ &= \sum_{l=1}^{L}\sum_{q=1}^{Q} P_{\tilde{\theta}}(D_l^+(q)|D_l) \log P_\theta(D_l^+(q)) \\ &= \sum_{i=1}^{n}\sum_{l=1}^{L}\sum_{q=1}^{Q} P_{\tilde{\theta}}(D_l^+(q)|D_l) \log P_{\theta_i}(X_i(D_l^+(q))|\Pi_i(D_l^+(q))) \end{aligned} \tag{5.27}$$

Compare (5.27) with (5.4), we can take similar decompositions and by lemma 5.2.6 we have for node X_i under $\Pi_i = \pi_i^k$,

$$\hat{\theta}_{ijk} = \frac{\tilde{N}_{ijk}}{\tilde{N}_{ik}}, \tag{5.28}$$

where $\tilde{N}_{ijk}$ and $\tilde{N}_{ik}$ can be obtained as in definition 5.2.2 and 5.2.3, except that the complete data $D_l^+(q)$ is weighed by $P_{\tilde{\theta}}(D_l^+(q)|D_l)$. So

$$\hat{\theta}_{ijk} = \frac{\sum_{l=1}^{L}\sum_{q=1}^{Q} I_{\{x_i^j, \pi_i^k | D_l^+(q)\}} P_{\tilde{\theta}}(D_l^+(q)|D_l)}{\sum_{l=1}^{L}\sum_{q=1}^{Q} I_{\{\pi_i^k | D_l^+(q)\}} P_{\tilde{\theta}}(D_l^+(q)|D_l)}, \tag{5.29}$$

where $\tilde{\theta}$ is the current set of parameters and $I_{\{z|D_l^+(q)\}}$ is defined as

$$I_{\{z|D_l^+(q)\}} = \begin{cases} 1 & \text{if } z \text{ occurs in } D_l^+(q) \\ 0 & \text{otherwise} \end{cases} \tag{5.30}$$

One may easily observe that such augmentation is of combinatorial complexity, and even when $A_i = \{0, 1\}, \forall i$, the entries for $C(D_l)$ would be $2^{|N_l|}$, which makes the computation in (5.29) intractable in most cases.

However, notice that

$$P_{\tilde{\theta}}(x_i^j, \pi_i^k | D_l) = \sum_{q=1}^{Q} I_{\{x_i^j, \pi_i^k | D_l^+(q)\}} P_{\tilde{\theta}}(D_l^+(q) | D_l), \tag{5.31}$$

$$P_{\tilde{\theta}}(\pi_i^k | D_l) = \sum_{q=1}^{Q} I_{\{\pi_i^k | D_l^+(q)\}} P_{\tilde{\theta}}(D_l^+(q) | D_l), \tag{5.32}$$

we can simplify (5.29) as

$$\hat{\theta}_{ijk} = \frac{\sum_{l=1}^{L} P_{\tilde{\theta}}(x_i^j, \pi_i^k | D_l)}{\sum_{l=1}^{L} P_{\tilde{\theta}}(\pi_i^k | D_l)}, \tag{5.33}$$

where $P_{\tilde{\theta}}(x_i^j, \pi_i^k | D_l)$ and $P_{\tilde{\theta}}(\pi_i^k | D_l)$ can be calculated using standard inference algorithm given D [80, 70, 26]. So we don't need to do the augmentation explicitly and hence avoid the combinatorial complexity. Compare (5.33) with (5.13), we can see that we just replace the "hard" counting measures y_j, $\forall j \in J$ in (5.13) with the "soft" estimation $P_{\tilde{\theta}}(x_i^j, \pi_i^k | D_l)$ for y_j, $\forall j \in J$. By lemma 5.2.7, (5.33) is the MVUE estimator for the complete data augmented using $\tilde{\theta}$.

Now we got the ML estimates for problem (5.26). Recall that our goal is to find the ML estimates that best model the incomplete data set D, we treat the estimates $\hat{\theta}$ as the current guess of true parameter θ and do the E-step and M-step again. Repeat such process and we get the EM algorithm for discrete-valued belief network. If we define operator $H(\psi)$ as

$$H(\psi)(ijk) = \frac{\sum_{l=1}^{L} P_{\psi}(x_i^j, \pi_i^k | D_l)}{\sum_{l=1}^{L} P_{\psi}(\pi_i^k | D_l)}, \tag{5.34}$$

then the EM process can be summarized as the iteration

$$\tilde{\theta}^{(p+1)} = H(\tilde{\theta}^{(p)}), \tag{5.35}$$

for some initial $\tilde{\theta}^{(0)}$. The EM algorithm can be thought of finding the *fixed point* of operator H, and, hopefully, such fixed point $\tilde{\theta}^*$ is just the best set of parameters that model the data set D.

More generally, we can extend (5.35) to the small-step size version of iteration, which falls within the stochastic approximation framework [90][91][92][93], as shown below:

$$\tilde{\theta}^{(p+1)} = (1-\gamma_p)\tilde{\theta}^{(p)} + \gamma_p H(\tilde{\theta}^{(p)}), \tag{5.36}$$

where $\gamma_p \in [0,1]$. Obviously, if $\gamma_p = 0$, $\tilde{\theta}^{(p+1)} = \tilde{\theta}^{(p)}$ and we ignores the influence from data; if $\gamma_p = 1$, however, we get (5.35) as a special case. We assume from now on the nontrivial case where $\gamma_p > 0$.

If we define operator $M(\psi)$ as

$$M(\psi) = (1-\gamma)\psi + \gamma H(\psi), \tag{5.37}$$

with ψ appropriately chosen, then the EM algorithm can be summarized as the iteration

$$\tilde{\theta}^{(p+1)} = M(\tilde{\theta}^{(p)}), \tag{5.38}$$

and the goal here is to find the fixed point $\tilde{\theta}^*$ of operator M.

5.3.3 Optimality and Convergence

Lemma 5.3.1 *For one sample $D = [D_1]$, the algorithm $\tilde{\theta}^{(p+1)} = H(\tilde{\theta}^{(p)})$ makes $\log L(\theta; D)$ non-decrease for each iteration.*

Proof. The following proof resembles that in [89]. Let Y denote the observed nodes and Z denote the non-observed nodes. So $\log L(\theta; D) = \log L(\theta; Y = y)$, and $X = [Y, Z]$. Given those nodes in Y "clamped" as indicated in D_1, we can obtain $\tilde{\theta}^{(p)}$ by using the augmented complete data set $X = X(y)$, where

$X(y)$ denotes the multiple cases of X that contains $Y = y$. Such $X(y)$ may be governed by some unknown parameter set θ, which we want to estimate using ML principle at M-step. For simplicity, we use Y to denote $Y = y$. For

$$\log L(\theta; Y) = \log P_\theta(Y) = \log P_\theta(X(y)) - \log P_\theta(X(y)|Y), \tag{5.39}$$

we take expectation with respect to the conditional probability density of X given Y under current parameter set $\tilde{\theta}^{(p)}$, and get

$$\begin{aligned} E_{\tilde{\theta}^{(p)}}\{\log P_\theta(Y)\} &= \log P_\theta(Y) \\ &= E_{\tilde{\theta}^{(p)}}\{\log P_\theta(X(y))\} - E_{\tilde{\theta}^{(p)}}\{\log P_\theta(X(y)|Y)\} \\ &= Q(\theta, \tilde{\theta}^{(p)}) - T(\theta, \tilde{\theta}^{(p)}), \end{aligned} \tag{5.40}$$

where $Q(\theta, \tilde{\theta}^{(p)}) = E_{\tilde{\theta}^{(p)}}\{\log P_\theta(X(y))\}$, $T(\theta, \tilde{\theta}^{(p)}) = E_{\tilde{\theta}^{(p)}}\{\log P_\theta(X(y)|Y)\}$.

At M-step, as described in section 5.3.2, we use the ML principle to find the most probable parameter based on the complete data set $X(y)$, each entry of which is appropriately weighed according to its conditional density under $\tilde{\theta}^{(p)}$.

$$\begin{aligned} \tilde{\theta}^{(p+1)} &= \arg\max_\theta Q(\theta, \tilde{\theta}^{(p)}) \\ &= \arg\max_\theta \sum_{x \in X(y)} p_{\tilde{\theta}^{(p)}}(x|y) \log P_\theta(x). \end{aligned} \tag{5.41}$$

and therefore,

$$Q(\tilde{\theta}^{(p+1)}, \tilde{\theta}^{(p)}) \geq Q(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}). \tag{5.42}$$

Also for any θ,

$$\begin{aligned} T(\theta, \tilde{\theta}^{(p)}) - T(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}) &= E_{\tilde{\theta}^{(p)}}\{\log \frac{P_\theta(X|Y)}{P_{\tilde{\theta}^{(p)}}(X|Y)}\} \\ &= -D(\tilde{\theta}^{(p)}||\theta) \\ &\leq 0, \end{aligned} \tag{5.43}$$

where $D(\tilde{\theta}^{(p)}||\theta) \geq 0$ is the relative entropy between $P_{\tilde{\theta}^{(p)}}(X|Y)$ and $P_\theta(X|Y)$, see [94]. By (5.42) and (5.43), we obtain $\log L(\tilde{\theta}^{(p+1)}; D) \geq \log L(\tilde{\theta}^{(p)}; D)$. $\square$

Lemma 5.3.2 *For independent uniform $D = [D_1, \ldots, D_L]$ with each $D_i \in R^n$ complying with the same schema S^+, the algorithm $\tilde{\theta}^{(p+1)} = H(\tilde{\theta}^{(p)})$ makes $\log L(\theta; D)$ non-decrease for each iteration.*

Proof. At E-step, we can obtain for each D_l the augmentation and thus the complete data set $C(D)$ under $\tilde{\theta}^{(p)}$. Let $D^+ = C(D)$. Observing that $D_1, \ldots, D_L$ are independent observations and also, each D_l^+ is obtained independently of D_m, $m \neq l$, we have

$$\log P_\theta(D_l) = \log P_\theta(D_l^+) - \log P_\theta(D_l^+ | D_l). \tag{5.44}$$

Take expectation with respect to the conditional probability density of D_l^+ given D_l under $\tilde{\theta}^{(p)}$, we get

$$\begin{aligned} \log P_\theta(D_l) &= E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D_l^+)\} - E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D_l^+ | D_l)\} \\ &= Q_l(\theta, \tilde{\theta}^{(p)}) - T_l(\theta, \tilde{\theta}^{(p)}), \end{aligned} \tag{5.45}$$

where $Q_l(\theta, \tilde{\theta}^{(p)}) = E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D_l^+)\}$ and $T_l(\theta, \tilde{\theta}^{(p)}) = E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D_l^+ | D_l)\}$.

Then for each D_l under $\tilde{\theta}^{(p)}$, we have

$$\log L(\theta; D) = \sum_l Q_l(\theta, \tilde{\theta}^{(p)}) - \sum_l T_l(\theta, \tilde{\theta}^{(p)}), \tag{5.46}$$

where

$$Q_l(\theta, \tilde{\theta}^{(p)}) = \sum_q P_{\tilde{\theta}^{(p)}}(D_l^+(q) | D_l) \log P_\theta(D_l^+(q)). \tag{5.47}$$

Like in Lemma 5.3.1, $T_l(\theta, \tilde{\theta}^{(p)}) \leq T_l(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)})$, for any $\theta \neq \tilde{\theta}^{(p)}$, so we have

$$\sum_l T_l(\theta, \tilde{\theta}^{(p)}) \leq \sum_l T_l(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}). \tag{5.48}$$

At M-step, $\tilde{\theta}^{(p+1)} = \arg\max_\theta E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D^+)\}$, where

$$E_{\tilde{\theta}^{(p)}}\{\log P_\theta(D^+)\} = \sum_l Q_l(\theta, \tilde{\theta}^{(p)}). \tag{5.49}$$

Thus we can see that

$$\sum_l Q_l(\tilde{\theta}^{(p+1)}, \tilde{\theta}^{(p)}) \geq \sum_l Q_l(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}). \tag{5.50}$$

Combine (5.46), (5.48) and (5.50) we get $\log L(\tilde{\theta}^{(p+1)}; D) \geq \log L(\tilde{\theta}^{(p)}; D)$. □

Lemma 5.3.3 *For independent nonuniform $D = [D_1, \ldots, D_L]$ with each $D_i \in R^n$ complying with schema S_i^+ ,the algorithm $\tilde{\theta}^{(p+1)} = H(\tilde{\theta}^{(p)})$ makes $\log L(\theta; D)$ non-decrease for each iteration.*

Proof. For formulae (5.44)–(5.50), we did not use the assumption of uniform data, the proof above can be adapted here. □

Proposition 5.3.1 *Given independent (but not necessarily uniform) data set $D = [D_1, \ldots, D_L]$, the algorithm $\tilde{\theta}^{(p+1)} = H(\tilde{\theta}^{(p)})$ makes $\log L(\theta; D)$ non-decrease for each iteration, or $\log L(H(\tilde{\theta}^{(p)}); D) \geq \log L(\tilde{\theta}^{(p)}; D)$.*

Proof. The proof is straightforward from Lemma 5.3.1, 5.3.2 and 5.3.3. □

Proposition 5.3.2 *Given independent (but not necessarily uniform) data set $D = [D_1, \ldots, D_L]$, the algorithm $\tilde{\theta}^{(p+1)} = M(\tilde{\theta}^{(p)})$ makes $L(\theta; D)$ non-decrease for each iteration.*

Proof. We rewrite (5.46) as

$$\log L(\theta; D) = Q(\theta, \tilde{\theta}^{(p)}) - T(\theta, \tilde{\theta}^{(p)}) \tag{5.51}$$

where $Q(\theta, \tilde{\theta}^{(p)}) = \sum_l Q_l(\theta, \tilde{\theta}^{(p)})$, $T(\theta, \tilde{\theta}^{(p)}) = \sum_l T_l(\theta, \tilde{\theta}^{(p)})$.

We have as usual for any $\theta \neq \tilde{\theta}^{(p)}$,

$$T(\theta, \tilde{\theta}^{(p)}) \leq T(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}). \tag{5.52}$$

For $Q(\theta, \tilde{\theta}^{(p)})$, we have

$$Q(\theta, \tilde{\theta}^{(p)}) = \sum_l \sum_q P_{\tilde{\theta}^{(p)}}(D_l^+(q)|D_l) \log P_\theta(D_l^+(q)). \tag{5.53}$$

By lemma 5.2.5 we see that each $\log P_\theta(D_l^+(q))$ is negative and strictly concave in θ. We conclude that $Q(\theta, \tilde{\theta}^{(p)})$ is also negative and strictly concave since it is a linear combination of them. So for operator M with $\gamma_p \in (0, 1]$,

$$\begin{aligned} Q(M(\tilde{\theta}^{(p)}), \tilde{\theta}^{(p)}) &= Q((1-\gamma_p)\tilde{\theta}^{(p)} + \gamma_p H(\tilde{\theta}^{(p)}), \tilde{\theta}^{(p)}) \\ &\geq (1-\gamma_p) Q(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}) + \gamma_p Q(H(\tilde{\theta}^{(p)}), \tilde{\theta}^{(p)}) \quad \text{by concavity} \\ &\geq (1-\gamma_p) Q(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}) + \gamma_p Q(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}) \quad \text{by proposition 5.3.1} \\ &= Q(\tilde{\theta}^{(p)}, \tilde{\theta}^{(p)}). \end{aligned} \tag{5.54}$$

Combine (5.52) and (5.54) we see that $\log L(M(\tilde{\theta}^{(p)}); D) \geq \log L(\tilde{\theta}^{(p)}; D)$. □

Proposition 5.3.3 *The algorithm* $\tilde{\theta}^{(p+1)} = M(\tilde{\theta}^{(p)})$ *will make* $\log L(\theta; D)$ *converge to* $\log L(\theta^*; D)$*, where* θ^* *is the fixed point of operator* M*, and* θ^* *is the local maxima.*

Proof. From proposition 5.3.2 we see that $\log L(\theta; D)$ is non-decreasing under operator M; also from lemma 5.2.5, we know that $\log L(\theta; D) < 0$, bounded above. So the sequence $\{\log L(\theta; D)\}$ under M converges to the limit, say $\log L^*(\theta; D)$. For continuous (and thus measurable) function $\log L(\theta; D)$, let θ^* be the set of parameters corresponding to $\log L^*(\theta; D)$, or $\log L(\theta^*; D) = \log L^*(\theta; D)$. From $\log L(\theta^{(p+1)}; D) - \log L(\theta^{(p)}; D) \longrightarrow 0$ and $\log L(\theta^{(p+1)}; D) \geq \log L(\theta^{(p)}; D)$, we see that $\nabla \log L(\theta^{(p)}; D) \longrightarrow 0$ and so θ^* is a stationary point. Now we turn to prove that θ^* is the fixed point of operator M.

For the constrained optimization problem (5.24), we exploit the Lagrange

Multiplier method [95] and define the Lagrangian as

$$L = \log P_\theta(D) - \lambda(\sum_{j'} \theta_{ij'k} - 1), \tag{5.55}$$

which implies that $\partial L/\partial\theta_{ijk} = \partial \log P_\theta(D)/\partial\theta_{ijk} - \lambda$.

The gradient $\partial \log P_\theta(D)/\partial\theta_{ijk}$ can be computed locally by using information that is available in the normal course of belief network calculations.

$$\begin{aligned} \frac{\partial \log P_\theta(D)}{\partial\theta_{ijk}} &= \sum_{l=1}^{L} \frac{\partial \log P_\theta(D_l)}{\partial\theta_{ijk}} \\ &= \sum_{l=1}^{L} \frac{\partial P_\theta(D_l)/\partial\theta_{ijk}}{P_\theta(D_l)}. \end{aligned} \tag{5.56}$$

In order to get an expression in terms of information local to the parameter θ_{ijk}, we introduce X_i and Π_i by averaging over their possible values:

$$\begin{aligned} &\frac{\partial P_\theta(D_l)/\partial\theta_{ijk}}{P_\theta(D_l)} \\ = &\frac{\frac{\partial}{\partial\theta_{ijk}}\left(\sum_{j',k'} P_\theta(D_l|x_i^{j'}, \pi_i^{k'}) P_\theta(x_i^{j'}, \pi_i^{k'})\right)}{P_\theta(D_l)} \\ = &\frac{\frac{\partial}{\partial\theta_{ijk}}\left(\sum_{j',k'} P_\theta(D_l|x_i^{j'}, \pi_i^{k'}) P_\theta(x_i^{j'}|\pi_i^{k'}) P_\theta(\pi_i^{k'})\right)}{P_\theta(D_l)}. \end{aligned} \tag{5.57}$$

Observe that the important property of this expression is that θ_{ijk} appears only in one term in the summation: the term for $j' = j, k' = k$. For this term, $P_\theta(x_i^{j'}|\pi_i^{k'})$ is just θ_{ijk}, so we have

$$\begin{aligned} \frac{\partial P_\theta(D_l)/\partial\theta_{ijk}}{P_\theta(D_l)} &= \frac{P_\theta(D_l|x_i^j, \pi_i^k) P_\theta(\pi_i^k)}{P_\theta(D_l)} \\ &= \frac{P_\theta(x_i^j, \pi_i^k|D_l) P_\theta(D_l) P_\theta(\pi_i^k)}{P_\theta(x_i^j, \pi_i^k) P_\theta(D_l)} \\ &= \frac{P_\theta(x_i^j, \pi_i^k|D_l)}{P_\theta(x_i^j|\pi_i^k)} = \frac{P_\theta(x_i^j, \pi_i^k|D_l)}{\theta_{ijk}}. \end{aligned} \tag{5.58}$$

Since θ^* is the stationary point, we have $[\partial \log P_\theta(D)/\partial\theta_{ijk} - \lambda]_{\theta=\theta^*} = 0$, or

$$\sum_{l}^{L} \frac{P_{\theta^*}(x_i^j, \pi_i^k|D_l)}{\theta^*_{ijk}} - \lambda = 0, \tag{5.59}$$

$$\begin{aligned}\lambda\theta^*_{ijk} &= \sum_l^L P_{\theta^*}(x_i^j,\pi_i^k|D_l) \quad \Rightarrow \\ \lambda\sum_j \theta^*_{ijk} &= \sum_l^L\sum_j P_{\theta^*}(x_i^j,\pi_i^k|D_l) \quad \Rightarrow \\ \lambda &= \sum_{l=1}^L P_{\theta^*}(\pi_i^k|D_l). \end{aligned} \tag{5.60}$$

so we have $\theta^*_{ijk} = \sum_l^L P_{\theta^*}(x_i^j,\pi_i^k|D_l)/\lambda = \sum_l^L P_{\theta^*}(x_i^j,\pi_i^k|D_l)/\sum_l^L P_{\theta^*}(\pi_i^k|D_l)$, or θ^* is the fixed point for operator H.

It is straightforward to check that $\theta^* = M(\theta^*)$, and θ^* is the local maxima. □

Theorem 5.3.4 *After convergence, θ^* is the MVUE for augmented complete data $C(D)$.*

Proof. By lemma 5.2.7, we see that after every iteration, $\tilde{\theta}^{(p+1)}$ is MVUE for $C(D)$ obtained under $\tilde{\theta}^{(p)}$. θ^* is the fixed point and thus MVUE. □

5.3.4 Convergence Rate and Choice of Step Size

For the 3-dimensional matrix θ, we define $\psi = \theta_{ik}$, which is a $J \times 1$ vector with $\psi_j = \theta_{ijk}$ as the j^{th} component. We study the convergence rate problem of matrix θ by looking at each such ψ. It can be shown that at the neighborhood of ψ^*,

$$\psi^{(p+1)} - \psi^* \approx J(\psi^*)(\psi^{(p)} - \psi^*), \tag{5.61}$$

where $J(\psi^*)$ is the Jacobian matrix and the rate of convergence is defined as

$$\nu = \lim_{p\to\infty} \frac{\parallel \psi^{(p+1)} - \psi^* \parallel}{\parallel \psi^{(p)} - \psi^* \parallel}. \tag{5.62}$$

Usually, under some regularity conditions, the rate of convergence is

$$\nu = \lambda_{\max}(J(\psi^*)) \quad \text{the largest eigenvalue of } J(\psi^*). \tag{5.63}$$

Let $J^H(\psi)$ and $J^M(\psi)$ denote the Jacobian matrices under operator H and M, respectively. To obtain $J^H(\psi)$ and $J^M(\psi)$ under $\psi = \psi^*$, we make the following definitions first.

Definition 5.3.5 *The gradient vector of* $\log L(\theta; D)$ *with respect to* ψ *is*

$$S(D;\psi) = \partial \log L(\theta; D)/\partial\psi = \sum_l S(D_l;\psi). \tag{5.64}$$

We can see from (5.58) that

$$S(D;\psi)(j) = \sum_l P_\theta(x_i^j, \pi_i^k | D_l)/\theta_{ijk}. \tag{5.65}$$

Definition 5.3.6 *The gradient vector of* $\log L_c(\theta; D^+)$ *with respect to* ψ *is*

$$S_c(D^+;\psi) = \partial \log L_c(\theta; D^+)/\partial\psi. \tag{5.66}$$

Similarly we have

$$S_c(D^+;\psi) = \sum_l S_c(D_l^+;\psi) \tag{5.67}$$

where $S_c(D_l^+;\psi) = \partial \log L_c(\theta; D_l^+)/\partial\psi$, and as in (5.58)

$$S_c(D_l^+;\psi)(j) = \frac{1}{\theta_{ijk}} P_\theta(x_i^j, \pi_i^k | D_l^+). \tag{5.68}$$

Lemma 5.3.7 $S(D;\psi) = E_\theta\{S_c(D^+;\psi)|D\} = \left[\frac{\partial}{\partial\psi} Q(\theta, \theta^{(p)})\right]_{\theta=\theta^{(p)}}$.

Proof.

$$\begin{aligned} S(D_l;\psi)(j) &= \frac{1}{\theta_{ijk}} P_\theta(x_i^j, \pi_i^k | D_l) \\ &= \frac{1}{\theta_{ijk}} \sum_q P_\theta(D_l^+(q)|D_l) P_\theta(x_i^j, \pi_i^k | D_l^+(q)) \\ &= \frac{1}{\theta_{ijk}} E_\theta\left\{P_\theta(x_i^j, \pi_i^k | D_l^+)\right\} = E_\theta\left\{S_c(D_l^+;\psi)(j)\right\} \end{aligned} \tag{5.69}$$

Given current $\theta^{(p)}$, $Q(\theta, \theta^{(p)}) = \sum_l E_{\theta^{(p)}}\left\{\log P_\theta(D_l^+)\right\}$ and so

$$\frac{\partial}{\partial\theta_{ijk}} Q(\theta, \theta^{(p)}) = \sum_l E_{\theta^{(p)}}\left\{\frac{\partial}{\partial\theta_{ijk}} \log P_\theta(D_l^+)\right\} \tag{5.70}$$

and we obtain the proof. □

Definition 5.3.8 *The negative of Hessian matrix under incomplete data is*

$$I(\psi; D) = -\partial^2 \log L(\theta; D)/\partial\psi\partial\psi^T.$$

Definition 5.3.9 *The negative of Hessian matrix under complete data is*

$$I_c(\psi; D^+) = -\partial^2 \log L_c(\theta; D^+)/\partial\psi\partial\psi^T.$$

Lemma 5.3.10 $I(\psi; D) = \mathcal{I}_c(\psi; D) - \mathcal{I}_m(\psi; D)$*, where* $\mathcal{I}_c(\psi; D) \triangleq E_\theta\{I_c(\psi; D^+)|D\}$ *and* $\mathcal{I}_m(\psi; D) \triangleq E_\theta\{\partial^2 \log P_\theta(D^+|D)/\partial\psi\partial\psi^T\}$.

Proof. This lemma is called the missing information principle. The following proof is adapted from [96]. Since $L(\theta; D) = \sum_l \log L_c(\theta; D^+) - \sum_l \log P_\theta(D_l^+|D_l)$,

$$\begin{aligned} I(\psi; D) &= E_\theta\{I(\psi; D)|D\} \\ &= -\sum_l \sum_q P_\theta(D_l^+(q)|D_l)\partial^2 \log L_c(\theta; D_l^+(q))/\partial\psi\partial\psi^T \\ &\quad + \sum_l \sum_q P_\theta(D_l^+(q)|D_l)\partial^2 \log P_\theta(D_l^+(q)|D_l)/\partial\psi\partial\psi^T \\ &= E_\theta\{I_c(\psi; D^+)|D\} + E_\theta\{\partial^2 \log P_\theta(D^+|D)/\partial\psi\partial\psi^T\}. \end{aligned} \tag{5.71}$$

$\Rightarrow I(\psi; D) = \mathcal{I}_c(\psi; D) - \mathcal{I}_m(\psi; D)$. □

Lemma 5.3.11 *For operator* H, $J^H(\psi^*) = \mathcal{I}_c^{-1}(\psi^*; D)\,\mathcal{I}_m(\psi^*; D)$.

Proof. This lemma is adapted from Dempster's original work on EM algorithm [88]. By proposition 5.3.3 we know that $S(D; \psi^*) = 0$ and at the neighborhood of ψ^*,

$$\begin{aligned} S(D; \psi^*) &\approx S(D; \psi^{(p)}) - I(\psi^{(p)}; D)(\psi^* - \psi^{(p)}) \quad \Rightarrow \\ \psi^* &\approx \psi^{(p)} + I^{-1}(\psi^{(p)}; D)S(D; \psi^{(p)}) \end{aligned} \tag{5.72}$$

At M-step, for $Q(\psi, \psi^{(p)}) = \sum_l \sum_q P_{\theta^{(p)}}(D_l^+(q)|D_l) \log P_\theta(D_l^+(q))$ we have

$$\begin{aligned}
0 &= [\partial Q(\psi, \psi^{(p)})/\partial\psi]_{\psi=\psi^{(p+1)}} \\
&\approx \underbrace{[\partial Q(\psi, \psi^{(p)})/\partial\psi]_{\psi=\psi^{(p)}}}_{S(D;\psi^{(p)}) \text{ by lemma 5.3.7}} + [\partial^2 Q(\psi, \psi^{(p)})/\partial\psi\partial\psi^T]_{\psi=\psi^{(p)}} (\psi^{(p+1)} - \psi^{(p)}) \\
&= S(D; \psi^{(p)}) - \mathcal{I}_c(\psi^{(p)}; D)(\psi^{(p+1)} - \psi^{(p)}) \quad \Rightarrow \\
S(D; \psi^{(p)}) &\approx \mathcal{I}_c(\psi^{(p)}; D)(\psi^{(p+1)} - \psi^{(p)}). \qquad (5.73)
\end{aligned}$$

From (5.72) and (5.73) we have

$$\begin{aligned}
\psi^* - \psi^{(p)} &\approx I^{-1}(\psi^{(p)}; D)\mathcal{I}_c(\psi^{(p)}; D)(\psi^{(p+1)} - \psi^{(p)}) \quad \Rightarrow \\
\psi^{(p+1)} - \psi^* &\approx [I_J - \mathcal{I}_c^{-1}(\psi^{(p)}; D)I(\psi^{(p)}; D)](\psi^{(p)} - \psi^*) \\
&\approx [I_J - \mathcal{I}_c^{-1}(\psi^*; D)I(\psi^*; D)](\psi^{(p)} - \psi^*) \\
&\approx \mathcal{I}_c^{-1}(\psi^*; D)\mathcal{I}_m(\psi^*; D)(\psi^{(p)} - \psi^*). \quad \text{by lemma 5.3.10} \qquad (5.74)
\end{aligned}$$

$\Rightarrow J^H(\psi^*) = \mathcal{I}_c^{-1}(\psi^*; D)\mathcal{I}_m(\psi^*; D)$. □

Lemma 5.3.12 $\mathcal{I}_m(\psi^*; D) = \sum_l \sum_q P_\theta(D_l^+(q)|D_l)\, S_c(D_l^+(q); \psi)\, S_c^T(D_l^+(q); \psi) - \sum_l S(D_l; \psi)\, S^T(D_l; \psi)$.

Proof. This lemma is adapted from that proposed by Louis in 1982 [97].

$$\begin{aligned}
I(\psi; D) &= -\partial S(D; \psi)/\partial\psi \\
&= -\sum_l \partial S(D_l; \psi)/\partial\psi = -\sum_l \partial[\frac{\partial L(\theta; D_l)/\partial\psi}{L(\theta; D_l)}]/\partial\psi \\
&= -\sum_l \partial[\frac{\sum_q \partial L_c(\theta; D_l^+(q))/\partial\psi}{L(\theta; D_l)}]/\partial\psi \\
&= -\sum_l \sum_q \frac{\partial^2 L_c(\theta; D_l^+(q))/\partial\psi\partial\psi^T}{L(\theta; D_l)} \\
&\quad + \sum_l \left[\frac{\sum_q \partial L_c(\theta; D_l^+(q))\,/\partial\psi}{L(\theta; D_l)}\right] \left[\frac{\sum_q \partial L_c(\theta; D_l^+(q))\,/\partial\psi}{L(\theta; D_l)}\right]^T
\end{aligned}$$

$$
\begin{aligned}
&= -\sum_l \sum_q \frac{\partial^2 L_c(\theta; D_l^+(q)) \,/\partial\psi\partial\psi^T}{L(\theta; D_l)} + \sum_l S(D_l;\psi)\, S^T(D_l;\psi) \\
&= -\sum_l \sum_q \left[\partial^2 \log L_c(\theta; D_l^+(q)) \,/\partial\psi\partial\psi^T\right] \frac{L(\theta; D_l^+(q))}{L(\theta; D_l)} \\
&\quad -\sum_l \sum_q \left[\frac{\partial L_c(\theta; D_l^+(q)) \,/\partial\psi}{L_c(\theta; D_l^+(q))}\right] \left[\frac{\partial L_c(\theta; D_l^+(q)) \,/\partial\psi}{L_c(\theta; D_l^+(q))}\right]^T + \sum_l S(D_l;\psi)\, S^T(D_l;\psi) \\
&= \sum_l \sum_q I_c(\psi; D_l^+(q))\, P_\theta(D_l^+(q)|D_l) \\
&\quad -\sum_l \sum_q S_c(D_l^+(q);\psi)\, S_c^T(D_l^+(q);\psi)\, P_\theta(D_l^+(q)|D_l) \\
&\quad +\sum_l S(D_l;\psi)\, S^T(D_l;\psi). \qquad (5.75)
\end{aligned}
$$

Since $\sum_l \sum_q I_c(\psi; D_l^+(q))\, P_\theta(D_l^+(q)|D_l) = \mathcal{I}(\psi; D)$, and by lemma 5.3.10 we finish the proof. □

So at the neighborhood of θ^*, we can now compute the Jacobian matrix.

Proposition 5.3.4 *Given independent (but not necessarily uniform) data set $D = [D_1, \ldots, D_L]$, the EM algorithm under H operator gives*

$$
\mathcal{I}_c(\psi^*; D) = diag\left\{\sum_l P_\theta(x_i^j, \pi_i^k|D_l)/\theta_{ijk}^2\right\}.
$$

Proof. Obviously, $I_c(\psi; D_l^+(q)) = -\partial S_c(D_l^+(q);\psi)\,/\partial\psi$, where $S_c(D_l^+(q);\psi)(j) = I_\theta(x_i^j, \pi_i^k|D_l^+(q))\,/\theta_{ijk}$. So for $j \neq j'$, $\partial S_c(D_l^+(q);\psi)(j)/\partial\psi_{j'} = 0$ and

$$
\begin{aligned}
\mathcal{I}_c(\psi; D) &= \sum_l \sum_q P_\theta(D_l^+(q)|D_l)\, I_c(\psi; D_l^+(q)) \\
&= \sum_l \sum_q P_\theta(D_l^+(q)|D_l)\, \mathrm{diag}\{I_\theta(x_i^j, \pi_i^k|D_l^+(q))\,/\theta_{ijk}^2\} \\
&= \mathrm{diag}\left\{\sum_l \sum_q P_\theta(D_l^+(q)|D_l)\, I_\theta(x_i^j, \pi_i^k|D_l^+(q))\,/\,\theta_{ijk}^2\right\} \\
&= \mathrm{diag}\left\{\sum_l P_\theta(x_i^j, \pi_i^k|D_l)/\theta_{ijk}^2\right\}. \qquad (5.76)
\end{aligned}
$$

We define $a_j \triangleq \sum_l P_\theta(x_i^j, \pi_i^k|D_l)/\theta_{ijk}^2 \quad j = 1, \ldots, J,$ for convenience. □

Proposition 5.3.5 *If we let* $V = \sum_l S(D_l;\psi)\, S^T(D_l;\psi)$, *then*

$$\mathcal{I}_m(\psi^*; D) = diag\{a_j\} - V.$$

Proof. From $S_c(D_l^+(q);\psi)(j) = I_\theta(x_i^j, \pi_i^k | D_l^+(q))\,/\theta_{ijk}$, we get

$$I_\theta(x_i^j, \pi_i^k | D_l^+(q))\, I_\theta(x_i^{j'}, \pi_i^k | D_l^+(q)) = 0,\ \forall j' \neq j$$

and thus

$$S_c(D_l^+(q);\psi)\, S_c^T(D_l^+(q);\psi) = \mathrm{diag}\{I_\theta(x_i^j, \pi_i^k | D_l^+(q))\,/\,\theta_{ijk}^2\}, \qquad (5.77)$$

and similarly we have

$$\sum_l \sum_q S_c(D_l^+(q);\psi)\, S_c^T(D_l^+(q);\psi)\, P_\theta(D_l^+(q)|D_l) = \mathrm{diag}\,\{a_j\}\,. \qquad (5.78)$$

Since $S(D_l;\psi)(j) = P_\theta(x_i^j, \pi_i^k|D_l)|\theta_{ijk}$, we have for matrix $V = \{v_{ms}\}$:

$$v_{ms} = \begin{cases} \frac{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)\, P_\theta(x_i^s, \pi_i^k|D_l)}{\theta_{imk}\,\theta_{isk}} & \text{if } m \neq s \\ \\ \frac{\sum_l P_\theta^2(x_i^m, \pi_i^k|D_l)}{\theta_{imk}^2} & \text{if } m = s \end{cases} \qquad (5.79)$$

□

Proposition 5.3.6 *Given independent (but not necessarily uniform) data set* $D = [D_1, \ldots, D_L]$, *the Jacobian matrix under* H *operator is*

$$J^H(\psi^*) = I_J -\ diag\{1/a_j\}\, V.$$

Proof.

$$\begin{aligned} J^H(\psi^*) &= \mathcal{I}_c^{-1}(\psi^*; D)\, \mathcal{I}_m(\psi^*; D) \\ &= \mathrm{diag}\{1/a_j\}\ [\mathrm{diag}\{a_j\} - V] \\ &= I_J -\ \mathrm{diag}\{1/a_j\}\, V. \end{aligned} \qquad (5.80)$$

□

We define $\bar{W} = \operatorname{diag}\{1/a_j\}\, V$, and for each of its element,

$$\begin{aligned}\bar{w}_{mm} &= \frac{\theta^2_{imk}}{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)} \frac{\sum_l P^2_\theta(x_i^m, \pi_i^k|D_l)}{\theta^2_{imk}} \\ &= \frac{\sum_l P^2_\theta(x_i^m, \pi_i^k|D_l)}{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)} \qquad (5.81)\end{aligned}$$

$$\begin{aligned}\bar{w}_{ms} &= \frac{\theta^2_{imk}}{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)} \frac{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)\, P_\theta(x_i^s, \pi_i^k|D_l)}{\theta_{imk}\,\theta_{isk}} \\ &= \frac{\theta_{imk}}{\theta_{isk}\,\sum_l P_\theta(x_i^m, \pi_i^k|D_l)} \sum_l P_\theta(x_i^m, \pi_i^k|D_l)\, P_\theta(x_i^s, \pi_i^k|D_l) \\ &= \frac{\sum_l P_\theta(x_i^m, \pi_i^k|D_l)\, P_\theta(x_i^s, \pi_i^k|D_l)}{\sum_l P_\theta(x_i^s, \pi_i^k|D_l)}, \qquad (5.82)\end{aligned}$$

based on the fact that if $\theta = \theta^*$, $\theta_{imk} = \sum_l P_\theta(x_i^m, \pi_i^k|D_l) \,/\, \sum_l P_\theta(\pi_i^k|D_l)$ under operator H. If we define $w_{jl} = P_\theta(x_i^j, \pi_i^k|D_l)$ at $\theta = \theta^*$, it is easy to see that $\bar{W} = W\,\Lambda$, where $\Lambda = \operatorname{diag}\{1/\sum_l w_{jl}\}$ and

$$W = \begin{pmatrix} \sum_l w^2_{1l} & \sum_l w_{1l}w_{2l} & \dots & \sum_l w_{1l}w_{Jl} \\ \sum_l w_{1l}w_{2l} & \sum_l w^2_{2l} & \dots & \sum_l w_{1l}w_{Jl} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_l w_{1l}w_{Jl} & \sum_l w_{1l}w_{2l} & \dots & \sum_l w^2_{Jl} \end{pmatrix} \qquad (5.83)$$

Obviously, $W = \sum_l w_l w_l^T$ where $w_l^T = [w_{1l}, \dots, w_{Jl}]$.

Proposition 5.3.7 *Given independent (but not necessarily uniform) data set $D = [D_1, \dots, D_L]$, the Jacobian matrix under M operator is*

$$J^M(\psi^*) = I_J - \gamma W\,\Lambda.$$

Proof. For operator M, we have

$$\begin{aligned}J^M(\psi^*) &= (1-\gamma)I_J + \gamma J^H(\psi^*) \\ &= I_J - \gamma W\,\Lambda. \qquad (5.84)\end{aligned}$$

□

To study the eigenvalues of $J^M(\psi^*)$, we first look at those of $W\Lambda$. It is straightforward to see that for any non-zero vector $\xi \in R^J$,

$$\begin{aligned}\lambda_{\max}(\bar{W}) &= \max_{\xi} \frac{(\xi, \sum_l w_l w_l^T \Lambda \xi)}{(\xi, \xi)} \\ &= \max_{\xi} \sum_l \underbrace{\frac{(\xi, w_l w_l^T \Lambda \xi)}{(\xi, \xi)}}_{\leq \lambda_{\max}(w_l w_l^T \Lambda)} \leq \sum_l \lambda_{\max}(w_l w_l^T \Lambda) \quad \text{and,} \end{aligned} \tag{5.85}$$

$$\begin{aligned}\lambda_{\min}(\bar{W}) &= \min_{\xi} \frac{(\xi, \sum_l w_l w_l^T \Lambda \xi)}{(\xi, \xi)} \\ &= \min_{\xi} \sum_l \underbrace{\frac{(\xi, w_l w_l^T \Lambda \xi)}{(\xi, \xi)}}_{\geq \lambda_{\min}(w_l w_l^T \Lambda)} \geq \sum_l \lambda_{\min}(w_l w_l^T \Lambda), \end{aligned} \tag{5.86}$$

and so we have the following lemma.

Lemma 5.3.13 $\lambda_{\max}(w_l w_l^T \Lambda) = \sum_j w_{jl}^2 / \sum_l w_{jl},\ \lambda_{\min}(w_l w_l^T \Lambda) = 0$

Proof. If we let $\Lambda = \text{diag}\{\rho_j\}$ with $\rho_j > 0$ (this condition stands true if data sample size L is large enough), then for any non-zero $\xi \in R^J$, we have

$$\begin{aligned}\sum_l \xi^T w_l w_l^T \Lambda \xi &= \sum_l (\xi_1 w_{1l} + \ldots + \xi_J w_{Jl}) (\rho_1 \xi_1 w_{1l} + \ldots + \rho_J \xi_J w_{Jl}) \\ &\geq \sum_l (\sqrt{\rho_1} \xi_1 w_{1l} + \ldots + \sqrt{\rho_J} \xi_j w_{jl})^2 \geq 0. \end{aligned} \tag{5.87}$$

So matrix $w_l w_l^T \Lambda$ is positive semi-definite, with all of its eigenvalues non-negative. Note also that in $w_l w_l^T \Lambda$, each row is of a constant factor of any other row, we see that [98]

$$\det (w_l w_l^T \Lambda) = \prod_j \lambda_j (w_l w_l^T \Lambda) = 0 \tag{5.88}$$

and $\lambda_{\min}(w_l w_l^T \Lambda) = 0$. It is also shown in [98] that rank one matrix $w_l w_l^T \Lambda$ has at most one non-zero eigenvalue and it is $w_l^T \Lambda w_l$, if exists. □

Theorem 5.3.14 $\lambda_{\max}(J^M(\psi^*)) \leq 1$, $\lambda_{\min}(J^M(\psi^*)) \geq 0$ *under* M *operator with* $0 < \gamma \leq \min(\frac{1}{\sum_l P_{\theta^*}(\pi_i^k|D_l)}, 1)$.

Proof.

$$\begin{aligned}
\lambda_{\max}(\bar{W}) &\leq \sum_l \lambda_{\max}(w_l w_l^T \Lambda) \\
&= \sum_j \left(\sum_l w_{jl}^2 / \sum_m w_{jm} \right) \\
&\leq \sum_j \left[(\sum_l w_{jl})^2 / \sum_m w_{jm} \right] \\
&= \sum_l \sum_j P_{\theta^*}(x_i^j, \pi_i^k | D_l) \\
&= \sum_l P_{\theta^*}(\pi_i^k | D_l) \qquad (5.89)
\end{aligned}$$

$$\lambda_{\min}(\bar{W}) \geq \sum_l \lambda_{\min}(w_l w_l^T \Lambda) = 0 \qquad (5.90)$$

Then for operator M with $0 < \gamma \leq \min(\frac{1}{\sum_l P_{\theta^*}(\pi_i^k|D_l)}, 1)$, we have $\lambda_{\max}(J^M(\psi^*)) = 1 - \gamma\lambda_{\min}(\bar{W}) \leq 1$ and $\lambda_{\min}(J^M(\psi^*)) = 1 - \gamma\lambda_{\max}(\bar{W}) \geq 0$. □

This theorem states that, with $0 < \gamma \leq \min(\frac{1}{\sum_l P_{\theta^*}(\pi_i^k|D_l)}, 1)$, operator M will make the parameter sequence converge to the local maxima θ^* at the neighborhood of θ^*. It is important to note that in the beginning of the iterations when $\sum_l P_\theta(\pi_i^k|D_l)$ is still small, we can choose the step size γ close to 1, which means gross adjustment of the parameters according to the data. As iterations go along and $\sum_l P_\theta(\pi_i^k|D_l)$ becomes larger, we are required to decrease our step sizes and put less weight to the information from data for finer adjustment of the estimates. Further, if there is random noise associated with the H operator, the effect of the noise on the variance of the estimates become vanishingly small and the possibility of convergence remains open, if we allow the step size to decrease to zero. In general, if we consider vector $\psi \in R^J$, for every component j, the choice of $\gamma_p(j)$ usually needs to fulfill the following conditions [99]:

$$\gamma_p(j) \geq 0 \tag{5.91}$$

$$\sum_{p=0}^{\infty} \gamma_p(j) = \infty \quad \text{with probability 1} \tag{5.92}$$

$$\sum_{p=0}^{\infty} \gamma_p^2(j) < \infty \quad \text{with probability 1} \tag{5.93}$$

5.4 Simulation Results

We use the following belief network example to show the effectiveness of the EM learning algorithms. Consider the example network shown in figure 5.1.

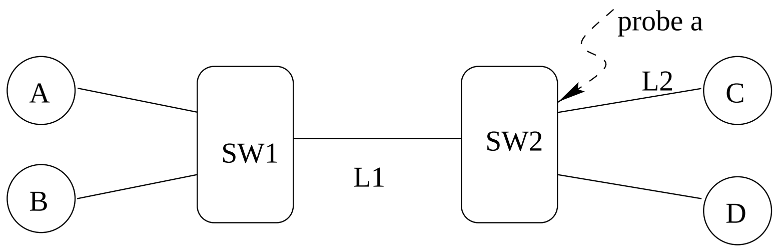

Figure 5.1: Example Network

Two switches $SW1$ and $SW2$ are connected to each other via link $L1$. Machines A and B are connected to $SW1$ and they would communicate with machines C and D, which are connected to switch $SW2$. We have a probe a hooked at the end of $SW2$ to measure the traffic throughput going out of $SW2$. Suppose the information we could obtain during network operation include whether or not: $SW1$ alarm is normal, A could connect $SW2$, B could connect $SW2$, A could connect C, C could connect $SW1$, throughput at probe a is normal, and D could connect $SW1$. The possible faults are identified as: $SW1$ works normal or not, $L1$ normal or congested, $SW2$ normal or not, and source pumped from C

to $L2$ is normal or not. We set up a belief network model for such situations, and figure 5.2 shows the structure of this belief network. The conditional probability distributions of each node are shown in figure 5.3, 5.4, 5.5, and 5.6.

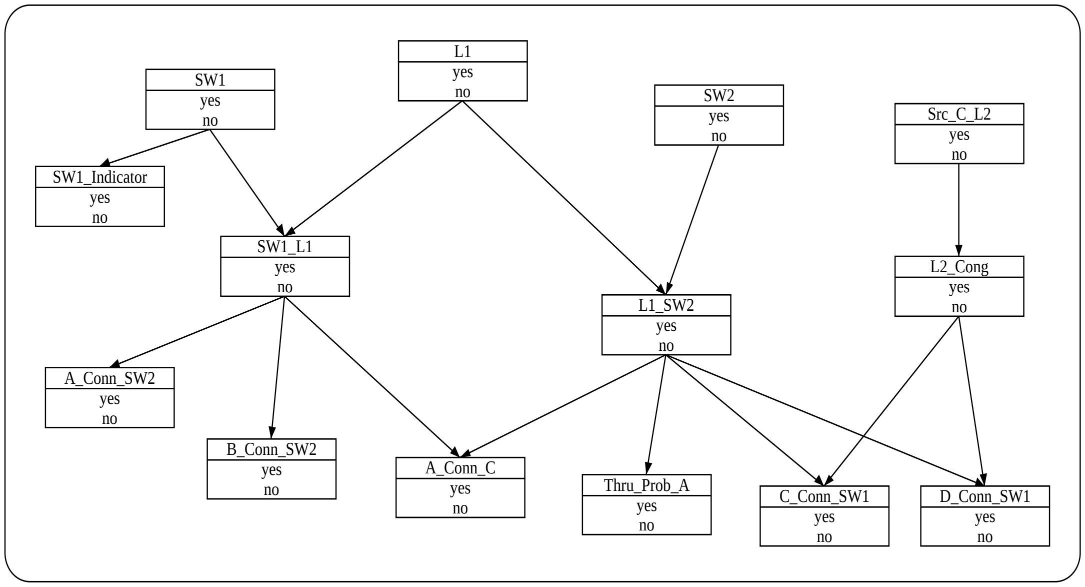

Figure 5.2: Belief Network for the Example Network

To show the effectiveness of the EM learning algorithm, we generate the experiment data using the true distribution probabilities as shown above. For the four root nodes, the chances of generating data is shown in table 5.1. No hidden node data are generated. One segment of the generated data file is shown in table 5.2. We generate 3000 samples using this schema, and extract 70% of the samples as the training set, with the remaining 30% samples as the testing set. The step sizes are chosen according to the requirements discussed in the previous section.

The initial conditional probability distributions of the example belief network are shown in figure 5.7, 5.8, 5.9, and 5.10. It could be seen that such distributions are more *flat* than their *true* counterparts.

Table 5.1: Chances of generating data for root nodes

SW1	SW2	L1	Src_C_L2
0.9	0.95	0.8	0.9

Table 5.2: Segment of the generated file

L1	S_C_L2	SW2	D_C_SW1	SW1	B_C_SW2	Th_P_A	C_C_SW1	A_C_SW2	A_C_C	SW1_Ind
”yes”	”no”	”yes”	”yes”		”yes”	”yes”	”yes”	”yes”	”yes”	”yes”
	”yes”		”no”		”no”	”no”	”no”	”yes”	”yes”	”yes”
	”no”	”no”	”no”	”yes”	”yes”	”no”	”no”	”yes”	”yes”	”yes”
”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”no”	”yes”	”yes”	”yes”	”yes”
	”yes”	”yes”	”yes”	”yes”	”no”	”yes”	”yes”	”yes”	”yes”	”yes”
	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”
	”no”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”
”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”no”	”yes”	”yes”	”yes”	”yes”
”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”
”yes”		”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”yes”	”no”	”yes”
...	...	...	...	...	...	...	...	...	...	...

Using the data as generated above, we carry out the EM learning procedure and monitor the trend of the likelihood under current estimate. If two consecutive likelihood estimates only differ in a negligible way, the learning process stops. Figure 5.11 shows the trend of the likelihood and the accumulated gross error in the learning process, where the likelihood is

$$\log P_\theta(D) = \sum_{l=1}^{L} \log P_\theta(D_l),$$

and the gross error is

$$\sum_i \sum_k \sum_j |\hat{\theta}_{ijk} - \theta_{ijk}|.$$

We see that the likelihood for **both** the training data and the testing data monotonously increase as the learning process gets along. This is a sharp distinction with other learning methods, for example multi-layer perceptron (MLP) type neural networks [100], where the so-called over-fitting phenomenon exists. Typically, in these methods, when the performance of the learning process on the training data improves to some certain point, the performance on the test testing data begins to deteriorate. So in order to make the learning entity to both fits well the training data and generalize well on the testing data, the overall performance measure should take into account those on both training and testing data. In our case, we are happy to see that such over-fitting does not exist. Also, the gross error monotonously decreases as learning proceeds.

The learned conditional probability distributions are shown in figure 5.12, 5.13, 5.14, 5.15. By comparing with figure 5.3, 5.4, 5.5, and 5.6, we could see that the learned results are relatively close to the true values.

5.5 Conclusions

In this chapter, we addressed the problem of parameter learning for belief networks with fixed structure. Both complete and incomplete (data) observations were included. Given complete data, we describe the simple problem of single parameter learning for intuition and then expand to belief networks under appropriate system decomposition. If the observations are incomplete, we first estimate the missing observations and treat them as though they are "real" observations, based on which the parameter learning can be executed as in complete data case. We derived a uniform algorithm based on this idea for incomplete data case. Further, we studied the rate of convergence via the derivation of Jacobian matrices of our algorithm and provided a guideline for choosing step size. Our simulation results show that the learned values are relatively close to the true values. It is further observed that, in terms of fault diagnosis[1], the *true* and *learned* belief networks would give the same test sequences and average cost under most of the symptom patterns, and we conclude that such diagnostic belief network models are not sensitive to the parameters.

The above algorithm is derived based on batch data and the updating procedure is assumed to be executed in a batch mode. This is appropriate for many cases where there are batch files available for off-line training. However, our algorithm is not limited to batch mode only. By setting $L = 1$ in formula (5.34), we obtain the on-line version of the H operator

$$H(\psi)(ijk) = \frac{P_\psi(x_i^j, \pi_i^k | D_l)}{P_\psi(\pi_i^k | D_l)}.$$

Surely, such updating only occurs for those cases when $P_\psi(\pi_i^k | D_l) > 0$.

[1]to be discussed in the next chapter

In fact, our algorithm can be used in both batch and on-line mode for a real application. Batch mode learning helps tune the concerned belief model to the bulk of domain data, e.g. log files of related events. On-line mode helps fine-tune the parameters as new data becomes available.

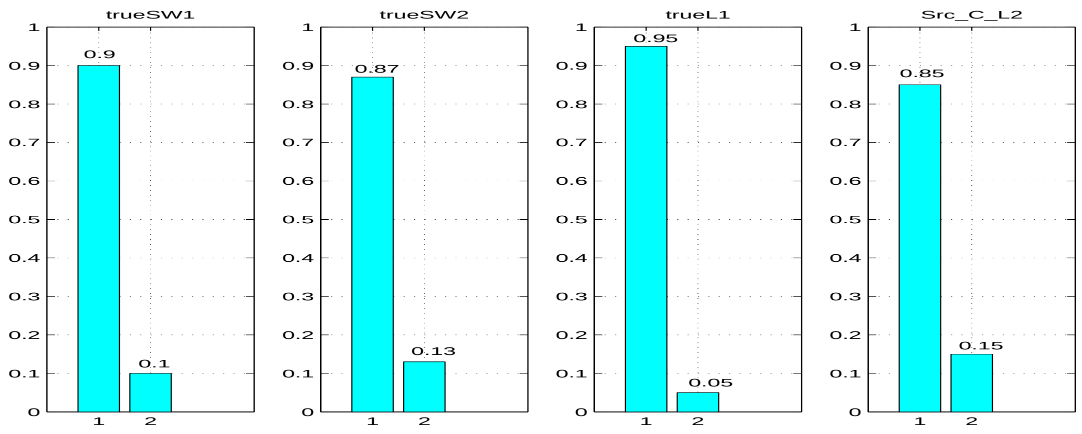

Figure 5.3: Root Nodes Conditional Probability Distributions

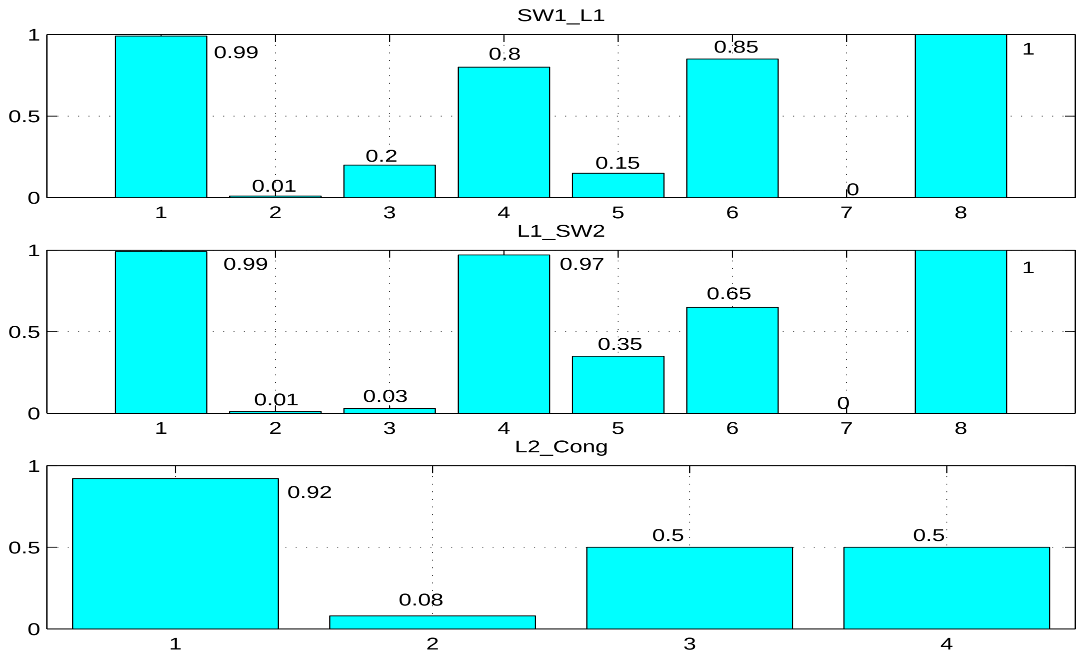

Figure 5.4: Hidden Nodes Conditional Probability Distributions

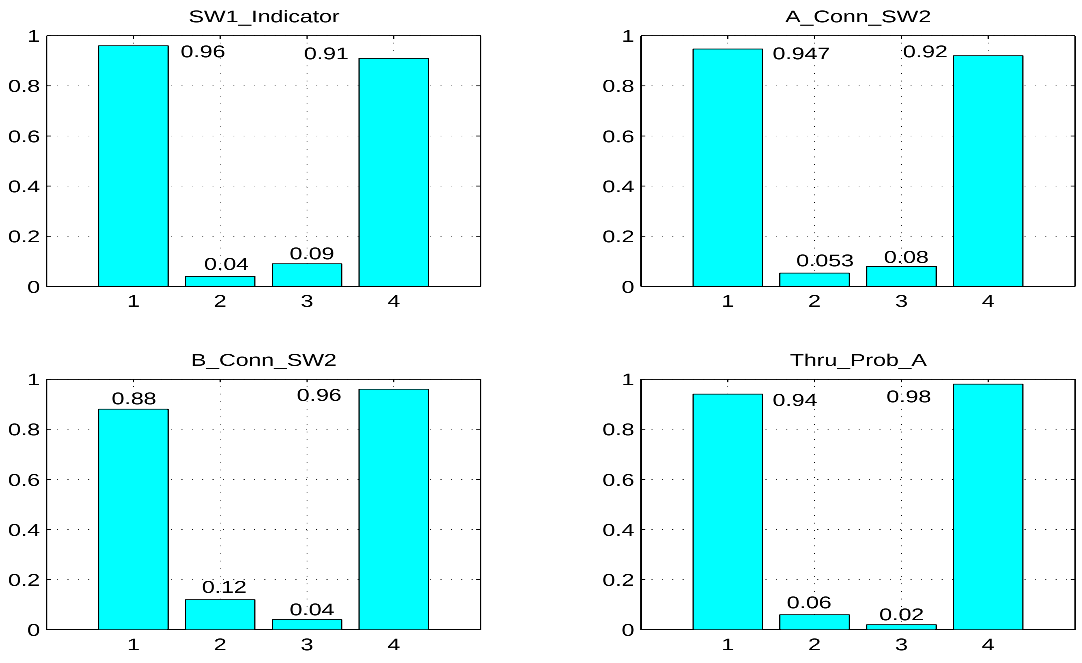

Figure 5.5: Leaf Nodes Conditional Probability Distributions–1

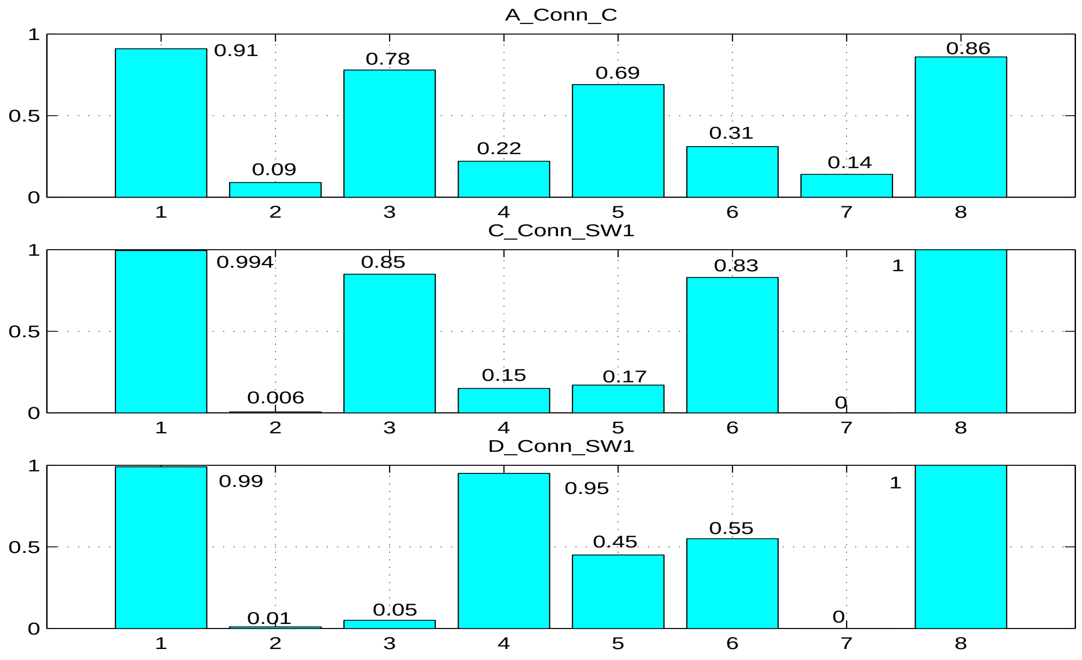

Figure 5.6: Leaf Nodes Conditional Probability Distributions–2

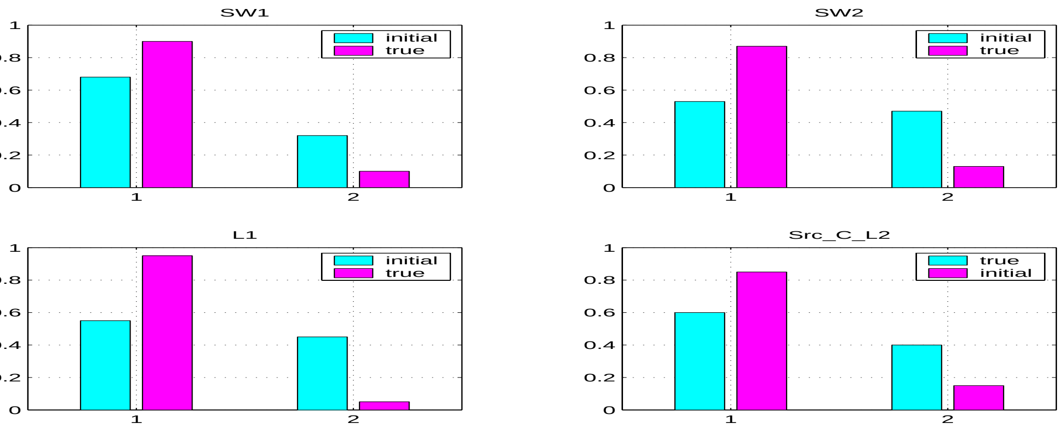

Figure 5.7: Initial Root Nodes Conditional Probability Distributions

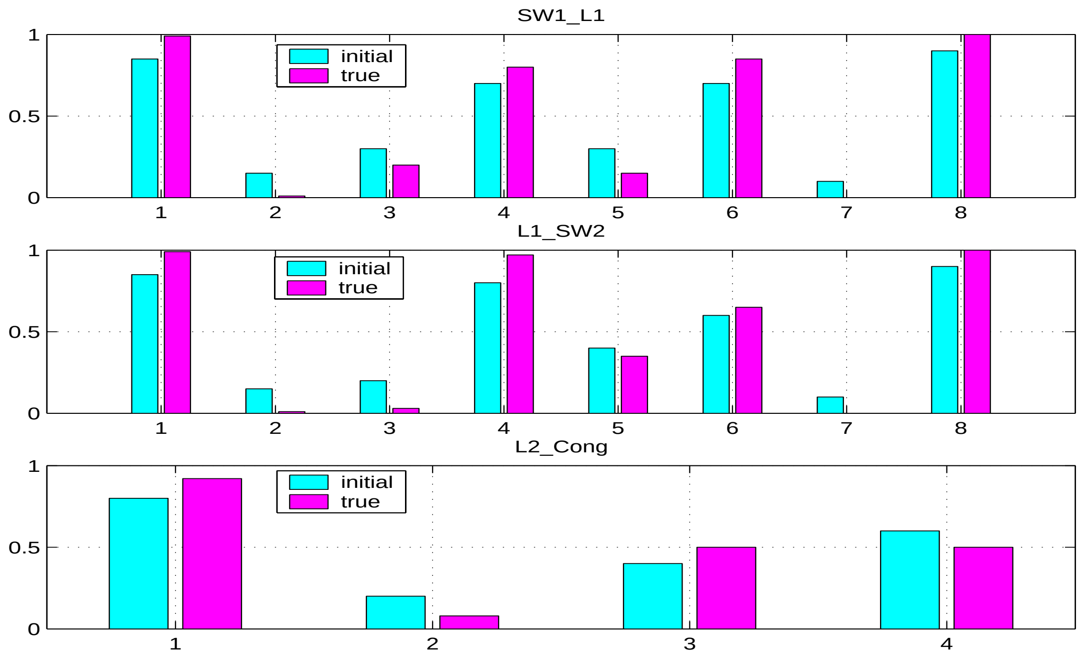

Figure 5.8: Initial Hidden Nodes Conditional Probability Distributions

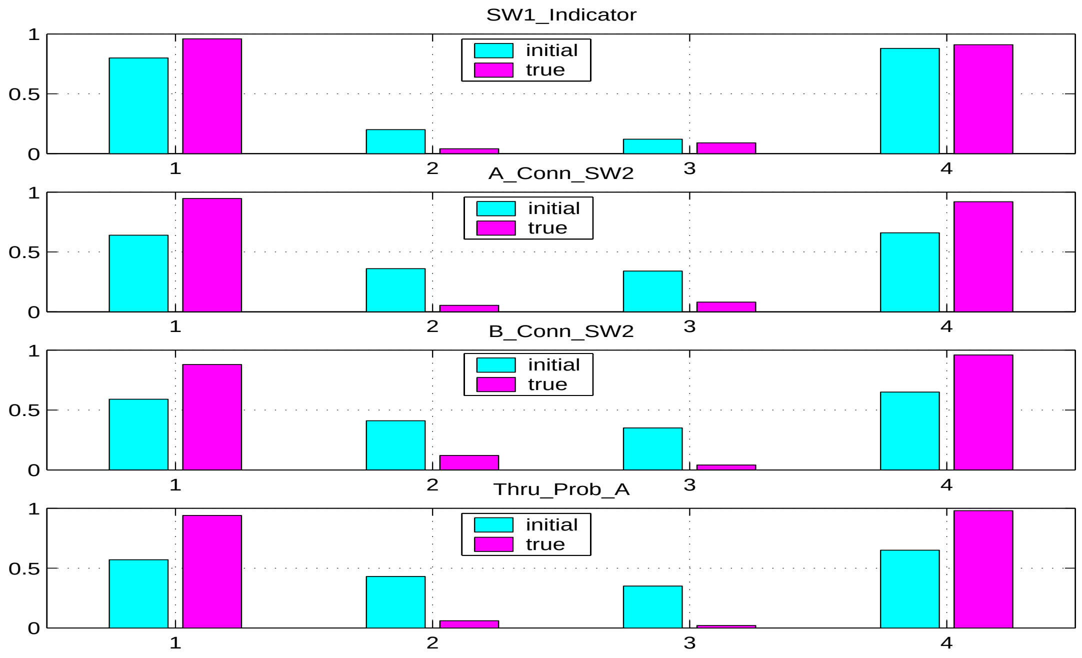

Figure 5.9: Initial Leaf Nodes Conditional Probability Distributions–1

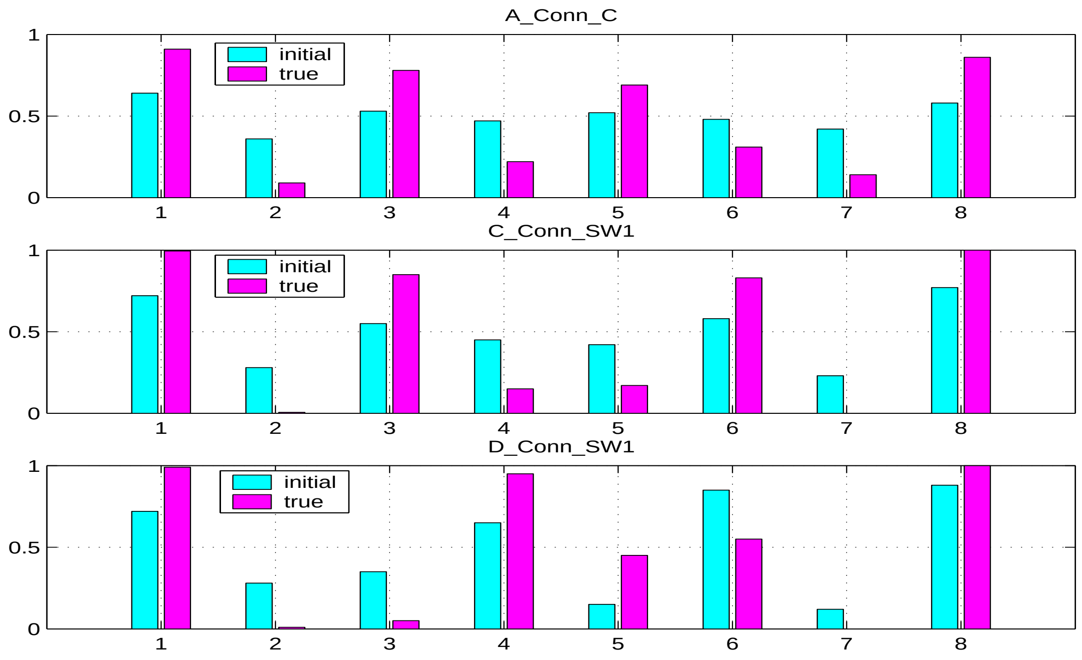

Figure 5.10: Initial Leaf Nodes Conditional Probability Distributions–2

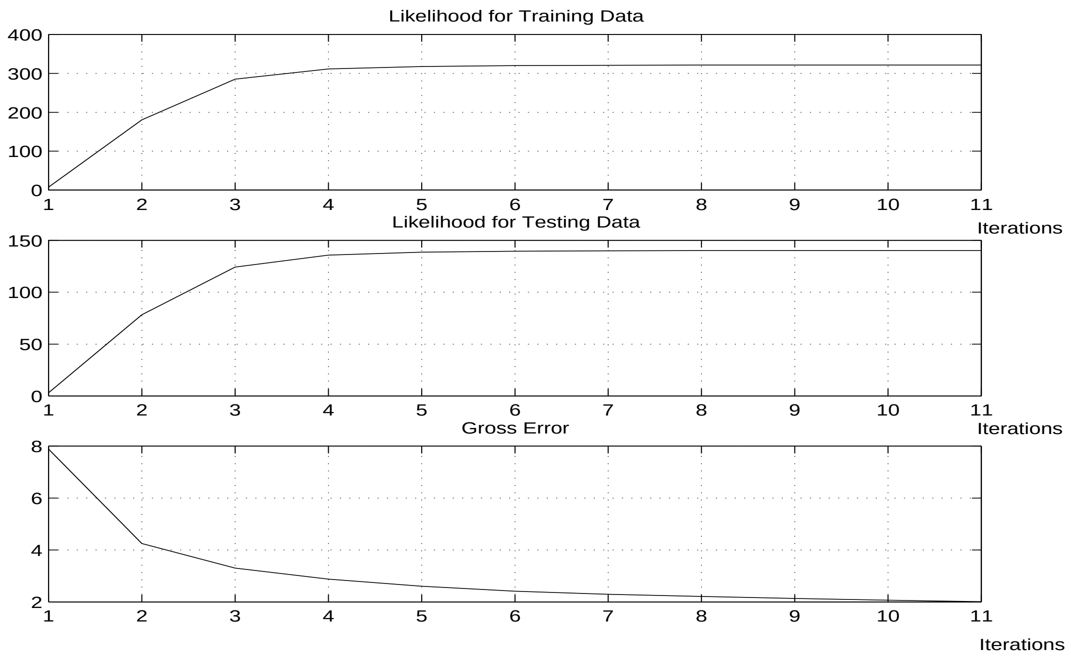

Figure 5.11: Trend of Likelihood and Gross Error during Learning Process

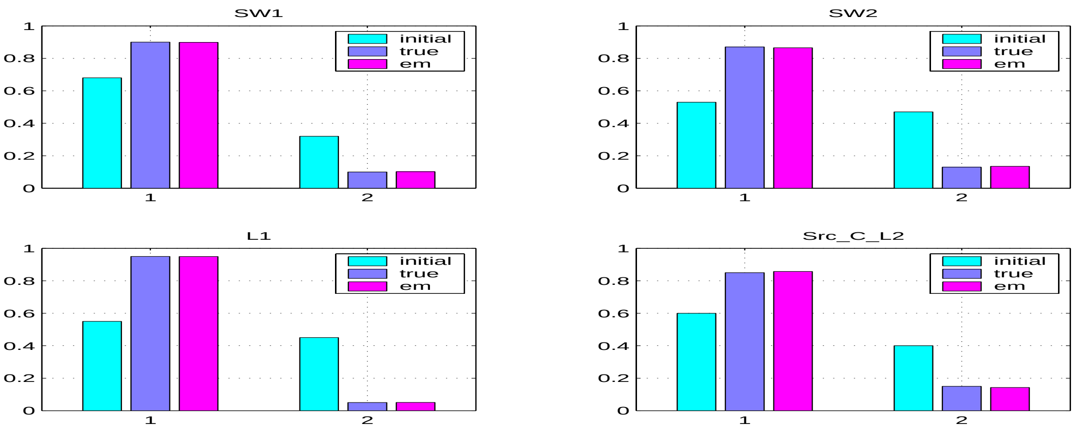

Figure 5.12: Learned Root Nodes Conditional Probability Distributions

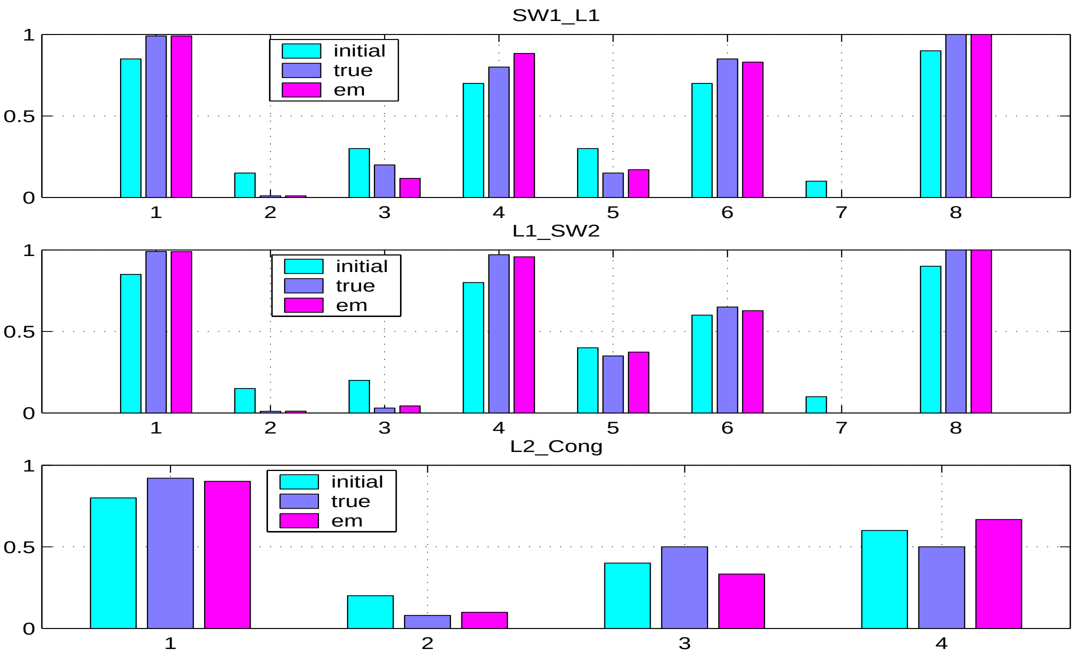

Figure 5.13: Learned Hidden Nodes Conditional Probability Distributions

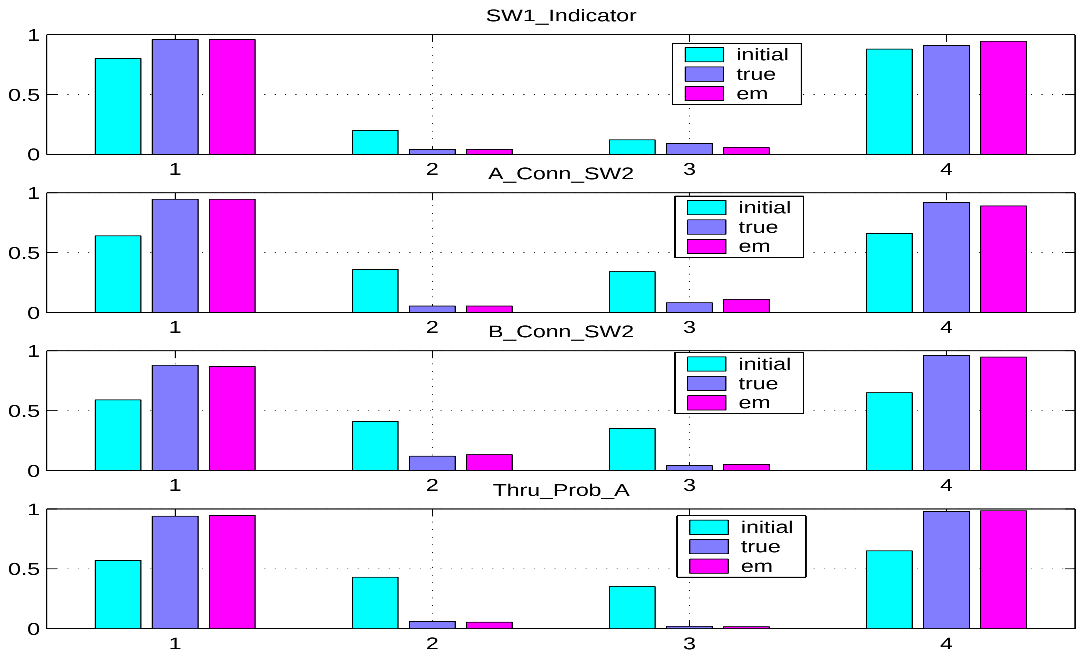

Figure 5.14: Learned Leaf Nodes Conditional Probability Distributions–1

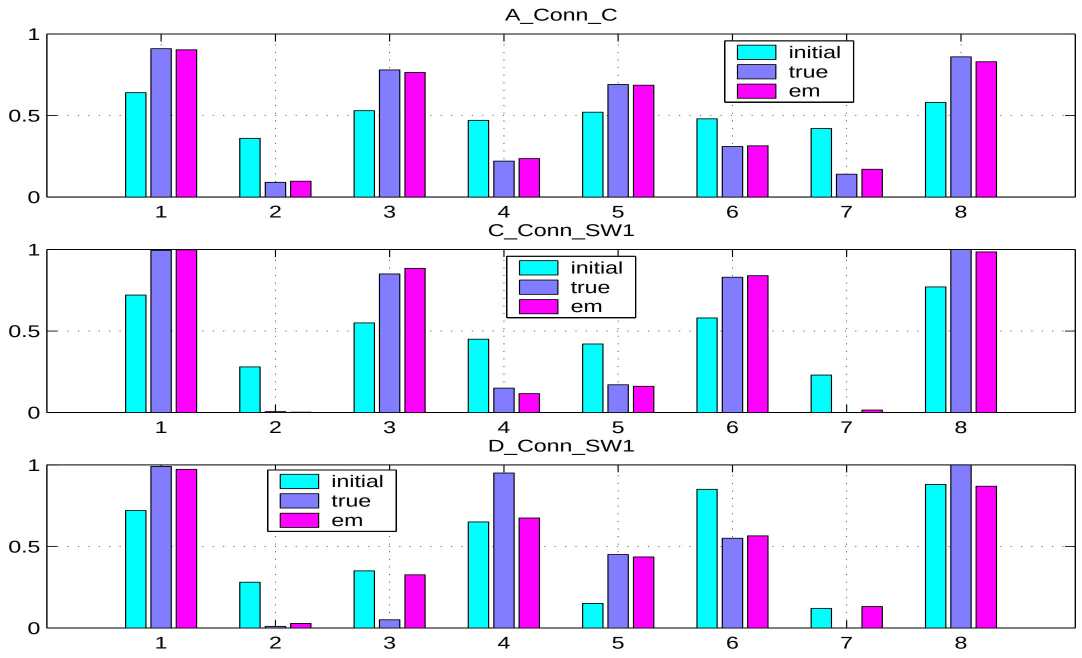

Figure 5.15: Learned Leaf Nodes Conditional Probability Distributions–2

Chapter 6

Fault Diagnosis Problems and Solutions using Belief Networks

6.1 An Intuitive Example

Suppose we are handling the problem call failure and identify the possible causes as follows: Server, link and switch may fail, and there might be heavy traffic that causes the network congestion. Luckily, we have access to the alarms associated with link failure and switch failure. This scenario is modeled as a belief network, as shown in figure 6.1. Each node takes binary value and the table associated with it represents the conditional probability distribution, given its parent nodes' instantiations. Without any observations, the initial marginal probabilities of each node are shown in figure 6.2.

Now suppose we observe that there are call failures. We wish to infer the most probable cause for this symptom from the belief network model. To do this, we input this evidence, execute the belief propagation, and obtain the updated beliefs of each non-evidential node, as shown in figure 6.3. Note that for each candidate fault node, namely Link Failure, Server Failure, Heavy Traffic and Switch

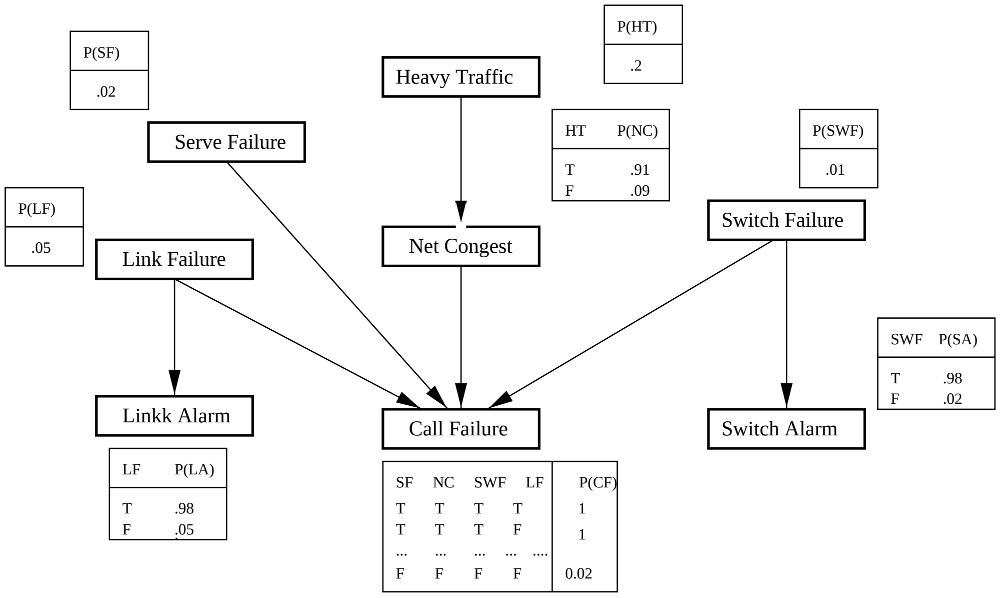

Figure 6.1: An example Belief Network

Failure, the probability of being faulty is increased. If we also observe link alarms, then we hope that this extra information could help locate the most probable fault, see figure 6.4. As we would have expected, the evidence of link alarms distinguishes Link Failure as the most probable fault, which also "explains away" other possible candidates in the sense that their updated beliefs are decreased, as compared with those in figure 6.3.

The above example shows the sketch of doing diagnosis using belief networks: obtain evidence, update beliefs, obtain evidence again, and so on. This is *active* diagnosis in that we are seeking more information on the fly during diagnosis. We will discuss this process in more details next.

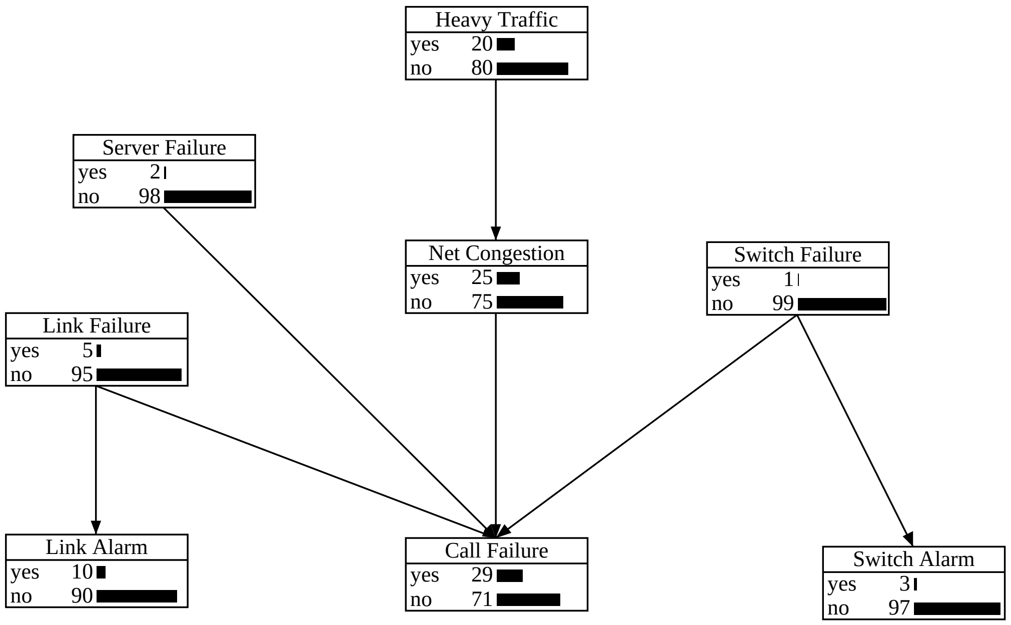

Figure 6.2: Initial marginal probabilities

6.2 Problem Definition

In communication networks, probes are attached to some hardware/software components to get operation status. Typically the raw data returned from the probes will be grouped into vector form $\mathbf{d} \in \mathbf{R^n}$ and then processed to get some aggregated values (e.g. average, peak value, etc.). A statistics is a function from $\mathbf{R^n}$ to $\mathbf{R}$ that maps the raw data vector $\mathbf{d}$ to a real number. Such statistics will usually be quantified and represented using discrete values. We use 0 to represent normal status, and other positive integers to represent abnormal status with different level of severity. A node v in a belief network model $\mathcal{B}=(V, L, P)$ is called observable if and only if it represents the health status of a statistics, or corresponds to a user report. The set of observable nodes is denoted by O. The non-observable set is simply $\tilde{O} = V \setminus O$. We restrict these observable nodes to be leaf nodes only, and vice versa. The regular evidence set R contains those nodes

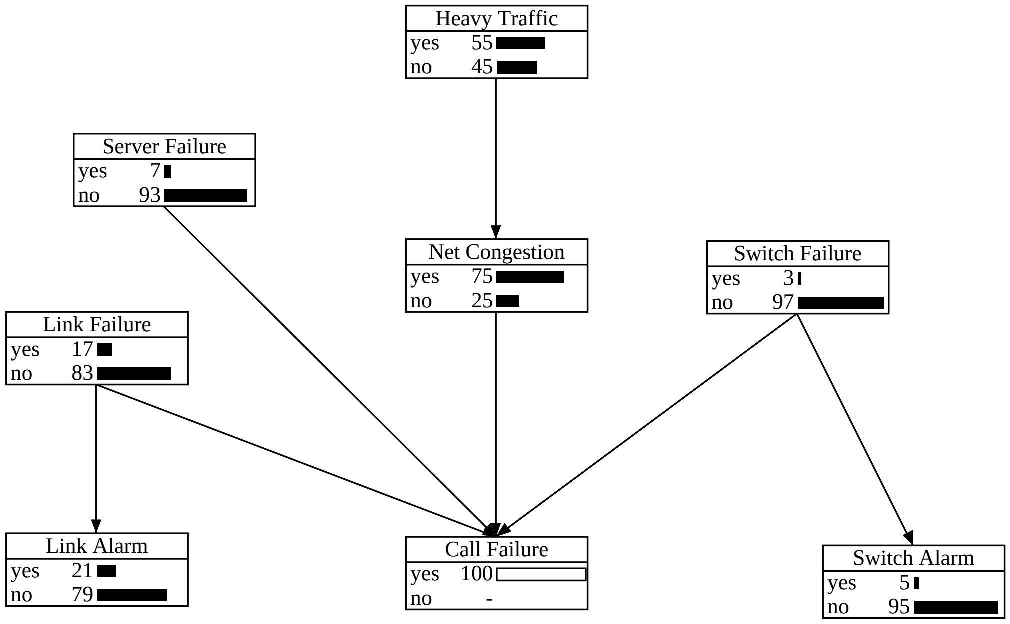

Figure 6.3: Updated beliefs after observing Call Failure

that we observe during regular network monitoring operations. Each $r \in R$ is called a symptom node. The test set ST contains all other observable nodes that are not currently in R, namely $ST = O \setminus R$. The fault set F is the set of root nodes, and they are not observable, $F \subseteq \tilde{O}$. We restrict that all root nodes are binary valued. The hidden node set H contains all nodes in $\tilde{O}$ but not in fault set F, $H = \tilde{O} \setminus F$. Hidden nodes are intermediate nodes between faults and symptoms and we don't usually put queries on them during diagnosis.

Within a certain time window, the problem domain is said to be working in normal status with respect to regular evidence set R if and only if every node in R takes value 0, or vector $\mathbf{r} = \mathbf{0}$, where $\mathbf{r} = (r_1, r_2, \ldots, r_{|R|})$. The problem domain is said to be working in abnormal status with respect to regular evidence set R if and only if there is at least one $r \in R$ whose value is other than 0. There might be cases when multiple symptom nodes in R take nonzero values. The

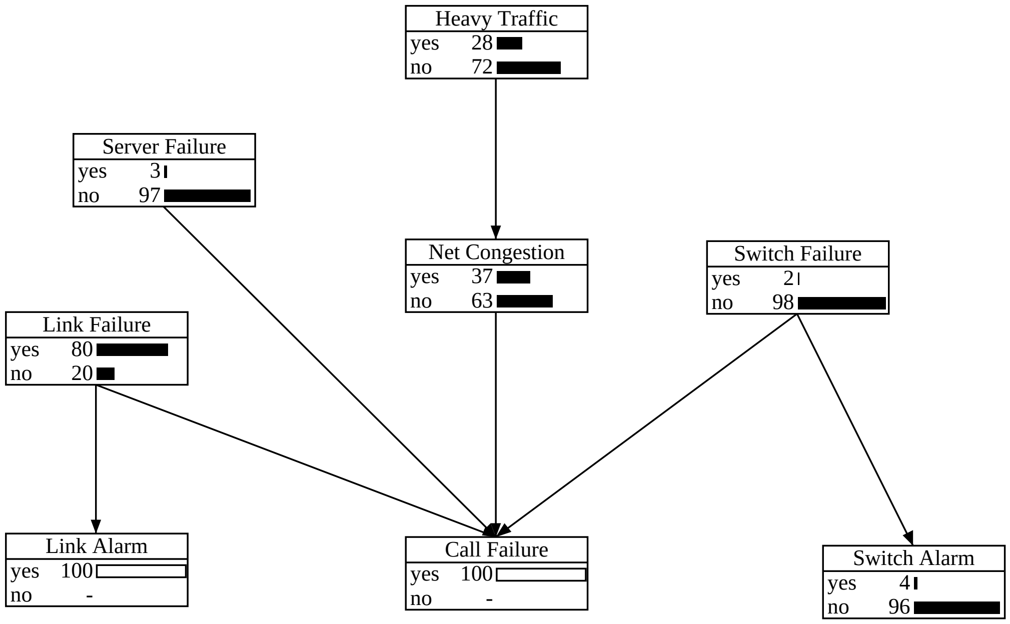

Figure 6.4: Updated beliefs after observing Call Failure and Link Alarm

syndrome with respect to regular evidence set R is simply the nonzero vector $\mathbf{r}$. Any syndrome can trigger the diagnosis process.

After fault diagnosis is triggered, the initial evidence is propagated and the posterior probability of any $f \in F$ being faulty can be calculated. It would be ideal if we can locate the fault with efforts up to this. But most of the time, similar to what happens in medical diagnosis, we need more information to help pinpoint the fault. So naturally, we identify two important problems associated with belief network based fault diagnosis: When can I say that I get the right diagnosis and stop? If right diagnosis has not been obtained yet, which test should I choose next? We address these two problems in the next sections. In our work, we postulate that all the observations and tests are constrained within the belief network model.

6.3 Right Diagnosis via Intervention

Consider what a human usually think during diagnosis. After obtaining one possible reason, one may naturally ask, for example, "Will the problematic circuit work normally if I replace this suspicious component with a good one?" He/she then goes ahead and sees what will happen after the replacement. If the syndrome disappears, one can claim that he/she actually found and trouble-shooted the fault. If the problem domain is tiny, not very complex, and the replacement burden is light, this paradigm will work well. But for communication networks, the story is totally different. We would like to do intelligent diagnosis via *computation*, rather than brutal replacement before we are very confident what the fault is.

To do this, we need to distinguish between two kinds of semantics for the instantiation of a node in a belief network: passive observation and active setting. All the instantiations of nodes we have talked about so far are passive observations, and we would like to know the consequences of, and the possible causes for such observations. The alternative semantics is that we can also *set* the value of a node via active experiment. One example is the above question, where external reasons (the human diagnoser) explain why the suspicious component becomes good and thus all the parent nodes for this node should not count as causes during belief updating. Other belief updating like evaluating consequences, however, are not influenced by this active setting. This external force is called *intervention* in [101].

With this *set* semantics, we could do virtual replacement in our belief network model. For simplicity, we assume here that the single symptom node is $S1$. For each node in F, we could get its posterior probability of being faulty given $S1 = 1$. Let $f = argmax_{g \in F} P(g = 1|S1 = 1)$, and we would evaluate $P(S1 = 0|setting(f = 0))$. Other nodes in F are treated as background variables and they keep at the same status as what has just been updated. In our work, we introduce the so-called intervention belief network to help this virtual replacement.

Definition 6.3.1 *An intervention belief network $\widetilde{\mathcal{B}}= (V, L, P, S, Fs)$ is obtained from the original belief network $\mathcal{B}= (V, L, P)$ with the same V, L, P. S is the symptom set and $Fs \in F$ is the set of suspicious nodes. We compute for each $s \in S$ the probability $P(s = 0|setting(Fs = \mathbf{0}))$ using $\widetilde{\mathcal{B}}$.*

For our particular example above, the virtual replacement procedure is as follows. First, in $\mathcal{B}= (V, L, P)$, update for each node $f_i \in F$ the probability $p_i \overset{\triangle}{=} P(f_i = 1|S1 = 1)$. Suppose $f_1 = argmax_{g \in F} P(g = 1|S1 = 1)$. Then in intervention belief network $\widetilde{\mathcal{B}} = (V, L, P, S1, f_1)$, set node $f_1 = 0$, and with $P(f_i = 1) = p_i$, $i = 2, \cdots, |F|$, compute $P(S1 = 0|setting(f_1 = 0))$. See figure 6.5 for an illustration, where we only have two root nodes F and G. F corresponds to f_1 here.

To determine whether or not this virtual replacement has led $S1$ to an acceptable status, we need a reference value for the computed $P(S1 = 0|setting(f_1 = 0))$ to compare with. Without any evidence input, the belief network model $\mathcal{B}$ itself gives the marginal probability of each leaf node to be normal. We use these values as the reference in our work.

Definition 6.3.2 *Given a small number ϵ, we say that node $S1$ becomes ϵ - normal via intervention on f_1 if and only if $P(S1 = 0) - P(S1 = 0|setting(f_1 = 0)) < \epsilon$.*

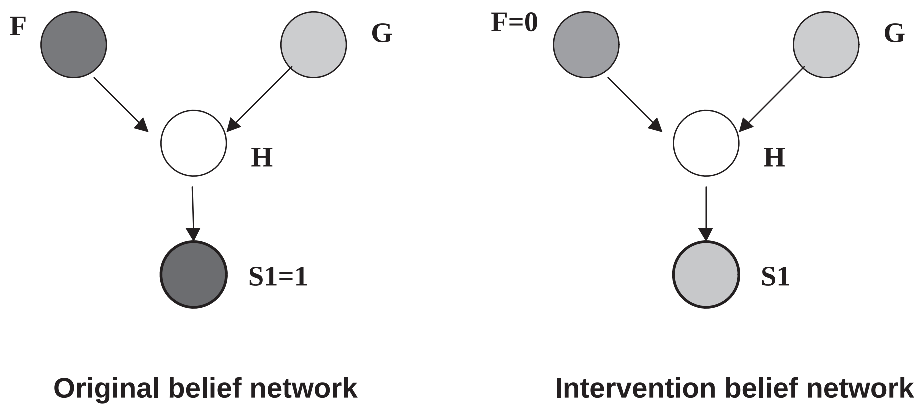

Figure 6.5: Example of An Intervention Network

Note that during diagnosis process, some of the testing nodes chosen may already manifested themselves as values other than "normal". These nodes should also be included in intervention network $\widetilde{\mathcal{B}}$.

Definition 6.3.3 *A nonempty set of suspicious nodes Fs is called the explanation or right diagnosis if and only if every node in set S, including both initial and newly-found symptoms, becomes ϵ-normal if we set every node in Fs to normal in the intervention belief network $\widetilde{\mathcal{B}} = (V, L, P, S, Fs)$. It is when Fs explains the set S that we terminate the diagnosis process.*

6.4 Decision Theoretic Fault Diagnosis Strategies

We formulate the test selection procedure as a partially observable Markov decision processes (POMDP) problem with optimal stopping. At each decision epoch, we could either choose a node to test or stop there. Test is rarely free, and termination incurs some costs. The goal is to find a good test sequence and the right time to stop. We will show that by choosing termination cost appropriately, the optimal stopping rule matches our notion of right diagnosis.

6.4.1 POMDP Formulation

State Space $\mathcal{S}$

The state is the status of the root nodes $F = \{F_1, \ldots, F_{|F|}\}$, and for a particular $s \in \mathcal{S} = 2^{|F|}$, $s = \{f_1, \ldots, f_{|F|}\}$. We use S_k to denote the state at time k. In our diagnosis case, the current state, which is unobservable, does not change regardless what tests will be chosen. The goal of diagnosis is to *identify* this state by using initial symptoms and subsequent test results. So here we have

$$P(S_{k+1}|S_k) = \begin{cases} 1 & \text{if } S_{k+1} = S_k \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

History Process

If we choose one test per decision epoch, the time step set is defined as $N = \{1, 2, \ldots, |ST|\}$. The active evidence set AE contains the nodes that are instantiated during the process of diagnosis. Initially $AE = R$ and it expands as more test nodes in ST are added into it. Nodes in AE are not to be considered for future use. The candidate test set C_{st} contains the nodes in ST that are available to be chosen and tested. Initially $C_{st} = ST$ and it shrinks as instantiated nodes are removed from it. The action set $A = C_{st} \cup \{STOP\}$. Let Z_{a_t} denote the value obtained by observing a_t, and we define the history process up to time k as $I_k = (Z_0, (a_1, Z_{a_1}), \ldots, (a_k, Z_{a_k}))$, where $Z_0 = \Big((r_1, Z_{r_1}), \ldots, (r_{|R|}, Z_{r_{|R|}})\Big)$ represents the regular evidence set and corresponding instantiations. I_k grows with diagnosis and obviously, $I_k = (I_{k-1}, (a_k, Z_{a_k}))$, the Markov property. We can simply take I_k as the *state* at time k and obtain a completely observable Markov decision problem. But the growing state process makes this approach impractical.

Belief / Information State

Given I_k, we define $b_k = P(\mathbf{F}|I_k)$ as the probability distribution of states in $\mathcal{S}$. It is proven that b_k is a sufficient statistics that contains all information embedded in the history process for control, and we call it belief or information state [102, 103]. Using Bayes rule, we can easily verify that the process $\{b_k\}$ is also Markov. If we choose b_k as the state at time k, we avoid the growth of the state space; but now, the state space is *continuous,* and we call it $\mathcal{B}_c$. In our case, if we are given $I_k, a_k, \text{and}\, Z_{a_k}$, the next belief state b_{k+1} is uniquely determined via belief network propagation, and we define $\Psi(b_k, a_k, Z_{a_k}) \triangleq Pr(b_{k+1}|b_k, a_k, Z_{a_k})$. If we let $\mathcal{X} = \mathcal{B}_c \cup \{T\}$ and x_k be the state at time k, then the augmented states evolve according to

$$x_{k+1} = \begin{cases} \Psi(x_k, a_k, Z_{a_k}) & \text{if}\, x_k \neq T \,\text{and}\, a_k \neq STOP \\ T & \text{if}\, x_k = T \,\text{or}\, (x_k \neq T \,\text{and}\, a_k = STOP) \end{cases} \tag{6.2}$$

The observation model for $a_k \neq STOP$ is $P(Z_{a_k}|I_k, a_k) = Pr(a_k = Z_{a_k}|I_k)$.

Choosing Suspicious Nodes

After we obtain x_k, it will not suffice to give out this probability distribution directly as the result. What is needed is the explanation. To see if we could obtain the explanation as defined above, we need to extract from x_k the suspicious nodes. However, it is always an important issue to determine how many suspicious nodes we should choose from the fault set F. In our belief network model and the parallel intervention model, we should be discreet in choosing multiple nodes. If we simply choose all nodes in F and do the intervention, the symptom nodes will all become ϵ-normal for sure. But clearly, calling every node in F as faulty is not acceptable; One of the most important aspects of fault diagnosis in general is to bias among the many possible faults and locate the real one(s)! In our work, we tried two schemes.

In the first scheme, we compute for each node in F the probability of being faulty given I_k, and sort them in a descending order, say $p_1 \geq p_2 \geq \cdots \geq p_{|F|}$. We only choose the first j nodes such that $\sum_{k=1}^{j} p_k / \sum_{k=1}^{|F|} p_k \geq \eta$, where $\eta \in (0,1)$. It should not be close to 1, since in that case, we would have to choose almost all nodes in F. In our work, we choose 0.4 initially. If we could not find the right diagnosis, we increase η by a small amount so that more root nodes could emerge as the candidates. By doing this, we are not limiting ourselves to the single fault scenario. This scheme is intuitive, I_k suffices to provide information for each node in F, and it is not necessary to calculate x_k. However, it is very hard to choose a good η that works well without knowing in advance how many faults there might be.

The second scheme makes use of x_k. We first compute the belief state and get a table that contains the joint distribution of the root nodes given I_k. Then we choose the largest entry from the table and mark the index of the entry. The suspicious nodes are obtained from the index. For example, if we only have four root nodes and the binary string corresponding to the index of the largest entry is 0101, then the second and fourth nodes are chosen. In this scheme, there is no need to find a good η, and it adapts to multiple causes easily. The drawback is that extra storage space is needed for the joint distribution table. If the number of root nodes is small, this is a preferable scheme.

Cost Structure

There is an immediate cost associated with each $s_i \in ST$. The cost function $C(s_i, t)$ entails careful deliberation about many factors like the difficulty and time to be consumed for the test, etc. Here we assume that the cost function is of form $C(s_i)$. This is usually the case in that the cost is normally associated with the test itself only, and the test itself does not usually change with time. Also, we wish to diagnose promptly and we penalize on diagnosis steps. If $a_k = STOP$ at time k, no penalty. Otherwise, we penalize this extra step using function $g(k)$. Here, we simply take $g(k) = 1$ for all k. At time k with state $x_k \neq T$, if we choose $a_k = STOP$, we incur $t(x_k)$ as the termination cost. Note that $t(T) = 0$. Given $x_k \neq T$ and suspicious node set Fs, we compute $t(x_k)$ as follows. First, in original belie network, let $K = F \setminus F_s$ and compute for each node in K the probability of being faulty as $q_i \overset{\triangle}{=} Pr(K_i = 1|I_k)$. Second, in intervention network, set the root nodes that correspond to those in K with the same probabilities as those in $\{q_i\}$, and set the root nodes that correspond to those in F_s to state "normal". Finally, in intervention network for each node S_i in the active symptom set S, and for some given small ϵ, define $\Delta = P(S_i = 0) - P(S_i = 0|\text{Setting root nodes as above})$. If $\Delta < \epsilon$, $t_{S_i}(x_k) = 0$, else $t_{S_i}(x_k) = CONST\,[\Delta - \epsilon\,]$, where $CONST$ is a constant to make $t_{S_i}(x_k)$ large. The total cost is $t(x_k) = \sum_{S_i \in S} t_{S_i}(x_k)$. So, the immediate cost of choosing action a_k at time k with state $x_k \neq T$ is

$$g_k(x_k, a_k) = \begin{cases} c(a_k) + g(k) & \text{if } a_k \neq STOP \\ t(x_k) & \text{otherwise} \end{cases} \tag{6.3}$$

At the last step N, the terminal cost $g_N(x_N)$ is defined as

$$g_N(x_N) = \begin{cases} t(x_N) & \text{if } x_N \neq T \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

Note that both $g_k(x_k, a_k)$ and $g_N(x_N)$ are deterministic functions. Now we have the finite horizon problem

$$\min_{a_k, k=0,\ldots,N-1} \left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, a_k) \right\}. \tag{6.5}$$

6.4.2 Solution for the Problem

Define $J_k(x_k)$ as the cost-to-go at state x_k and time k [103]. At termination state T, $J_k(T) = 0, \forall k = 0, \ldots, N-1$. For $x_k \neq T$, we have the dynamic programming algorithm:

$$J_N(x_N) = g_N(x_N) \tag{6.6}$$

$$J_k(x_k) = \min \left[t(x_k), \min_{a_k \in A_k} \left[c(a_k) + g(k) + \sum_j P(a_k = j | I_k) J_{k+1}(\Psi(x_k, a_k, Z_{a_k} = j)) \right] \right] \tag{6.7}$$

So the optimal stopping policy is: Choose STOP if

$$t(x_k) \leq \min_{a_k \in A_k} \left[c(a_k) + g(k) + \sum_j P(a_k = j | I_k) J_{k+1}(\Psi(x_k, a_k, Z_{a_k} = j)) \right], \tag{6.8}$$

at current state x_k and time k. If we choose $t(x_k)$, as shown above, such that $t(x_k) = 0$ in the case of right diagnosis and let $t(x_k)$ be very large otherwise, then the optimal stopping policy is: STOP if and only if we obtain the right diagnosis. Now let us look at the test selection strategies.

To solve the problem (6.5) using the dynamic programming update (6.7), the continuous state space is the major obstacle. It would be very desirable if we could find some structures for the value function or optimal policy. In one class of problems [104], the optimal value function for the finite horizon problem is piecewise linear and concave. Thus we could represent the value functions using a set of discrete vectors and avoid the direct handling of the continuous space. Unfortunately, we don't have this good property in our problem, and we need to seek to approximate methods. As discussed above, we need to extract from state $x_k \neq T$ the suspicious node set F_s. We ignore those root nodes that are not very fault-prone and this is our first approximation. Now, given that F_s does not explain the current active symptoms, we need some heuristics to help choose the next test. Let us begin with a simpler problem for intuition.

Suppose the concern here is to locate the single faulty component. There are symptoms indicating the malfunction (e.g. car doesn't start) and for each possible faulty component there is a *direct* test associated with it. The cost for testing component i is c_i. Based on the symptoms, we obtain P_i, the probability that component i is in failure, for every component. We are supposed to test those components one at a time. As soon as one component fails its associated test, we claim that we find the single fault and stop. By interchange argument [103], it is easy to see that in an optimal strategy, all elements must be in non-decreasing sequence of c/P values, see also [105].

Our problem is different from this scenario in the following aspects. It tackles failures while our problem integrates both hard and soft faults. It assumes the existence of direct test while we don't have that luxury. For a communication network environment which is distributed, complex and heterogeneous, it is impossible to predefine and store a direct test for each possible cause. Actually one of the goals here is to *generate* dynamically the test sequence on the fly. In our setup, right diagnosis is determined through computation, rather than brutal replacement. Finally, our algorithm should be able to tackle multiple faults.

But the c/P algorithm does provide insight in that it reflects the following observation: in order to minimize the total cost, people are more likely to try those more fault-prone, cheaper components before the less-probable, expensive ones. In our diagnosis algorithm, we wish to find an appropriate test node st if Fs could not explain the active symptom set S. In particular, we would like to choose the test node from candidate test set C_{st} that is cheapest and most relevant to Fs. To achieve this, we need a measure for relevance between a test node in C_{st} and a fault node in F_s.

Definition 6.4.1 *Given I_k, the relevance of random variable Y relative to random variable X is defined as*

$$R(X;Y|I_k) = \frac{I(X;Y|I_k)}{H(X|I_k)}.$$

$H(X|I_k) = -\sum_{x\in\mathcal{X}} p(x|I_k)\log p(x|I_k)$ is the conditional entropy of a random variable X, $I(X;Y|I_k) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y|I_k)\log\frac{p(x,y|I_k)}{p(x|I_k)p(y|I_k)}$ is the conditional mutual information between random variable X and Y [94]. $R(X;Y|I_k)\in[0,1]$ indicates to what extent Y can provide information about X. $R(X;Y|I_k) = 1$ means that Y can uniquely determine X, while $R(X;Y|I_k) = 0$ indicates that Y and X are independent, given current I_k. Note that $R(X;Y|I_k) \neq R(Y;X|I_k)$. More generally,

Definition 6.4.2 *Given I_k, the relevance of random variable Y relative to a set of random variables $\mathbf{X}$ is*

$$R(\mathbf{X};Y|I_k) = \frac{I(\mathbf{X};Y|I_k)}{H(\mathbf{X}|I_k)},$$

where $H(\mathbf{X}|I_k)$ and $I(\mathbf{X};Y|I_k)$ are defined similarly as above.

With the relevance measure, our next test node given I_k at time k is simply

$$st = argmax_{g \in C_{st}} R(\mathbf{Fs}; g)/c(g), \tag{6.9}$$

and our fault diagnosis process is summarized as algorithm 6.4.2, also shown in figure 6.6.

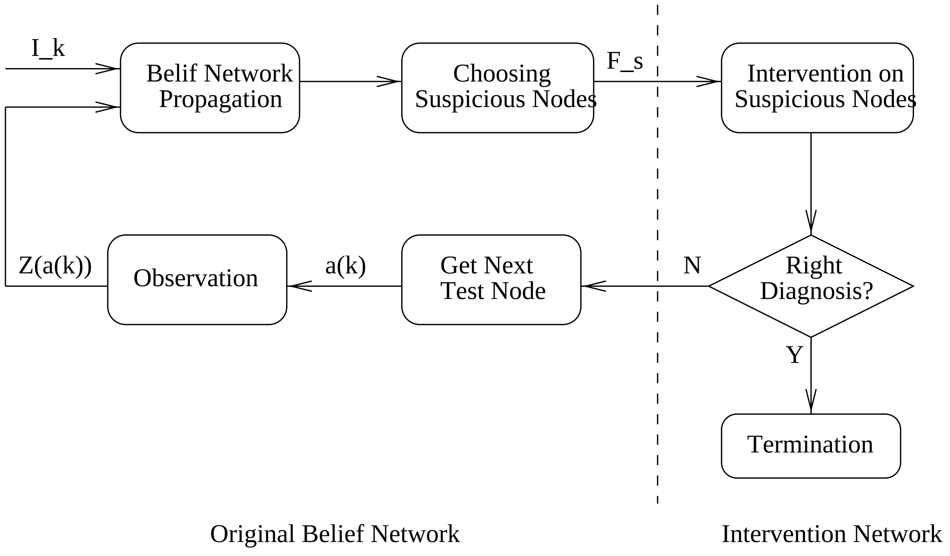

Figure 6.6: Illustration of the diagnosis process using intervention belief network

6.5 Simulation

To illustrate the effectiveness of our fault diagnosis algorithm, consider the example network in figure 6.7. Two switches $SW1$ and $SW2$ are connected via link $L1$. We have a probe a hooked at the end of $SW2$ to measure the traffic throughput going out of $SW2$. Suppose the information we could obtain during network operation include whether or not: $SW1$ alarm is normal, A could connect $SW2$, B could connect $SW2$, A could connect C, C could connect $SW1$,

Algorithm 1 Fault Diagnosis Procedure

- Step 1. Initialization
  - Set time step $tp = 0$, $AE = R$, $C_{st} = ST$.
  - Input evidence by setting the nodes in set AE according to current active values ae.
- Step 2. Belief Propagation in belief network $\mathcal{B}$ and get the set of suspicious nodes F_s according to scheme one or two.
- Step 3. Set the root nodes in $\widetilde{\mathcal{B}}= (V, L, P, S, Fs)$ accordingly, and execute the intervention. If Fs explains S, update total cost and TERMINATE.
- Step 4. Get next testing node
  - If $C_{st} = \Phi$, update total cost and give out the set F_s and say "Didn't find the right diagnosis, but here is the list of possible faults in decreasing order".
  - Else: Get node st according to (6.9).
- Step 5. Observing test node st and get observation Z_{st}
  - Input this evidence $st = Z_{st}$ to original belief network $\mathcal{B}$. Update tp, C_{st}, and AE.
  - Goto Step 2.

throughput at probe a is normal, and D could connect $SW1$. The possible faults are identified as: $SW1$ works normal or not, $L1$ normal or congested, $SW2$ normal or not, and source pumped from C to $L2$ is normal or not. We set up a belief network model for such situations, and figure 6.8 shows the structure and initial probability distributions.

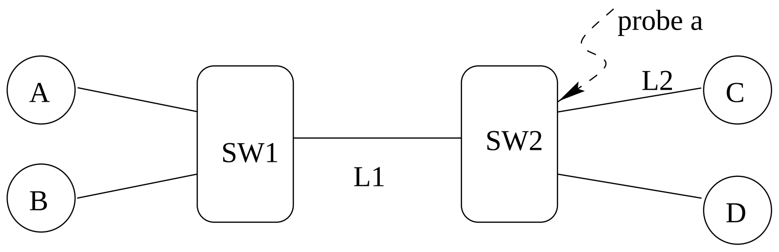

Figure 6.7: Example Network

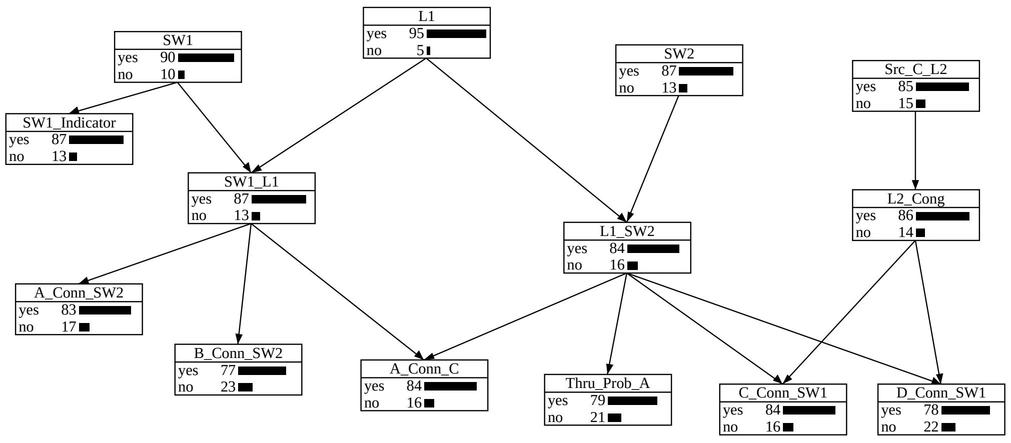

Figure 6.8: Belief Network for Example Network

Let us look at one diagnosis scenario. Suppose we observe that A_Conn_SW2 goes wrong, and we get the updated distribution as shown in figure 6.9. We see that $SW1$ is the suspicious node and the intervention result is $P(A\_Conn\_SW2 = yes|Intervention) = 0.78$. Initially, $P(A\_Conn\_SW2 = yes) = 0.83$, and we have not yet got the right diagnosis for $\epsilon = 0.4$. Based on our test selection scheme, node $SW1\_Indicator$ is chosen and the observation of it is "normal". The updated distribution is shown in figure 6.10. Again, $L1$ is intervened and no right diagnosis is obtained. The next node selected this time is A_Conn_C and the observation is "abnormal". We got the updated distribution again in figure 6.11. If we intervene node $L1$, we have $P(A\_Conn\_SW2 = yes|Intervention) = 0.87 > 0.83$, and we obtain the right diagnosis!

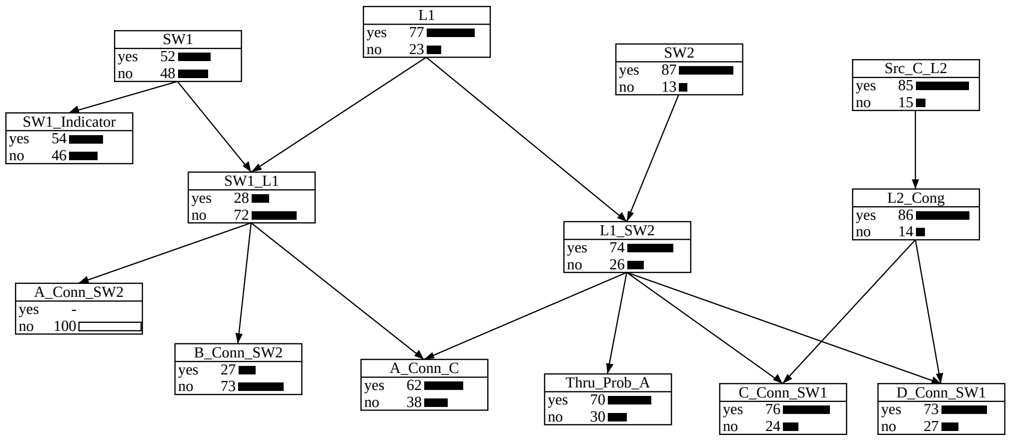

Figure 6.9: After A_Conn_SW2 Goes Wrong

As a comparison to our node selection scheme, we use the random scheme meaning that each time we need a test node, we simply choose one uniformly from all current available nodes in C_{st}. In our simulation, the outcome of chosen test node st is uniformly generated as either 0 or 1. The costs for testing each leaf node is shown in Table 6.1, with 40 as the penalty for not being able to find the right diagnosis. Table 6.2 shows for three scenarios the comparisons of the two test generation schemes with 2000 runs, which take only about 40 milliseconds per run for each scenario on a SUN Ultra2 running Solaris 8. We see that node selection via relevance is much better than that via random selection.

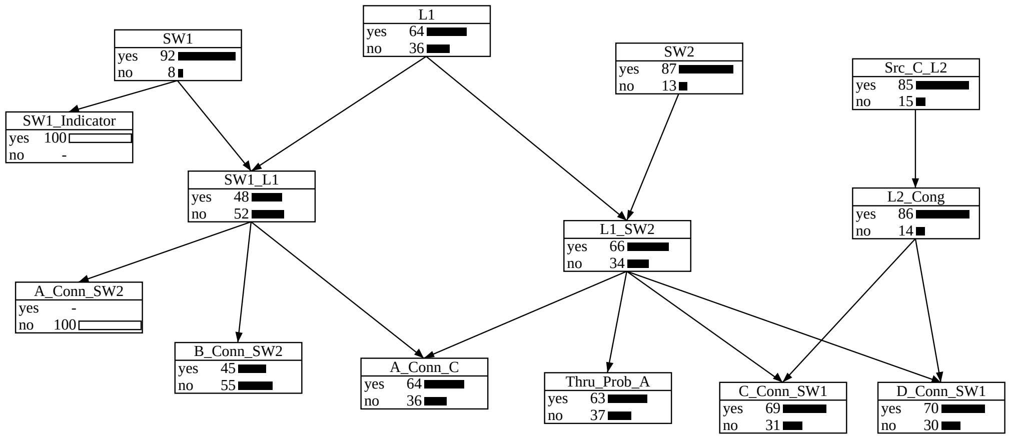

Figure 6.10: After *SW1_Indicator* Observed as *normal*

Simulations on other scenarios present similar results, which we do not list here.

Table 6.1: Cost for All Leaf Nodes

SW1_Indicator	A_Conn_SW2	B_Conn_SW2	A_Conn_C	Thru_Prob_A	C_Conn_SW1	D_Conn_SW1
2	1	7	1	3	1	3

6.6 Service Failure Diagnosis

The previous example and discussions are presented from the *network* point view. In this section, we extend our method to service failure diagnosis.

6.6.1 Network-Oriented and Service-Oriented Views

In Telecommunication Management Network (TMN) specifications, the ITU-T defined a "Logical Layered Architecture" - LLA [73]. According to this model, some of the most important aspects of the management process are utilized as

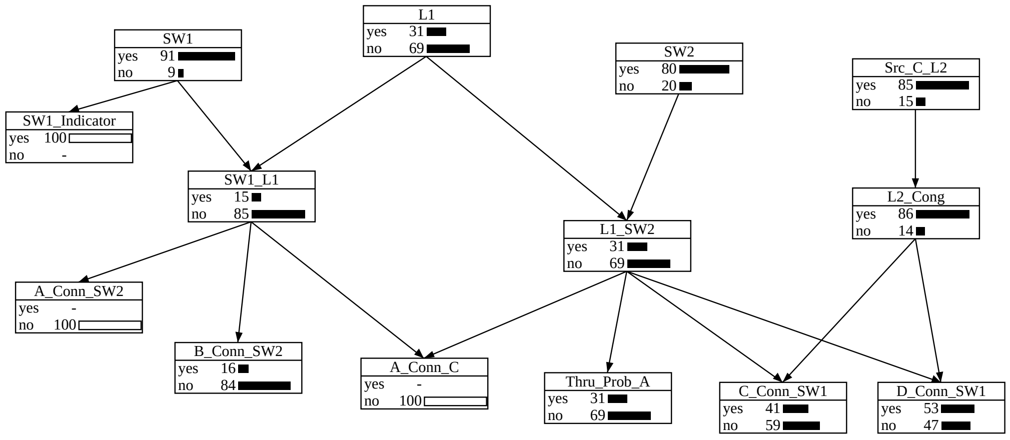

Figure 6.11: After *A_Conn_C* Observed as *Abnormal*

Table 6.2: Comparison of Node Selection Schemes

Symptom Nodes	Random Selection		Relevance Selection	
	Avg. Cost	Success Rate	Avg. Cost	Success Rate
A_Conn_SW2	15.38	84.5%	9.13	94%
A_Conn_C	26.21	70.1%	14.22	88%
A_Conn_SW2 and A_Conn_C	24.68	67.8%	3	100%

criteria for the grouping of the functionality of the operation support systems (OSF) according to the four logical management layers:

- Network Element Management Layer

  In this layer, the functions referring to the management of individual network elements or to the network element groups are situated. The OSFs of this layer provides, to the OSFs of the upper layer, the access to the functionality of network elements and to the implementation of relationships among these elements.

- Network Management Layer

  Supported by the functionality of the network elements management layer, an OSF of this layer aims at the management of a network as a whole, which is typically distributed over an extensive geographical area. It is also the objectives of this layer to provide the upper layer with a network vision which is independent of the technologies utilized in its implementation.

  Since they have a global vision of the managed network, the OSFs of this layer are able to know, monitor and control the utilization of the network resources, thus guaranteeing its functioning according to adequate performance standards and service quality.

- Services Management Layer

  In this layer, the OSFs aim at knowing, monitoring and controlling the contractual aspects of the services offered to the clients, including the receipt, processing and closing of service orders and complaints.

  This layer provides the main point of contact of clients with the service provider and so it must have updated and precise information on the activation and deactivation of services, the quality of these services and the occurrence of failures in the rendering of these services.

- Business Management Layer

  One of the goals of the OSFs of this layer is the interaction with other OSFs, in order to obtain a better utilization of the telecommunications resources, under the business point of view, which consists of searching the best return over the investment. Other attributions of the OSFs of this layer include the support to the decision processes related to the realization of new investments and to the allocation of resources (human and material) for the operation, administration and maintenance of telecommunications resources.

OSFs with fault management functionality can exist in various layers, and as we stated in chapter 3, belief networks can cover the modeling needs for fault diagnosis purposes of all the first three layers.

Our previous examples and discussions focus on the fault diagnosis issues in the network management layer, where the events are collected from the *network* or *system* point of view. Such a view relates to network-specific information about topology and connections between network elements and it also applies to network element layer fault diagnosis in a device-oriented manner.

In the service management layer, however, the events are usually collected not from the viewpoint of their underlying network system, rather, they are collected from service point of view.

In a telecommunications environment, we have to deal with an end user (customer), a service provider and a network provider. Consider a large customer, maybe a bank or an inter-continental enterprise, that has internal computing systems at a number of locations, and subscribes to the network services offered by one or more communication network service providers. The *services* provided by these providers, though specialized based on specific customer needs and tariffs, are nonetheless formed from common networking resources or facilities (e.g., high-capacity optical fiber and shared public switches, etc.) maintained by network providers.

The end users are usually furnished with service surveillance tools that help them monitor the status of their offered services according to the contractual aspects of the services. In cases of service failure, either no service or degradation of service, the end users report troubles to the service provider. Such reports represent troubles in terms of user services and they are typically stored in some database(s) at the service provider site.

The service provider has information about the services it offers to the customers and may be able to do some correlation and diagnosis based on reports from multiple customers for particular services, according to some internal fault model.

The network people think of the troubles in terms of the operation of network facilities, not service or customer terms. Our previous examples take this viewpoint.

Thus, in service layer fault diagnosis, we need to correlate the user's view, thinking in terms of user services (e.g. file transfer, QoS parameters, etc.), and the personnel's view, mostly thinking in terms of supportive services (e.g. reachability, IP, Database, naming, etc.) within a service hierarchy. Such a service hierarchy typically spans multiple protocol layers (application, tcp, ip, etc.) and it can be represented by a belief network. Our fault diagnosis method discussed in this chapter can then take effect.

6.6.2 From Service Dependency to Belief Network

From the service-oriented point of view, a service, used by an end user, is described with a set of subservices which themselves may be represented with a set of subservices. A subservice in layer N provides its functionality to the service on layer $N - 1$, and uses some subservices on layer $N + 1$. Relations between the services and subservices are represented in a *service graph* . Vertices in the service graph represent services and subservices, while the directed edges represent the relation that a service "uses" the contained subservices for the provision of its functionality. Due to the recursive nature of the service description, and the existing service hierarchies, the service graph has to be layered. The root vertices of the service graph represent services as used by end users, and are placed on the refinement layer 0. When a user senses some service problem, either no service or service quality degradation, he/she then sends out a trouble report to the service provider. Fault documentation and diagnosis will happen at the service provider site.

In [2] the fault diagnosis procedure essentially starts from the reported service and inspects the subsequent subservices on some particular branches as indicated in the service graph, until a leaf vertex is reached. Here, we use belief network as the probabilistic fault model to encapsulate the inherent uncertainty among the services and subservices. Our fault diagnosis method can handle multiple faults and facilitate efficient query generation for the customer report database.

The conversion from a service graph to a belief network is straightforward. Vertices keep the same. The directed edges in a service graph is adapted by reversing the arrow direction. And CPTs are provided for each vertex to represent quantitatively the dependent relation. See figures 6.12 and 6.13 for a service graph and a converted belief network.

6.6.3 Service Failure Diagnosis using Belief Networks

Given a belief network model for the service dependency, we can apply our decision-theoretic diagnosis strategy discussed in this chapter to pinpoint the most likely cause, up to the granularity of the current model. By appropriately assigning the step cost and termination cost, we could stop as soon as *one* root

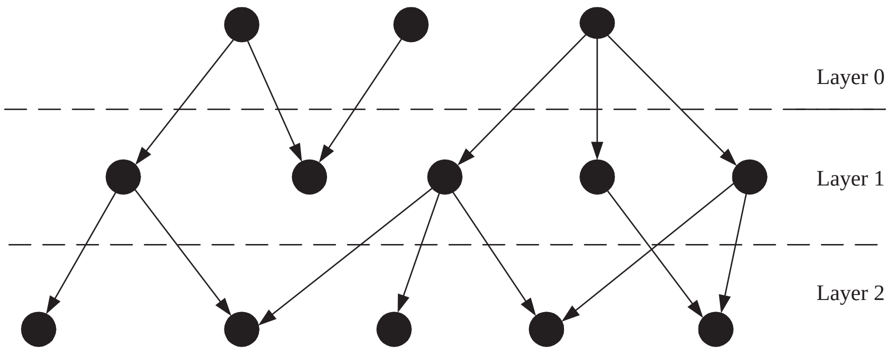

Figure 6.12: An Example Service Graph

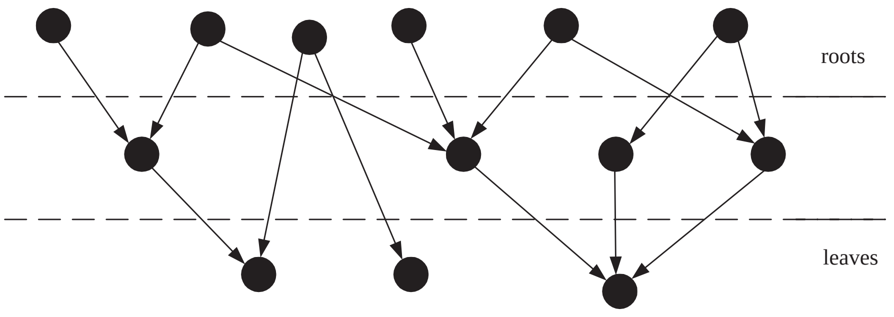

Figure 6.13: Belief Network Model from Service Graph

node[1] is identified with only *one* round of inference. This emulates the work in [2].

Further, we could choose appropriately the step and termination cost such that *multiple* steps could be taken and *multiple* causes could be identified. This entails cleverly choosing and checking other types of services that possibly share some subservices with current problematic service(s). A query to the customer trouble database(s) can be automatically generated based on the selection of the next test node. See figure 6.14 for an illustration.

Recent work at IBM [106, 107] about problem determination for application

[1]leaf node in [2].

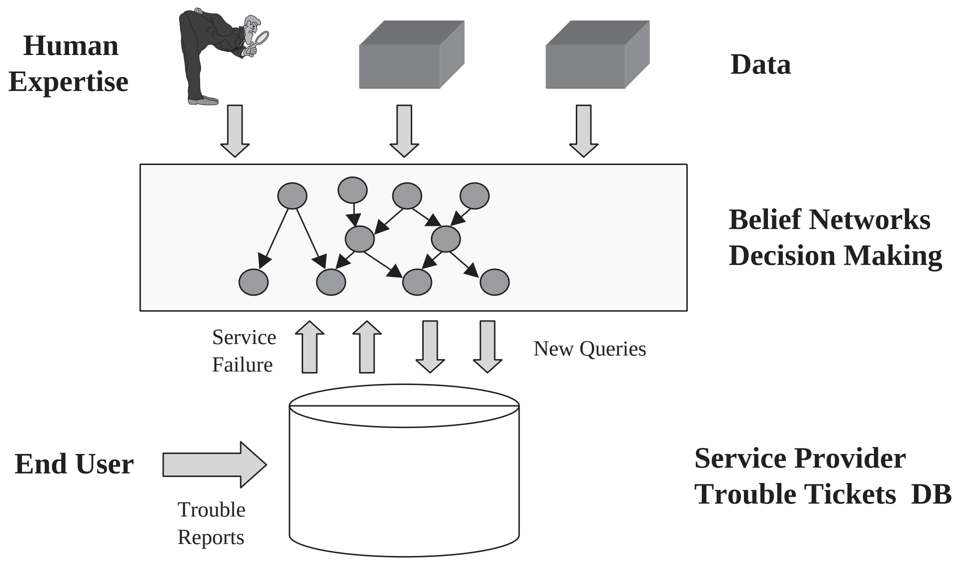

Figure 6.14: Service Failure Diagnosis using Belief Networks

services in an e-commerce environment uses similar ideas as that presented in this chapter. They use dependency graphs to encode the service hierarchy and the diagnosis is based on computing a basis set from the dependency chart (matrix) derived from the dependency graph. The basis set contains the candidate causes for current malfunctions and it could be large initially. Then, similar to my work, they have a parallel simulation model of the dependency graph based on which such candidate causes are set to *faulty* and see if they lead to the original symptoms. If so, correct diagnosis is claimed; if not, this set is shrinked via some elimination mechanism. This procedure is called *fault injection*.

Our work also use the parallel simulation model, the intervention belief network, and our idea of right diagnosis is based on setting the suspicious nodes to be *normal*. Further, we are cautious in choosing multiple candidate causes and our set of suspicious nodes is increased, rather than shrinked with rounds of diagnosis. This is based on our belief that symptoms are mostly due to one or at most a few causes. Moreover, we propose the decision-theoretic test generation scheme, which facilitates efficient network element monitoring or automatic database query generation. Our method is a novel approach in the service failure diagnosis research area.

6.6.4 Remarks

The different views discussed above lead to interesting thoughts regarding belief network constructions for fault diagnosis. The network view focuses on the operation status of the network components, and such status depend on various layers of protocol parameters, all the way *up* to network users behavior. These will all serve as ancestral nodes in the constructed belief network model, with various status nodes of network components as the leaf nodes. On the other hand, the service-oriented view starts from customer's service trouble report. Customer's services depend on subservices in a recursive way, all the way *down* to network facilities. Consequently, customer services will be the descendent nodes.

The seemingly reverse order of the two kinds of belief networks raise interesting questions. For example, what problems are best handled by dealing only with network-oriented models? What problems are best handled only with the service-oriented models? Further, since the services depend on the network resources eventually, can we build belief network models that can encapsulate belief networks built from both views, in some object-oriented manner, and the diagnosis can be carried out by their cooperation? What would be the communication scheme that ensures the validity? Further, is it worth doing this? These are open research questions.

6.7 Conclusions

In this chapter, we presented a framework that supports intelligent fault and performance management for communication networks. We used belief networks as the knowledge representation scheme and inference engine for the problem domain. The optimal stopping problem is tackled by using the notion of right diagnosis via intervention, and test selection is based on a heuristic dynamic strategy. Simulation shows that this scheme is much superior than a random selection scheme. Note that as evidence accumulates, we may input them one by one followed by a propagation right after each evidence-input, as we have shown in this chapter, or we may input them once altogether and do only one propagation. This provides us the flexibility for either on-line diagnosis or off-line diagnosis/analysis.

This framework is quite **general**. As discussed in section 3.3, belief network models have very rich expressive capability and can encompass various graph-based fault models like dependency graph, service graph and causal graph. The causes and effects are not necessary to be linked in a direct way; hidden and complex dependencies can be modeled easily. Further, the belief network model and the associated decision making algorithm could exist at any management station in a network management system.

Due to the event correlation procedure prior to the diagnosis process, only a small fraction of the so many alarms generated in a big problem domain is chosen as input to a belief network model. Thus, the diagnosis based on such condensed events tackles much less symptoms, which makes our framework and algorithm **scalable** and run fast.

Moreover, our framework is **robust** to noise and incomplete data. By nature, belief network models handle the problem of uncertainty in the cause and effect relationship among propositions. Based on probability theory and the local conditional independence structure, a belief network model is a compact way for knowledge representation under uncertainty and facilitates efficient inference over the random variables included therein. In terms of observation noise, spurious alarms can be easily tackled in the event correlation phase; if the input events are not complete, i.e., one or more of the condensed events from the event correlation process is lost, the lost events can be easily demanded by our dynamic troubleshooting strategy if such lost events are calculated as relevant for further diagnosis. By the term of model noise, we mean the model is not accurate in describing some causal relations. Then during diagnosis we will find that our model can not give out solutions that lead to right diagnosis in many situations. This can be solved by analyzing logs and correcting the corresponding belief network model, both improving the structure and the associated parameters. Further, we observe from our experience of statistical parameter learning that, in terms of fault diagnosis, the *true* and *learned* belief networks would give the *same* test sequences and average cost under most of the symptom patterns, and we conclude that such diagnostic belief network models are not so sensitive to the parameters.

After a test node is chosen, the observation for this test may take advantage of the traditional SNMP paradigm by polling appropriate MIB variables; or in our case, delegated (mobile) agents could be sent to the network elements to collect the data by using the management by delegation paradigm [108, 21]. As one example of such an agent-based environment, the authors presented in [33] a couple of system designs for adaptive, distributed network monitoring and control. See also Chapter 2.

Chapter 7

Conclusions and Future Work

In this dissertation, we presented our design of an intelligent, distributed fault and performance management system for communication networks. The architecture is based on a distributed agent paradigm, with belief networks as the framework for knowledge representation and evidence propagation.

The dissertation consists of four major parts. First, we choose the mobile code technology to help implement a distributed, extensible framework for supporting adaptive, dynamic network monitoring and control. The focus of our work is on three aspects. First, the design of the standard infrastructure, or Virtual Machine, based on which agents could be created, deployed, managed and initiated to run. Second, the collection API for our delegated agents to collect data from network elements. Third, the communicating finite state machine based callback mechanism through which the functionality of the delegated agents or even the native software could be extended. We propose three system designs based on such ideas.

Second, we propose a distributed framework for intelligent fault management purpose. The managed network is divided into several domains and for each domain, there is an intelligent agent attached to it, which is responsible for this domain's fault management tasks. Belief network models are embedded in such an agent as the probabilistic fault models. For those problems that none of the individual agent can solve, there is a mechanism by which the agents can report to the coordinator and share the information in order to get a global view and solve it cooperatively.

Third, we address the problem of parameter learning for belief networks with fixed structure. Based on the idea of Expectation-Maximization (EM), we derive a uniform learning algorithm under incomplete observations. Further, we study the rate of convergence via the derivation of Jacobian matrices of our algorithm and provide a guideline for choosing step size. Our simulation results show that the learned values are relatively close to the true values. This algorithm is suitable for both batch and on-line mode.

Finally, when using belief networks as the fault models, we identify two fundamental questions: When can I say that I get the right diagnosis and stop? If right diagnosis has not been obtained yet, which test should I choose next? The first question is tackled by the notion of right diagnosis via intervention, and we solve the second problem based on a dynamic decision theoretic strategy. Simulation shows that our strategy works well for the diagnosis purpose. This framework is general, flexible, scalable and robust.

The work accomplished in this dissertation can be illustrated in figure 7.1. Network elements are equipped with some intelligence and API to facilitate hosting environment to delegated agents for the local monitoring and control. Belief network models are built by the incorporation of human expertise and empirical data, based on which evidence propagation and decision making are carried out. The information about the selected test node in the belief network is then obtained through the adaptive monitoring framework, whereby another round of diagnosis can take place. Illustration of the service failure diagnosis has been shown in figure 6.14.

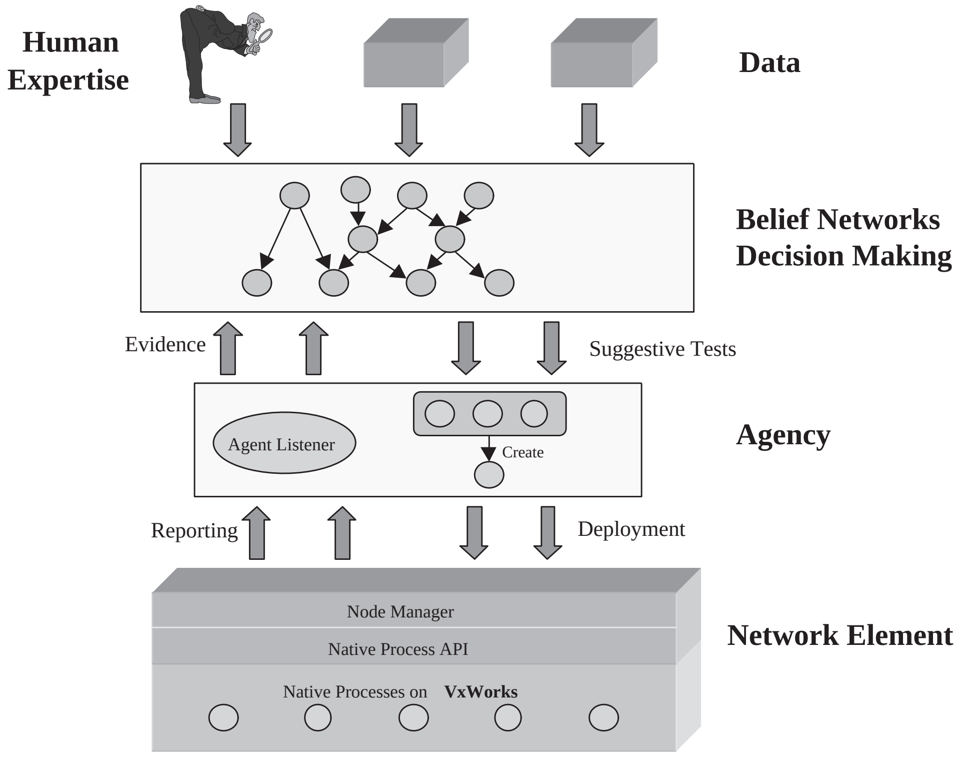

Figure 7.1: Whole Figure

Future research remains in the following aspects.

- *Formal Modeling for Function Extensions*

  In chapter 2 we presented the concept of function extension and change of logic by using pointer replacement and defining callback hooks appropriately. To ensure the correctness of these procedures, we propose to model the processing logic using extended state machines and do system state analysis on these state machines.

Further, in many applications, there is the need for the coexistence, communication and cooperation between equipments of different generations. One example is in Satellite communication networks where different versions of terminals, equipped with possibly different processing capabilities about compression, encapsulation, etc., need to coexist and communicate with each other. In such cases, we need to specify the design guidelines for state machines to make sure that such designed state machines are both backward compatible and forward extensible.

- *Extensions to Multi-Layer Fault Diagnosis*

  The methodologies presented in chapter 6 are presumably to be used within an intelligent domain trouble-shooting assistant (IDTA). However, such ideas can be extended to multi-layer fault diagnosis cases where the hierarchical belief network models comply with some appropriate constraints.

  In particular, in the network management layer, we could further divide the problem domain into multiple layers according to the inherent hierarchical architecture of a communication network (e.g. LAN, subnet, network, etc.). For each layer, we associate belief network models for fault diagnosis purposes.

  Conceptually, belief networks are used in each layer to model the cause-and-effect relations of the propositions of interest within its own layer of view. Such belief networks, each of which models a component within this layer N, provide to the upper layer $N+1$ the access to the necessary information of these belief networks for the upper layer belief networks to obtain a larger view. On the other hand, a belief network model of each layer N (except the bottom layer) also receives such necessary information from some belief networks of layer $N-1$. The top layer has the global view of the whole managed domain.

We notice that components[1] typically interact in a network environment. Components of the same hierarchy are each modeled by a belief network model of this layer, while the interactions among such components and the semantics of such interactions are modeled by upper layer belief networks with a coarser view. So in terms of our multi-layer belief networks, all the interactions among components are through vertically directed links. The interfacing nodes in the constructed hierarchical belief network should form various d-sepsets, as defined in [76], and each component belief network should be specified in an object-oriented manner [109]. Further, how should we combine both the network and service point of view?

- *Improving the Dynamic Strategy*

  The dynamic heuristic strategy could be improved via reinforcement learning [99, 110], and in particular, Q-learning techniques [111]. The idea is that, by interacting with the environment, the decision making module could accumulate experience and improves its performance. We could use the above dynamic strategy as the starting point. One possible problem when using Q-learning is that we need to represent each state explicitly to store the associated learned value. As we discussed above, this might be intractable due to the increasing history or continuous state space. However, by noticing that there will be many cases that our diagnosis could

[1]The term *component* is used here in a general and abstract sense. It could be a host, switch, a LAN, or a subnet, etc., that will interact with other peer components to form a higher layer component.

stop within only a couple of steps, we should be able truncate the expanding state space (using history process) significantly. We could start from a small example, like the one taken in chapter 6. It is shown [99, 110, 111] that such dynamic trial-and-error mechanism lead to optimal policies and we hope that we can discover some *structure* in the optimal policies derived from the simple example. It would be ideal if we can extend such a structure to a general case.

Bibliography

[1] Y. Yemini, "A critical survey of network management protocol standards," in *Telecommunications Network Management into the 21st Century*, S. Aidarous and T. Plevyak, Eds. 1994, IEEE Press.

[2] Gabrijela Dreo, *A Framework for Supporting Fault Diagnosis in Integrated Network and Systems Management: Methodologies for the Correlation of Trouble Tickets and Access to ProblemSolving Expertise*, Ph.D. thesis, Department of Computer Science, University of Munich, 1995.

[3] J. Duffy, "Net control start-up gets smarts," *Network World*, January 1996.

[4] K. Houck, S. Calo, and A. Finkel, "Towards a practical alarm correlation system," in *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management, IV (ISINM'95)*, 1995, pp. 226–237.

[5] Irene Katzela and Mischa Schwarz, "Schemes for fault identification in communication networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 753–764, December 1995.

[6] I. Katzela, *Fault Diagnosis in Telecommunication Networks*, Ph.D. thesis, Columbia University, 1996.

[7] L. Lewis, *Managing Computer Networks: A Cased-based Reasoning Approach*, Artech House, 1995.

[8] G. Mahamat, A. Das, and G.V. Bochmann, "An overview of fault management in telecommunication networks," in *Advanced Information Processing Techniques for LAN and MAN Management*, 1994.

[9] M. Stefik, *Introduction to Knowledge Systems*, Morgan Kaufmann, 1995.

[10] J. Chow and J. Rushby, "Model-based reconfiguration: Diagnosis and recovery," Tech. Rep. 4596, NASA, 1994.

[11] A. Bouloutas, G. W. Hart, and M. Schwartz, "Simple finite-state fault detectors for communication networks," *IEEE Trans. on Communication*, vol. 40, no. 3, pp. 477–479, March 1992.

[12] D. Lee, A.N. Netravali, K.K. Sabnani, B. Sugla, and A. John, "Passive testing and applications to network management," in *Proceedings of the 1997 International Conference on Network Protocols*, 1997, pp. 113–122.

[13] C. Wang and M. Schwartz, "Fault detection with multiple observers," in *Proceedings of INFOCOM*, 1992.

[14] Cynthia S. Hood and Chuanyi Ji, "Proactive network-fault detection," in *Proceedings of the IEEE INFOCOM*, 1997, pp. 333–341.

[15] M. Thottan and C. Ji, "Adaptive thresholding for proactive network problem detection," in *Third IEEE International Workshop on Systems Management*, Newport, RI, 1998, pp. 108–116.

[16] G. Jacobson and M. D. Weissman, "Alarm correlation," *IEEE Network Magazine*, vol. 7, no. 6, pp. 52–59, 1993.

[17] S. Kliger, S. Yemini, Y. Yemini, D. Ohsie, and S. Stolfo, "A coding approach to event correlation," in *Integrated Network Management*, 1995, pp. 266–277.

[18] I. Rouvellou and G. W. Hart, "Automatic alarm correlation for fault identification," in *Proceedings of IEEE INFOCOM'95*, 1995.

[19] W. Stallings, *SNMP, SNMPv2 and CMIP: the practical guide to network management standards*, Addison-Wesley, Reading, Mass., 1993.

[20] W. Stallings, *SNMP, SNMPv2 and RMON: practical network management*, Addison-Wesley, Reading, Mass., 1996.

[21] G. Goldszmidt and Y. Yemini, "Distributed management by delegation," in *Proceedings of 15th International Conference on Distributed Computing Systems*, 1995.

[22] S. Waldbusser, "Remote network momitoring management information base," 1995, RFC 1757.

[23] L. Kerschberg, R. Baum, A. Waisanen, I. Huang, and J. Yoon, "Managing faults in telecommunications networks: A taxonomy to knowledge-based approaches," *IEEE*, pp. 779–784, 1991.

[24] D. Heckerman and M. Wellman, "Bayesian networks," *Communications of the ACM*, vol. 38, no. 3, pp. 27–30, 1995.

[25] F. V. Jensen, *An Introduction to Bayesian Networks*, University College London, 1997.

[26] J. Pearl, *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, 1988.

[27] D. Spiegelhalter, P. Dawid, S. Lauritzen, and R. Cowell, "Bayesian analysis in expert systems," *Statistical Science*, vol. 8, no. 3, pp. 219–282, 1993.

[28] R. Maxion, "A case study of ethernet anomalies in a distributed computing environment," *IEEE Trans. on Reliability*, vol. 39, no. 4, pp. 433–443, 1990.

[29] H.G. Hegering, S. Abeck, and B. Neumair, *Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*, Morgan Kaufmann, San Francisco, CA, 1999.

[30] A. Leinwand and K. F. Conroy, *Network Management, A practical perspective*, Addison-Wesley, 2nd edition, 1996.

[31] J. Huard and A. A. Lazar, "Fault isolation based on decision-theoretic troubleshooting," Tech. Rep. TR 442-96-08, Center for Telecommunications Research, Columbia University, 1996.

[32] H. Li, S. Yang, and J. S. Baras, "Adaptive distributed network monitoring for performance and fault management–system designs," MIPS/HNS Project Report, May 2001.

[33] H. Li, S. Yang, H. Xi, and J. S. Baras, "Systems designs for adaptive, distributed network monitoring and control," in *Proceedings of IFIP/IEEE*

International Symposium on Integrated Network Management, Seattle, WA, May 2001, pp. 77–90.

[34] H. Li, S. Yang, and J. S. Baras, "On system designs for distributed, extensible framework for network monitoring and control," Tech. Rep. CSHCN TR 2001-12, Center for Satellite and Hybrid Communication Networks, University of Maryland, 2001.

[35] M. Sloman and K. Twidle, *Networkand Distributed Systems Management*, Addison-Wesley, Wokingham, UK, 1994.

[36] J. S. Baras, H. Li, and G. Mykoniatis, "Integrated, distributed fault management for communication networks," Tech. Rep. CSHCN TR 98-10, Center for Satellite and Hybrid Communication Networks, University of Maryland, 1998.

[37] H. Li, J. S. Baras, and G. Mykoniatis, "An automated, distributed, intelligent fault management system for communication networks," in *Proceedings of ATIRP'99*, 1999.

[38] H. Li, "Statistical parameter learning for belief networks with fixed structure," Tech. Rep. CSHCN TR 99-32, Center for Satellite and Hybrid Communication Networks, University of Maryland, 1999.

[39] H. Li and J. S. Baras, "A framework for supporting intelligent fault and performance management for communication networks," in *Proceedings of IFIP/IEEE International Conference on Management of Multimedia Networks and Services*, Chicago, IL, October 2001, pp. 227–240.

[40] A. Fuggetta, G.P. Picco, and G. Vigna, “Understanding code mobility,” *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 342–361, 1998.

[41] Luderer G., Ku H., Subbiah B., and Narayanan A., “Network management agents supported by a java environment,” in *Proceedings of the 5th IFIP/IEEE International Symposium on Integrated Network Management (IM’97)*, 1997.

[42] G. Pavlou et al, “Distributed intelligent monitoring and reporting facilities,” *Distributed Systems Engineering*, vol. 3, no. 2, pp. 124–135, 1996.

[43] M. Mountzia, “A distributed management approach based on flexible agents,” *Interoperable Communication Networks*, vol. 1, pp. 99–120, 1998.

[44] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco, “Agent extensibility (agentx) protocol,” 2000, RFC 2741.

[45] L. Heintz, S. Gudur, and M. Ellison, “Definition of managed objects for extensible snmp agents,” 2000, RFC 2742.

[46] A. Bieszczad, B. Pagurek, and T. White, “Mobile agents for network management,” *IEEE Communication Surveys*, vol. 1, no. 1, pp. 2–9, September 1998.

[47] T. Magedanz, “Intelligent mobile agents in telecommunication environments - basics, standards, products, applications,” 1999, Tutorial at the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM’99).

[48] General Magic Inc., “Mobile agent technology,” .

[49] J. Gosling and H. McCuilton, "The java language environment (white paper)," 1995.

[50] B. Welch, *Practical Programming in Tcl and Tk*, Prentice-Hall, 1995.

[51] General Magic Inc., "Telescript technology: the foundation for the electronic market-place," 1996, white paper.

[52] M. Leppinen, "Java- and corba-based network management," *Computer*, vol. 30, no. 6, pp. 83–87, 1997.

[53] J.P. Martin-Flatin, "Push vs. pull in web-based network management," in *Proceedings of the 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99)*, Boston, MA, 1999, pp. 3–18.

[54] WindRiver, "Vxworks," .

[55] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *Journal of the ACM*, vol. 2, no. 5, pp. 323–342, 1983.

[56] G.M. Lundy and R.E. Miller, "Specification and analysis of a data transfer protocol using systems of communicating machines," *Distributed Computing*, vol. 5, pp. 145–157, 1991.

[57] D.C. Feldmeier, A.J. McAuley, J.M. Smith, D.S. Bakin, W.S.Marcus, and T.M. Raleigh, "Protocol boosters," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 437–444, April 1998.

[58] H. Xi, "Java-based intelligent network monitoring," 1999, M.S. Thesis, University of Maryland.

[59] Inc Sun Microsystems, "Java 2 platform micro edition (j2me) technology for creating mobile devices," May 2000, White Paper, Sun Microsystems, Inc.

[60] Mario Baldi and Gian Pietro Picco, "Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications," in *Proceedings of the 20th International Conference on Software Engineering*, R. Kemmerer, Ed. 1998, pp. 146–155, IEEE CS Press.

[61] S. Russell and P. Norvig, *Artificial Intelligence–A Modern Approach*, Prentice-Hall, 1995.

[62] HUGIN, "http://www.hugin.dk," .

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

[64] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems(with discussion)," *Journal Royal Statistical Society*, vol. 50, no. 2, pp. 157–224, 1988.

[65] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[66] U. Kjaerulff, "Triangulation of graphs–algorithms giving small total state space," Tech. Rep. R-90-09, Department of Mathematics and Computer Science, Aalborg University, 1990.

[67] C. Huang and A. Darwiche, "Inference in belief networks: A procedural guide," *International Journal of Approximate Reasoning*, vol. 15, pp. 225–263, 1996.

[68] G. F. Cooper, "Computational complexity of probabilistic inference using bayesian belief networks(research notes)," *Artificial Intelligence*, vol. 42, pp. 393–405, 1990.

[69] P. Dagum and M. Luby, "Approximately probabilistic reasoning in bayesian belief networks is np-hard," *Artificial Intelligence*, vol. 45, pp. 141–153, 1993.

[70] J. Pearl, "Fusion, propagation, and structuring in belief networks," *Artificial Intelligence*, vol. 29, pp. 241–288, 1986.

[71] Y. Xiang and H. Geng, "Distributed monitoring and diagnosis with multiply sectioned bayesian networ," in *AAAI Spring symposium on AI in Equipment Service Maintenance and Support*, 1999, pp. 18–25.

[72] B. Gruschke, "Integrated event management: Event correlation using dependency graphs," in *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, October 1998.

[73] ITU-T, "Itu-t. recommendation m.3010: Principles for a telecommunications management network," May 1996.

[74] D. E. Heckerman, E. J. Horvitz, and B. N. Nathwani, "Toward normative expert systems: Part i the pathfinder project," *Methods of Information in Medicine*, vol. 31, pp. 90–105, 1992.

[75] Horvitz and M Barry, "Display of information for time-critical decision making," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, August 1995.

[76] Y. Xiang, B. Pant, A. Eisen, M.P. Beddoes, and D. Poole, "Multiply sectioned bayesian networks for neuromuscular diagnosis," *Artificial Intelligence in Medicine*, vol. 5, pp. 293–314, 1993.

[77] Y. Xiang, K.G. Olesen, and F.V. Jensen, "Practical issues in modeling large diagnostic systems with multiply sectioned bayesian networks," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 14, no. 1, pp. 59–71, 2000.

[78] J. S. Breese and D. Koller, "Bayesian networks and decision-theoretic reasoning for artificial intelligence," 1997, Tutorial at Fourteenth National Conference on Artificial Intelligence (AAAI 1997).

[79] N. Friedman, M. Goldszmidt, D. Heckerman, and S. Russell, "Challenge: Where is the impact of bayesian networks in learning?," 1997.

[80] E. Castillo, J. M. gutierrez, and A. S. Hadi, *Expert Systems and Probabilistic Network Models*, Springer, 1997.

[81] E. Charniak, "Bayesian networks without tears," *AI Magazine*, vol. 12, no. 4, pp. 50–63, 1991.

[82] H. Li, "An introduction to belief networks," Tech. Rep. CSHCN TR 99-31, Center for Satellite and Hybrid Communication Networks, University of Maryland, 1999.

[83] J. S. Breese and D. Heckerman, "Decision-theoretic case-based reasoning," *IEEE Transactions on Systems, Man, and Cybernetics–Part A:Systems and Humans*, vol. 26, no. 6, pp. 838–842, 1996.

[84] G. Jakobson and M. D. Weissman, "Real-time telecommunication network management: Extending event correlation with temporal constraints," in *Proceedings of the Fourth Symposium on Integrated Network Management*, Santa Barbara, California, 1995, pp. 290–301.

[85] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[86] D. Heckerman, "A tutorial on learning with bayesian networks," Tech. Rep. MSR-TR-94-09, Microsoft Research, 1996.

[87] H. V. Poor, *An Introduction to Signal Detection and Estimation*, Springer-Verlag, New York, 2nd edition, 1994.

[88] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the *em* algorithm (with discussion)," *Journal of the Royal Statistics Society B*, vol. 39, pp. 1–38, 1977.

[89] G. J. McLachlan and T. Krishnan, *The EM Algorithm ad Extensions*, Wiley Interscience, 1997.

[90] A. Benveniste, M. Metivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations*, Springer-Verlag, New York, 1st edition, 1987.

[91] V. Fabian, "Asymptotically efficient stochastic approximation; the rm case," *The Annals of Statistics*, vol. 1, no. 3, pp. 486–495, 1973.

[92] L. Ljung, G. Pflug, and H. Walk, *Stochastic Approximation and Optimization of Random Systems*, Birkhauser, 1992.

[93] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[94] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[95] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, 1984.

[96] T. Orchard and M. A. Woodbury, "A missing information principle: theory and applications," in *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, CA, 1972, pp. 697–715.

[97] T. A. Louis, "Finding the observed information matrix when using the em algorithm," *Journal of the Royal Statistical Society B*, vol. 44, pp. 226–233, 1982.

[98] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, United Kingdom, 1985.

[99] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Belmont, MA, 1st edition, 1996.

[100] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1999.

[101] J. Pearl, *Causality*, Cambridge Press, 2000.

[102] K. J. Astrom, "Optimal control of markov decision processes with incomplete state estimation," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, 1965.

[103] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol I and II*, Athena Scientific, Belmont, MA, 1st edition, 1995.

[104] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[105] J. Kalagnanam and M. Henrion, "A comparison of decision analysis and expert rules for sequential diagnosis," in *Uncertainty in Artificial Intelligence 4*, 1990, pp. 271–281.

[106] S. Bagchi, G. Kar, and J. Hellerstein, "Dependency analysis in distributed systems using fault injection: Application to problem determination in an e-commerce environment," in *Proceedings of IEEE Symposium on Distributed Systems:Operations and Management (DSOM)*, Nancy, France, October 2001.

[107] G. Kar and A. Keller, "An architecture for managing application services over global networks," in *Proceedings of IEEE Infocom*, Anchorage, Alaska, April 2001.

[108] M. El-Darieby and A. Bieszczad, "Intelligent mobile agents: Towards network fault management automation," in *Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management*, Boston, MA, 1999, pp. 611–622.

[109] S. Bapat, *Object-Oriented Networks: Models for Architecture, Operations and Management*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[110] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[111] H. Watkins C.J.C and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.