TECHNICAL RESEARCH REPORT

Integrated, Distributed Fault Management for Communication Networks

by J. Baras, H. Li, G. Mykoniatis

CSHCN T.R. 98-10
(ISR T.R. 98-21)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Integrated, Distributed Fault Management for Communication Networks*

John S. Baras, Hongjun Li and George Mykoniatis
Center for Satellite and Hybrid Communication Networks
Department of Electrical Engineering and
Institute for Systems Research
University of Maryland, College Park, MD 20742

April 26, 1998

Abstract

This report describes an integrated, distributed fault management (IDFM) system for communication networks. The architecture is based on a distributed intelligent agent paradigm, with probabilistic networks as the framework for knowledge representation and evidence inferencing. A static strategy for generating the suggestive test sequence is proposed, based on which a heuristic dynamic strategy is initiated. Another dynamic strategy, formulated as a Markov decision problem, is also provided. To solve this problem, reinforcement learning techniques are investigated.

*Research supported by the U.S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0002, by NASA under NASA Cooperative Agreement NCC3-528, and by a grant from Lockheed Martin Telecommunications.

1 Introduction and Motivation

In this section, we will briefly introduce the fault management problem for communication networks, followed by the evaluation of current research and the motivation for the proposed system.

1.1 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 [40]. To meet the needs of current and future communication environments, it is the responsibility of network management to maintain the network operation and service. 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. In this sense we say fault management serves as the foundation of other network management functions. 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.

Fault Management is based on three main assumptions [11]:

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

In general, any fault diagnosis procedure can be interpreted in terms of search spaces and corresponding operations [30]. 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 diagnosis,

we can also take this paradigm.

1.2 Motivation

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

- 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. 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.

  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. Therefore, fault management will have to be *automated*.

- Probabilistic expert system: Knowledge-based expert systems, as examples of automated systems, have been very appealing for complex system fault diagnosis [22] and the effort in this field is still growing. 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 and they are designed to replace the human experts. 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. Instead of replacing the human expert, the expert system based on such a probabilistic model is expected to behave as the assistant to a human expert by providing processed information and suggestions timely and automatically. Such systems are called *normative* expert systems.

- Distributed architecture: So far, most research and standards on fault management assume a centralized architecture where all of the symptom information has to be sent to the central manager for processing. One example is the simple manager-agent paradigm adopted by SNMP. 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 the fault diagnosis steps. Such a paradigm works well for small networks. But as the networks become larger, the centralized paradigm will incur vast amounts of information communication and thus occupy too much bandwidth unwisely. Since there are many cases where the faults can be resolved on-the-spot, there is no need to report the local faults to the central manager to get a global view. In this regard, we propose that the faults should be dealt with locally if they are local. Only those that cannot be handled locally should draw global attention. It is the authors' belief that it will be more efficient, both in time and bandwidth utilization, if faults were handled in this way. Hence, observing that communication networks are hierarchial and distributed by nature, it is most desirable to come up with a multi-layer architecture and distribute the intelligence to the lower layers that are closer to the managed objects. The entities, which have the distributed intelligence and whose responsibilities are fault diagnosis in the local domain, are referred to as "Intelligent Agents" (IA). We provide more details on IA in the next section.

- 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. Such faults are called "*hard* " faults and can be solved by replacing hardware elements or software debugging and/or re-initialization. The diagnosis of the "hard" faults is called *"re-active"* diagnosis in the sense that it

consists of basically the reactions to the actual failures. 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. Since there might not be a failure in any of the components, we call such faults "*soft*" faults. "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.

In summary, our goal is to come up with an automated, integrated, distributed fault management (IDFM) system for communication networks, which assumes a *probabilistic* model and integrates the management of both *hard* and *soft* faults. The system assumes a distributed, multi-agent architecture, in which each individual agent is responsible for the fault management of a certain local domain. In order to generate the test sequence, reinforcement learning techniques are applied.

This report is organized as follows: In section 2, we provide background knowledge on the three areas founding our approaches: intelligent agent, probabilistic reasoning and belief networks, and reinforcement learning (especially temporal difference algorithms). The proposed system architecture and function definitions are described in section 3. In section 4, we provide the mathematical formulations of the problems of interest and we describe the proposed solution methods.

2 Background

2.1 Intelligent Agents

The term "Intelligent Agent (IA)" is originated from the field of Artificial Intelligence (AI), in particular Distributed Artificial Intelligence (DAI). It has been used since early 1980's to reflect the idea of creating "autonomous objects that think". This term has also been used for years in the field of distributed computing where it refers to some specific entities (client or server) that will solve specific tasks in a distributed environment.

There are various definitions on what an intelligent agent is. One of them is [28]: *An* **agent** *is anything that can be viewed as* **perceiving** *its environment through sensors and* **acting** *upon that environment through* **effectors**. The terms "sensors" and "effectors" should be interpreted in a broad sense. According to this definition, an intelligent agent can be either a hardware or a software entity. In this report, we concentrate on software agents. One such agent is shown in figure 1.

There are various types of IAs: some are designed to solve the whole problem all by themselves while others have to share the local information and work together; some are mobile, which will be flexible in some cases, some are static; some can learn and adapt to the dynamic environment, some don't. Despite the diversity, there are some important common properties that distinguish an IA from conventional software programs:

- Autonomy: an agent is an autonomous and self-contained software entity which acts typically on behalf of a user or a process, enabling

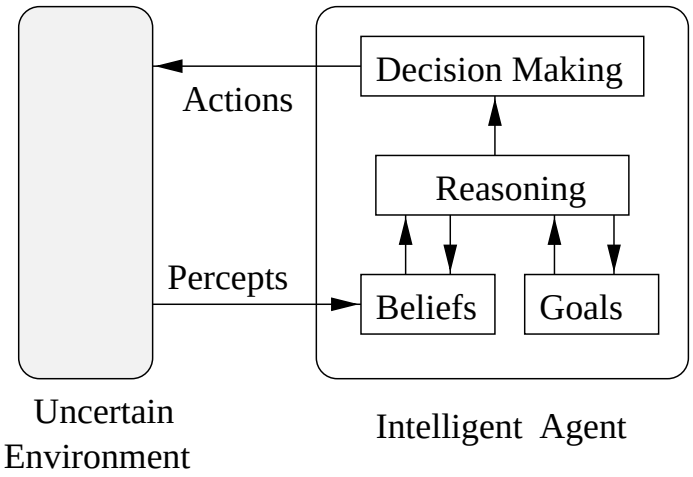

Figure 1: Illustration of an intelligent agent

task automation without direct intervention of a human or others.

- Intelligence: an agent contains some level of intelligence, ranging from simple prespecified rules to self-learning adaptive machines. In our problems, we are considering self-learning agents.

- Cooperation: the agent system allows for cooperation between individual agent entities, especially in a distributed problem-solving case. Such a system is called a "multi-agent distributed system", as our proposed system for fault management would be.

- Asynchronous Operation: an agent may be event or time triggered, independent of its users or other agents. It may be even mobile, moving from one domain to another to access the remote resource.

The various agent technologies existing today can be classified roughly as

single-agent systems and multi-agent systems. In a single-agent system, an agent performs a task on behalf of a user or a process. It can communicate with the users as well as with local or remote resources, but not with other peer agents. In a multi-agent system, on the contrary, agents may communicate with each other extensively to achieve their individual or joint goals. Surely the agents can also communicate with users or resources. In a single-agent system, *Local agents* and *Networked agents* can be identified; In a multi-agent system, *DAI-based agents* and *Mobile Agents* are distinguished, as shown below [24]:

- Local agents: designed to access local resources only.
- Networked agents: can access not only local but remote resources, and thus have a more or less detailed knowledge about the network infrastructure and services provided throughout.
- DAI-based agents: Distributed Artificial Intelligence (DAI) based multi-agent systems are concerned with the coordination of the intelligent behavior among a collection of autonomous intelligent agents. The agents can be designed using AI techniques, like rule-based or case-based reasoning. Agents can communicate with each other and with users or system resources.
- Mobile agents: mobile agents aim primarily at large computer networks which offer a huge number of sophisticated services. This in particular enables the concept of "remote programming" [24] , which

is regarded as an alternative to the traditional "Client/Server programming" based on the Remote Procedure Call (RPC) paradigm.

In this report, we propose a DAI-based problem-solving agent system for fault management. The focus is on agent design and cooperation with no mobility included.

To design an intelligent agent, there are some issues to be considered [28]:

- Agent type: basically the name that reflects the purpose of the agent. For example, medical diagnosis system, or satellite image analysis system.
- Percept: the inputs for the system. For example symptoms and observations for the diagnosis system.
- Actions: the possible action an agent can take to fulfill the goal. In medical diagnosis, the actions are questions, tests and treatments.
- Goals: for instance, a healthy patient and minimal costs in the medical diagnosis case.
- Environment: Is it accessible or not? Is it deterministic or stochastic? Is it static or dynamic? Is it discrete or continuous?

For more information on intelligent agent theory and practice, we refer to [39]. Multi-agent systems are introduced in an excellent survey paper by P. Stone from a machine learning perspective [31].

2.2 Probabilistic Reasoning and Belief Networks

Belief networks, also called Bayesian networks or probabilistic causal networks, were developed in the late 1970's to model the distributed processing in reading comprehension. Since then they have attracted much attention and have become the general knowledge representation scheme under uncertainty [27].

2.2.1 Representing knowledge in an uncertain domain

The attempts to model human's 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 much research in various areas within the artificial intelligence discipline. One of the most popular approaches to AI involves constructing an *"intelligent agent"* that functions as a narrowly focused expert.

While the past decades have seen some important contributions of expert systems in medical diagnosis, financial analysis 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 [25]. For example, if we assume that human knowledge is represented by a joint probability distribution(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.

Though it is true that JPD suffices to answer all kinds of queries on $x_1, ..., x_n$, 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—an unthinkably large number. 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. For example, to compute $p(\mathbf{H}|\mathbf{e})$ (where $\mathbf{H}$ and $\mathbf{e}$ are the sets of hypotheses and evidence, respectively), according to Bayes' rule,

$$p(\mathbf{H}|\mathbf{e}) = \frac{\mathrm{p}(\mathbf{e}|\mathbf{H})\mathrm{p}(\mathbf{H})}{\mathrm{p}(\mathbf{e})}$$

we need to compute the marginal probabilities $p(\mathbf{e})$, $p(\mathbf{H})$ and the likelihood $p(\mathbf{e}|\mathbf{H})$, which incurs enormously large number of calculations.

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.

Definition *A belief network is a Directed Acyclic Graph (DAG) in which:*

- *The nodes represent variables of interest (propositions), which maybe be discrete, assuming values from finite or countable states, or may be continuous.*

- *The set of directed links or arrows represent the causal influence among the variables and the parents of a node are all those nodes with arrows pointing to it.*

- *The strength of an influence is represented by conditional probabilities*

attached to each cluster of parent-child nodes in the network.

2.2.2 The semantics of belief networks

I. Representation of joint probabilities

Based on the chain rule of probability, we have

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

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

$$p(x_1, ..., x_n) = \prod_{i=1}^{n} p(x_i | \Pi_i) \tag{2}$$

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

$$p(x_i | x_1, ..., x_{i-1}) = p(x_i | \Pi_i) \tag{3}$$

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 $\{x_1, ..., x_{i-1}\} \backslash \Pi_i$.

Therefore, we can construct a belief network following the steps below [28]:

- 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) is satisfied
  - Specify the CPT for it

II. 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? This is an important issue in designing inference algorithms.

Fortunately, the answer is yes and the methods are provided by the notion of **d-separation**, which means **direction-dependent separation** [26]. If each undirected path from a node in X to a node in Y is d-separated by E, we say X and Y are *blocked* , which means there will be no way at all for X and Y to communicate if we remove E, and thus conditionally independent.

Definition [26] : *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*

- *A is a head-to-head node (with converging arrows) in the path, and neither A nor its descendents are in Z, or*
- *A is in Z and A is not a head-to-head node.*

2.2.3 Inference in belief networks

We have seen that a belief network can simulate the mechanism that operates in the environment and represent the JPD of the domain random variables. Thus it allows for various kinds of inferences(also called **evidence propagation**) [28]:

- **Diagnosis inferences**: From effects to causes, also called abductive inferences, bottom-up or backward inference. For example: "What is the most probable explanations for the given set of evidence?"
- **Causal inferences** : From causes to effects, also called predictive inferences, top-down or 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.

There are basically three types of algorithms for propagating evidence: exact, approximate and symbolic. By *exact propagation* we mean a method that, apart from precision or round-off errors, computes the probability distribution of the nodes exactly. The most influential exact inference algorithm is proposed in [23] and this is the algorithm we would prefer to use for fault management; but its complexity is high. By *approximate propagation* we mean the answers computed are not exact but, with high probability, lie within some small distance of the correct answer. Finally, *symbolic propagation*, which computes the probabilities in symbolic form, can deal not only with numerical values, but also with symbolic parameters.

Exact evidence propagation in an arbitrary belief network is NP-hard [12]. Fortunately, the complexity can be estimated prior to actual processing and when the estimates exceed reasonable bounds, an approximation methods such as stochastic simulation can be used instead. But even approximate inference (using *Monte Carlo simulation*) is also NP-hard if treated in general [14]. 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.

2.2.4 Learning belief networks

In previous discussions we assumed that both the network structure and the associated CPT are 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 CPT can be estimated using 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. For more information on learning in belief networks, we refer to [13] [17].

As one may expect, learning belief networks consists of both structural learning (deriving the dependency structure G) and parametric learning (estimating P). The structure of the network may be *known* or *unknown*, and the variables in the network may be *observable* or *hidden*. There are four types of combinations [28]:

- **Known structure, fully observable**: The only thing needed is to specify the CPT and this can be done by directly using the statistics of the data set S. Search methods such as hill-climbing or simulated annealing can be exploited to accomplish the fitting of data.

- **Unknown structure, fully observable**: We have to first reconstruct (extract) the topology, which entails determining the best possible structure through a space of available alternatives. The search is basically enumerative. Fitting the data to a particular structure

reduces to the above fixed-structure problem.

- **Known structure, hidden variables**: As stated before, human experts would rather give dependence relationships between random variables than the corresponding numerical values, especially when not all of the variables are observable. Therefore we can say that finding the topology of the network is often the easy part and thus the *known structure, hidden variable* learning problem is of great importance. Such problems are quite analogous to neural network learning, where gradient descent methods can be used [6].

- **Unknown structure, hidden variables**: When there are some hidden variables, the previous extraction techniques would not apply and there is no known good algorithms for this problem.

2.2.5 Remarks

In a word, 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. The contributions of belief networks can be summarized as follows [8]:

- 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 [15].

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 [9][10][20][26].

2.3 Reinforcement Learning

2.3.1 Introduction

In order to let the (software) agent do the right things, the human designer can choose to program beforehand everything for every possible cases or, he/she can just set up the objectives and some rules for the agent, which will in turn find *how* to fulfill the objectives by learning from the interactions with the environment governed by those rules. In the first approach, since the designer has programmed everything, or the designer has told the agent exactly how to behave, the agent is nothing more than a conventional software program. The design labor is concentrated totally on the designer and such software programs don't have any adaptability. For the second approach, it is the agent itself that learns the right things to do by gaining experience through trial-and-error. The designer does not prescribe to the agent what to do and so the design labor is reduced drastically. As discussed before, such an adaptive agent is called "intelligent" agent. In practice, es-

pecially in an uncertain environment whose properties are difficult to predict *a priori*, the latter approach is more favorable.

For an intelligent agent that has to learn by itself (we call it a reinforcement learning agent, or RL agent), the underlying mechanism of reinforcement learning is motivated by animal learning studies, as illustrated psychologically by Thorndike's Law of Effect:

"Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. "

So if the RL agent gets rewards by performing an action in a situation, it will most likely choose the same action again when encountering the same situation later. If it gets penalties instead, the action will be less likely to be chosen again. The rewards or penalties are referred to as "reinforcement signal".

In a situation, if the RL agent would already know all of the reinforcement signals for each possible action, it would simply choose the best action that would incur the best reward. This best reward is called "value function" for this situation. Things would be much easier in such cases. The agent doesn't have to learn anything, what it has to do is just to select the

best action.

However, the problem faced by a RL agent is: It doesn't know beforehand all of the reinforcement signals for each action. So, it has to figure them out by performing the actions and see; this can also be done via simulation, as suggested in [5]. This process is called exploration. Usually, the agent cannot wait until all of the actions have been tried, which will be very time-costly, especially when the state space and/or the action pool are of large dimension. By performing exploration, it is hoped that enough knowledge of the environment can be gained in a short length of time, without exhaustive trials on the actions.

One of the objectives of the agent is to minimize the total cost along the way of reaching goals. In order to do this, it is straightforward for the agent to be "greedy" by choosing the currently best action in a state. This process is called exploitation. Most of the time the greedy style would be supposed to lead to minimal costs. But, since exploration is not supposed to be exhaustive, and thus the current best action may not be the ultimate best one at all, there are still some cases when the agent cannot achieve an optimal or even sub-optimal solution using greedy methods only. In this regard, it might be advisable to explore more once in a while rather than sticking to the current knowledge all the time. This is actually a trade-off between exploration and exploitation. For a good introduction on this topic, we refer to [35]. The idea of such trade-off is just like what Confusius said more than a thousand years ago: "*learning without thought is labor lost; thought without learning is perilous* ."

Note that in many cases, the rewards might be of short-term or long-run. Markov decision processes provide a framework for analyzing the trade-off between these two types of rewards. So if formulated properly, the RL agent can also model delayed rewards.

In conclusion, reinforcement learning is a model of the learning problems encountered by an agent that learns behavior through trial-and-error interactions with a dynamic, uncertain environment. In this section, we will use TD-methods, the most important RL algorithms, as an illustration.

2.3.2 Temporal-difference learning

Temporal-difference learning was originated by Sutton [32] when he considered the problem of *learning to predict*, which uses past experience of an incompletely known system to predict its future behavior. Unlike the conventional prediction-learning methods that assign credit based on the difference between predicted and actual outcomes, this method assigns credit by means of the difference between *temporally successive predictions*. So its training examples can be taken *directly* from the temporal sequences without a special supervisor or teacher. This method, incremental in nature, is especially useful in the problems where future behaviors (teachers) are not available or too difficult to obtain, as most of the practical applications manifest. Moreover, it is claimed that TD-methods make more efficient use of the experience than supervised-learning methods and also, they converge more rapidly and make more accurate predictions along the way. The requirement of using TD-methods on a system is quite gentle — the system

is dynamical, so that it is sequential in nature and the emphasis on the temporal sequences makes sense.

Perhaps the most distinguishable success of using TD-methods is Tesauro's TD-Gammon [33] [34], which, by learning from the environment with no prior knowledge from a teacher, plays significantly better than the previous world-champion program and as well as expert human players. In the communications networks area, Singh and Bertsekas [29] formulate the dynamic channel allocation problem in cellular telephone systems as a dynamic programming (DP) problem and apply $TD(0)$ to execute the approximation. It has been shown that, in terms of call blocking probability, their schemes are better than those previously used in industry.

In this report, we consider the problem of predicting the cost-to-go function in a DP formulation. The applicable environments are finite or countably infinite state Markov chains and continuous state Markov processes [36].

For a Markov chain $\{i_t\}$, the cost-to-go function defined as

$$J^*(i) \stackrel{\triangle}{=} E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right]$$

is the infinite horizon expected cost for the system with initial state i, where the scalar $g(i,j)$ represents the cost of transition from state i to state j. The future costs are included by multiplying a discount factor α^t with $\alpha \in (0,1)$, which means that immediate costs are more important than future costs. The cost-to-go can be shown to satisfy some form of *Bellman's equation*

$$J^*(i) = E\left[g(i,j) + J^*(j) | i_0 = i\right], \forall i$$

The objective of DP is to calculate numerically the optimal cost-to-go function J^*. However, it is well known that for many important problems where the number of states is very large, the computational requirements of DP are so overwhelming that, in such cases, only suboptimal solutions based on some approximations can be obtained [5]. Temporal difference algorithms can be used for such approximations.

2.3.3 The $TD(\lambda)$-Algorithm

For a TD approximation to J^*, we consider a function of linear architecture

$$\tilde{J}(i,r) = \sum_{k=1}^{K} r(k)\phi_k(i)$$

where $\tilde{J}$ is an approximation to J^*, each ϕ_k is a fixed and easily computable scalar basis function and r is a parameter vector, or the associated weight vector. To approximate the cost-to-go function, one usually tries to minimize some error metric between the function $\tilde{J}$ and J^*, for example the square error, by choosing the *optimal* weight vector r^*. This is basically an unconstrained optimization problem and therefore, Newton-type methods can be used. Here, the goal is to derive a recursive updating formula for r that converges asymptotically to its *optimal* solution.

To this end, we use the following notation

$$\tilde{J}(r) = \Phi' r,$$

where $\tilde{J}(r) = [\tilde{J}(1,r), \ldots, \tilde{J}(n,r)]'$, $r = [r(1), \ldots, r(K)]'$ and Φ is a $K \times n$ basis function matrix whose i^{th} column is equal to $\phi(i)$. Then we have the

Jacobian matrix $\nabla\tilde{J}(r) = \Phi$, with i^{th} column equal to $\nabla\tilde{J}(i,r) = \phi(i)$, the partial derivatives with respect to the components of r.

Now suppose we observe a sequence of states i_t generated according to a transition probability matrix P and at time t, r has been set to some value r_t. Then at time $t+1$, we define the temporal difference d_t corresponding to the transition from i_t to i_{t+1} as

$$d_t = g(i_t, i_{t+1}) + \alpha\tilde{J}(i_{t+1}, r_t) \Leftrightarrow \tilde{J}(i_t, r_t)$$

where $\tilde{J}(i_t, r_t) \Leftrightarrow \alpha\tilde{J}(i_{t+1}, r_t)$ is the expected cost of the transition from i_t to i_{t+1}. Since we observe an actual cost $g(i_t, i_{t+1})$, the error is given, in some sense, by the difference of the two approximations, i.e. d_t.

Next, we update r_t to a better estimate r_{t+1} according to

$$\begin{aligned} r_{t+1} &= r_t + \gamma_t d_t \left[\sum_{k=0}^{t} (\alpha\lambda)^{t-k} \nabla\tilde{J}(i_k, r_t)\right] \\ &= r_t + \gamma_t d_t \left[\sum_{k=0}^{t} (\alpha\lambda)^{t-k} \phi(i_k)\right] \end{aligned}$$

where r_0 is initialized to some arbitrary vector, γ_t is a sequence of *learning rate*, which can be constant, diminishing or determined through line search. Parameterized by λ, this algorithm is usually called TD(λ).

When we put $z_t = \sum_{k=0}^{t} (\alpha\lambda)^{t-k}\phi(i_k)$, we can have a more convenient representation of TD(λ) with two iteration formulae

$$r_{t+1} = r_t + \gamma_t d_t z_t$$

$$z_{t+1} = \alpha\lambda z_t + \phi(i_{t+1})$$

$$\text{with } z_{-1} = 0$$

We would be interested in finding out whether this algorithm converges to an *optimal* r^* and if so, under what conditions. Some researchers, including Sutton [32], have shown that it does indeed converge for a class of approximators in which there are as many tunable parameters $r(k)$ as the number of states in the system. Such cases are not practical when the number of states are close to infinity or uncountable (continuous). So the more general case called *compact representations* , where the number of parameters may be less than the cardinality of the state space, should be of more interest.

Note that the equations above, as a whole, is an example of *stochastic approximation* algorithms, which have attracted much interest from various areas, such as adaptive filtering, signal modeling and system identification [3].

2.3.4 Q-Learning

The $TD(\lambda)$ algorithm can be written as the following, if we use the value function directly:

On every step, update all state i

$$J(i) \leftarrow J(i) + \gamma(g + \alpha J(j) \Leftrightarrow J(i))e(i)$$

where g is the immediate cost, $e(i)$ is the eligibility of i

$$e(i) = \sum_{k=0}^{t} (\alpha\lambda)^{t-k} I(i = i_k).$$

Q-learning is an instance of temporal difference learning. The idea is based on Q-values: let $Q^*(i, a)$ be the expected discounted reinforcement of taking

action a in state i, then continuing with the policy of choosing actions to maximize (or minimize according to the objectives) Q^*

$$Q^*(i,a) = R(i,a) + \alpha \sum_j P_{ij}(a) \max_{a\prime} Q^*(j,a\prime)$$

It is well known that the policy obtained by $\pi(i) = argmax_a Q^*(i,a)$ is an optimal policy.

The Q-learning algorithm aims to estimate the Q-values on-line, finding the policy and the value function together. For each state i, instead of trying all of its successors (like in value iteration), it chooses one action a and enters the next state j, incurring an immediate cost g, and thus obtains the learning instance (i,a,j,g) from which it can gain experience. At each state, the action can be chosen using appropriate exploration-exploitation scheme, like $\epsilon \Leftrightarrow greedy$ methods. This idea is quite similar to that in depth-first-search algorithms for spanning a graph. The update formula for $TD(0)$ Q-learning, the most widely-used Q-learning algorithm, is shown below:

$$Q(i,a) \leftarrow (1 \Leftrightarrow \gamma) Q(i,a) + \gamma \left(g + \alpha \max_{a\prime} Q(j,a\prime) \right)$$

Note that Q-learning doesn't require the system transition model $P_{ij}(a)$, so it is model-free reinforcement learning. Watkins and Dayan [38] have shown that: if each action is executed in each state an infinite number of times, and γ is decayed, the Q values will converge to Q^*, which will help yield an optimal policy.

3 Architecture and Function Definitions

In this section, we propose the architecture for our IDFM system and describe the tasks, components and the functions for an intelligent agent.

3.1 Architecture

Figure 2 shows the architecture of our IDFM system. 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. 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.

Each agent is called a "Domain Diagnostic Agent (DDA)" with the goals of monitoring the health of the domain, diagnosing the faults in a cost-efficient manner. The percepts (inputs) of a DDA are the measured data, alarms or user reports 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 stochastic in nature. The tasks of such an agent are described below.

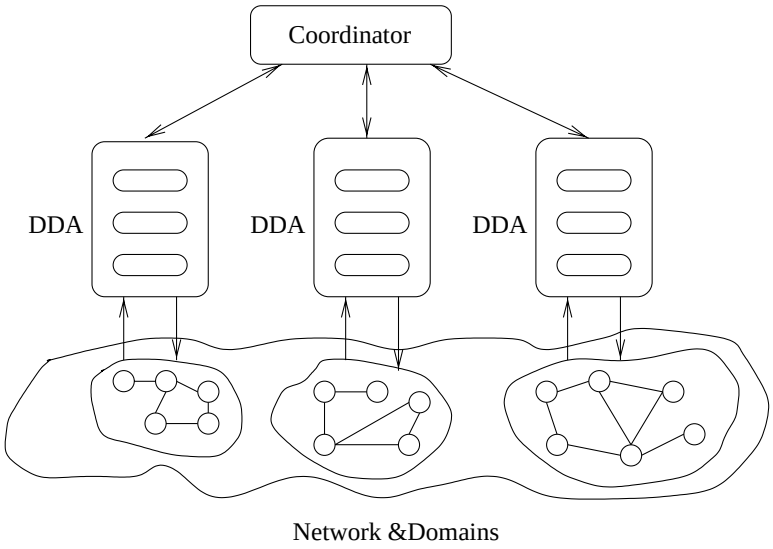

Figure 2: Architecture of Integrated, Intelligent Fault Management

3.2 Outline of DDA tasks

Adopting the space-operation paradigm discussed in the introduction , the tasks for each local DDA in IFDM are identified as: Fault Detection and Classification (FDC), Fault Localization and Identification (FLI), and Fault Corrections (FC), respectively.

- **Fault Detection and Classification**: The inputs are measured data, alarms or users reports. Such inputs are analyzed so that the current system behavior is obtained, based on which the fault hypotheses can be generated and tested. The model of "normal" behavior may be stored explicitly (such as an AR model) or implicitly (such as a MLP neural network), and model parameters should be adapted (learned) along the way. The output of the FDC is the type of fault(s). Such

detection and classification might integrate many techniques, such as filtering, change detection, hypothesis testing and neural network classifications.

- **Fault Localization and Identification**: The principal operation is to determine what might be the primary *causes* for the symptoms (fault types) recognized by the FDC. As indicated in [2], both rule-based (deterministic) expert system and neural networks can be applied here to implement such mappings and further, they can be integrated in some way to overcome either's disadvantages [1] [37]. Once more, since the relationship between symptoms and causes is probabilistic in nature, such schemes are insufficient by their own. Thus *probabilistic* fault models should be considered. Belief networks, which form the basis for probabilistic reasoning and expert systems, manifest themselves as the most suitable choice [26].

- **Fault Corrections**: We are going to generate, given the possible causes, a set of tests or repair sequences based on some heuristic or decision-theoretic strategies. This is basically a *sequential decision process* and thus can be formulated mathematically in some careful way as a Markov decision problem. In [19], Huard and Lazar give a dynamic programming (DP) formulation for the network troubleshooting problem. Noting that Huard's problem assumes single fault, our goal is to formulate the problem in a more general sense. However, traditional dynamic programming has been rejected as a feasible method for many decision problems because of the two well-known drawbacks:

"curse of dimensionality"—increase in dimension will incur explosion in state space, and "curse of model"—it's very hard to obtain the system model. So it might be very difficult to formulate a DP problem. Even if one gets the formulation, one might not be able to solve it given the huge state space. Such cases are usually referred to as "difficult" problems. Reinforcement Learning (RL), with the name borrowed from the animal learning discipline, can overcome the drawbacks mentioned above and has drawn much attention in areas like AI (especially machine learning) and control theory (especially decision and control) [5]. Such techniques will be investigated and applied to the fault diagnosis system.

Those tasks are implemented by different components of an DDA. Many such DDAs will then be distributed in the networks and act as the "local experts" for different domains.

3.3 Components of a DDA

A DDA consists of the following components, as shown in Figure 3.

- Intelligent Monitoring and Detection Assistant (IMDA)— The role of the IMDA is to monitor and analyze the data and classify the raw input data to a (set of) symptom type(s). FDC is implemented here.
- 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.

So it includes both FLI and FC. The probabilistic network model is located here.

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

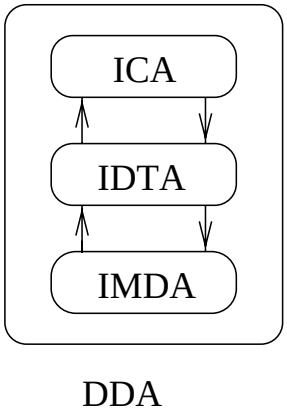

Figure 3: Components of a DDA

3.3.1 Function Definition—IMDA

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 to provide symptoms information to the IDTA, as described below and illustrated in Figure 4.

- Input: Data from network element agents MIBs, including IMDA's periodic polling and the alarms sent by the NEA.
- Output: The activation status on the output nodes, each of which is called a Problem Definition Node (PDN) and acts to represent a

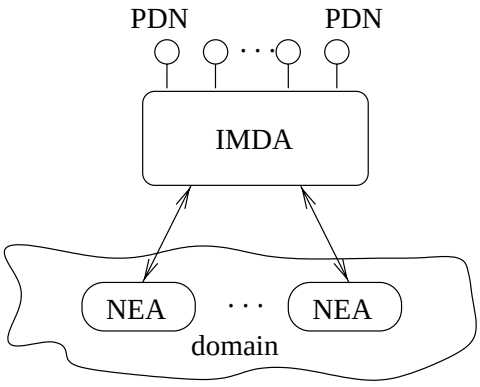

Figure 4: Illustration of an IMDA

certain type of fault. 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. Note that the PDNs should be defined carefully to reflect the most typical kinds of problems and the input-output mapping here is a kind of pattern recognition problem.

- Functions: Basically monitoring and FDC.
  - Monitoring, includes two closely integrated parts:
    - Data gathering (from the MIBs): Periodically, the IMDA will poll the MIBs for operation information and execute pattern classification. The alarms, initiated by the NEAs, are also accepted and they will in general trigger a process of classification.

* Learning the "normal" behavior: In order to decide whether or not there exist fault(s) and if so, of what type, there must be some form of *internal representations* of the MIB variables' *expected* behavior with which the comparisons can be made. Such representations are usually referred to as system (behavior) models, and hence we call our diagnosis system *model-based*. The system model can be set up in various ways, such as AR modeling or neural networks, etc. It is one of our objectives to develop an efficient representation of the system behavior, which can fit well the current given data, generalize well the unknown data and predict well the future behavior. To make such a model adaptive, change detection will also be considered.

- Fault Detection and Classification: as described before in section 3.2.

3.3.2 Function Definition—IDTA

The IDTA is located above IMDA and acts as the trouble-shooter for the symptoms reported from the IMDA. 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 5.

- Input: The activation status of the PDNs.

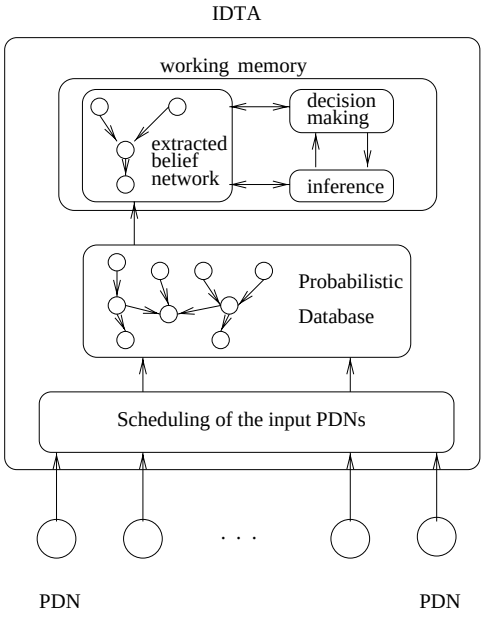

Figure 5: Illustration of an IDTA

- Output: Primary causes and the suggested test sequence.
- Functions: The IDTA functions include scheduling of the PDNs, extraction of the sub-belief network (model construction), inference and trouble-shooting, and they consist of a trouble-shooting cycle.
  - 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 defined in the introduction to belief networks in section 2.2.2. The nodes extracted are those that are not d-independent of the selected PDNs. One such example can be found in [7].
  - 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.

- 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 **pro-diagnosis** 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.

- Knowledge engineering considerations:

  The overall probabilistic expert system (belief network database) can be constructed as follows:

  - Identifying the most typical fault types as the problem definition nodes.
  - For each PDN, construct a belief network for it. The structure is relatively easy to get from the experts. The weights can be firstly set up by the experts and then get validated by the statistical data.
  - Combine the individual belief network into a large belief network by joining, aggregating and deleting nodes.

  The division into domains makes the local probabilistic database manageable and thus we don't have to wait for long for the model construction. The constructed model is also expected to be tractable. Note that a new PDN might be added on-line some time, but this should be done very carefully.

3.3.3 Function Definition—ICA

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. The ICA is illustrated in Figure 6.

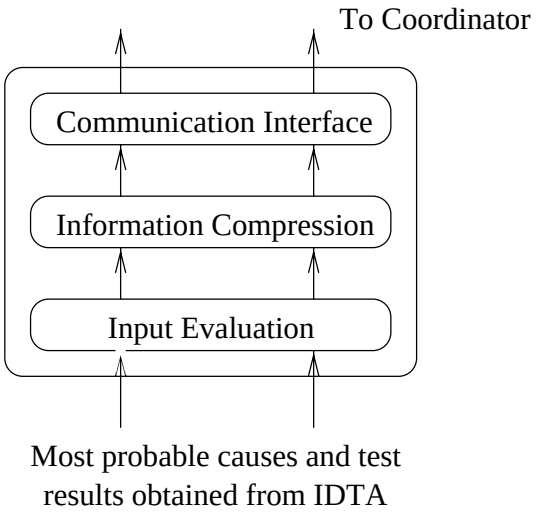

Figure 6: Illustration of an ICA

- 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.

4 Problem Formulations

In this section, we include the mathematical problem formulations for the IDTA. The IMDA and ICA parts will be discussed in separate papers.

4.1 Problem definition

We assume here that at least one of the PDNs is observed to be in an "abnormal" state and the PDNs for which a belief network will be extracted have been selected by some scheduling algorithm. There might be multiple such PDNs.

For the extracted belief network $B = ((V, L), \mathcal{P})$ with N nodes, V is a finite set of nodes, $V = \{V_1, \ldots, V_N\}$; L is the set of links and $\mathcal{P}$ denotes the associated Conditional Probability Tables (CPT). We have the following definitions:

- A time-step set $T = \{1, 2, \ldots, N \Leftrightarrow p\}$.

- Problem definition nodes set $D = \{d_1, \ldots, d_p\}$ with each element in a kind of "abnormal" state. There are altogether p PDNs, which assume binary values.

- Evidence set E: Serves as the evidence in the probabilistic reasoning. Initially, we let $E = D$. As the diagnosis progresses, new elements will be added into the evidence set. The elements in E assume binary values.

- Candidate set U: The set containing the faulty node(s), or all of the nodes in V except those already in the evidence set, namely $U = V \backslash E$. Initially, let $U = V \backslash D$. The elements in U assume binary values.

- A set of actions $A = U \bigvee \{STOP\}$. The possible actions are: the next candidate to choose or STOP. The set A will change as the diagnosis progresses and more evidence accumulates.

- Faulty node set F: The set of the faulty non-PDN nodes diagnosed. Initially, $F = NULL$. In single fault cases, $F = a$, where a is the node tested to be faulty. In case of multiple faults, F may contain more than one nodes. Here we assume the occurrence of multiple faults are independent of each other.

- Given the current state of the evidence set E, we can execute the *backward* inference to get for each node in the candidate set the probability of being faulty: $P_i = Pr\{u_i = 1|E\}, \forall u_i \in U$, and it is easy to observe that $\sum_i P_i = 1$.

- History process: For an action sequence $\{a_1, \ldots, a_k\}, k \leq N \Leftrightarrow p$, we define the observation sequence $\{Z_{a_1}, \ldots, Z_{a_k}\}$, each element of which indicates whether or not the corresponding candidate is faulty or not. If a candidate l is faulty, then $Z_l = 1$, otherwise, $Z_l = 0$. The history process is defined as follows:

$$I_k = (Z_0, (a_1, Z_{a_1}), \ldots, (a_k, Z_{a_k}))$$

It follows that $I_k = (I_{k-1}, (a_k, Z_{a_k}))$ and we say the history process $\mathcal{I}$ is Markov. We take the history process as the state process.

- Limited observations: The observations and tests are constrained within the extracted belief network.

- Cost function: There is an immediate cost incurred by selecting an action a. If $a = STOP$, the cost is zero. For any $a \in V \backslash D$, the non-PDN nodes, define the cost function as $c(a)$. Note that $c(a)$ is just the cost of testing node a and it is fixed throughout and independent of the actual faults. The determination of a cost function might include many considerations like labor cost and time factors, etc.

  Observe that different nodes may be at different **urgency** levels when faulty, so a good test sequence should also consider the urgency information. Here, we define the urgency as q, which assumes values from q_1, q_2, q_3 in ascending urgency level. q_1, q_2, q_3 can be determined *ad hoc* for different problems.

  We distinguish the non-PDN nodes according to observability and repairability. Suppose we have perfect information at this time, i.e. all

of such nodes are observable (the partially observable cases will be discussed elsewhere). We then divide the nodes into two sets: repairable set $\mathcal{R}$, and non-repairable set $\mathcal{N}$. Usually for $a \in \mathcal{R}$, $b \in \mathcal{N}$, we have $c(b) << c(a)$, which means the cost of observing only will be much less than that of both observing and repairing. In order to minimize the total cost, one would not be willing to take the risk of choosing the repairable nodes first, if it is not necessary to do so. It is our plan to take advantage of the cheaper nodes to lead to **disbelief** of those expensive nodes and thus help reduce the total cost.

The goal of an IDTA is to find the possible causes and generate the test sequences in a cost-efficient manner. In the next couple of sections, we propose strategies, both static and dynamic, to achieve this goal.

4.2 Static strategy—single fault assumption

By a *static* strategy we mean a strategy which is generated once based on the original evidence. No updates occur during diagnosis.

First, let's begin with the simplest case, static strategy for a single fault. By assuming a single fault, the diagnosis can stop as soon as we find a faulty node.

For a test sequence $\{1, 2, \ldots, j, k, \ldots, n\}$ with $k = j + 1, n = N \Leftrightarrow p$, the probability that the jth candidate have to be tested is the probability that none of its predecessors have failed the tests, namely $1 \Leftrightarrow \sum_{i=1}^{j-1} P_i$, or $\sum_{i=j}^{n} P_i$, the probability that the faulty node is either the jth candidate or

its successors. So the expected cost is:

$$EC_1 = c_1 + c_2 \sum_{i=2}^{n} P_i + \ldots + c_j \sum_{i=j}^{n} P_i + c_k \sum_{i=k}^{n} P_i + \ldots + c_n P_n$$

If we exchange k and j, then we get another test sequence $\{1, 2, \ldots, k, j, \ldots, n\}$, for which the expected cost is:

$$EC_1 = c_1 + c_2 \sum_{i=2}^{n} P_i + \ldots + c_k [\sum_{i=j}^{n} P_i + P_j] + c_j [\sum_{i=k}^{n} P_i \Leftrightarrow P_k] + \ldots + c_n P_n$$

The difference between the above costs is: $EC_1 \Leftrightarrow EC_2 = c_j P_k \Leftrightarrow c_k P_j$, and it is straightforward that

$$EC_1 \leq EC_2 \Leftrightarrow c_j P_k \leq c_k P_j \Leftrightarrow c_j / P_j \leq c_k / P_k, \forall P_i > 0$$

We see that EC_1 is cheaper than EC_2 if and only if the c/P value for candidate j is less than that for candidate k. Thus, any strategy with an element that has higher c/P value than its successor can be improved upon by simply exchanging the two elements. So for an optimal strategy, all elements must be in non-decreasing sequence of c/P values, see also [16][21]. Here is the c/P algorithm for single fault diagnosis.

- 0. Set $E = D$, $U = V/D$, $F = NULL$.
- 1. Compute the probability of being faulty for each candidate.
- 2. Observe the candidate with the smallest c_l/P_l value. Ties can be broken arbitrarily.
- 3. If the chosen node l is faulty, let $F = l$, return F and terminate. Otherwise, go to 2.

The c/P algorithm makes great sense in that it reflects the following observation: in order to minimize the total cost, people are more likely to test those more fault-prone, cheaper nodes first than the less-probable, expensive nodes.

Besides, the more urgent nodes should also be given some kind of priority and tested early. The idea behind integrating urgency is to redefine the cost function as $c(a) \leftarrow c(a)/q, \forall a \in U$. Then in the c/P algorithm, those nodes with higher level of urgency will get a lower c/P value and thus can be tested earlier. From now on, we assume this new cost definition.

In the next section, we eliminate the single fault assumption and propose a dynamic strategy with belief updating as the diagnosis progresses.

4.3 Heuristic dynamic strategy—multiple faults

Heuristically, the c/P algorithm above can be adapted to the case of multiple faults, basically as a sequential decision problem , using the following ideas:

- 1. Based on each candidate's probability (belief) of being faulty, choose the node l with the smallest c/P value to test first.
- 2. If this node l is working normally, then eliminate it from the candidate set U and add it into the evidence set E; go to 1.
- 3. If it is not working normally, then assume it is working normally and calculate the status of the problematic PDNs. If the probabilities of the PDNs to be working normally is high, then this node is the only fault and we can put it into F and terminate; otherwise, there must

be multiple faults. We put node l in F, update the fault beliefs, go to 1 and continue to find other faults. We call this process dynamic since the new round of diagnosis is based on the updated beliefs gained from the previous round of diagnosis.

It is the belief updating that changes the faulty probabilities of those "expensive" nodes and thus, hopefully, reduce their opportunities of being tested early. Note that in such an algorithm, the candidate set U (or the action set A) is diminishing in terms of number of elements.

The following is our algorithm for multiple faults.

- 0. Set $E = D$, $U = V \backslash D$, $F = NULL$. Set sequence number $SN = 1$. Set indicator $found = 0$;.
- 1. While($SN \leq n$) {
  Calculate for each candidate the probability of being faulty given the evidence (backward inference), namely, $P_i = Pr\{U_i = 1|E\}$.
- 2. Choose from the current candidate set U the action(candidate) u with minimum $c(u)/P_u$.
- 3. Test the chosen candidate u
  - (i) If $Z_u = 0$ (not faulty)

$$U \leftarrow U \backslash \{u\}$$

$$E \leftarrow E \bigvee \{u\}, Z_u = 0$$

$$SN \leftarrow SN + 1, \text{ go to } 1$$

– (ii) If $Z_u = 1$ (faulty)

$$F \leftarrow F \bigvee \{u\}, Z_u = 0$$

$$U \leftarrow U \backslash \{u\}$$

let $Z_u = 0$ (assuming it has been repaired)

calculate for each PDN the probability of being normal

$P_{d_i} = Pr\{d_i = 0 | U, Z_u = 0\}, \forall d_i \in D$ (forward inference)

* If $P_{d_i} > \tau_i, \forall d_i \in D$, τ_i is the threshold

u is the unique faulty node

set $found = 1$, go to 5

* Otherwise

$$E \leftarrow E \bigvee \{u\}, Z_u = 0$$

$SN \leftarrow SN + 1$, go to 1

- 4. End of while($SN \leq n$) }
- 5. If ($found = 1$) return $found$ and F; otherwise, trigger the ICA for cooperation.

In the next section, we formulate the sequential decision process using Markov decision process techniques.

4.4 Dynamic strategy—Markov decision process formulation

For the sequential decision problem with state process $\mathcal{I}$ as defined before, we consider a discrete-time dynamic system whose state transitions depend on

a control, $P_{xy}(u)$. Here the control u represents just the action of choosing a node as the test candidate. The costs are accumulated additively over time and depend on the states visited and the controls chosen.

We define $J_k(i)$ as the expected cost when starting from state i and k steps remain while $J_k^*(i)$ is the minimum such cost, or the best k-step cost-to-go, $\forall k \leq n$. Such cost-to-go functions are also called J-functions. Consider first the case when there is only one stage and the optimal cost-to-go is by definition

$$J_1^*(i) = \min_{u \in U(i)} \sum_j P_{ij}(u) g(i,u,j)$$

where $g(i,u,j)$ is the immediate cost incurred by leaving state i for state j when control u is chosen. Here, $g(i,u,j)$ is simply $c(u)$. In the general case, we claim that for the **finite horizon** problem,

$$J_k^*(i) = \min_{u \in U(i)} \sum_j P_{ij}(u)[c(u) + \alpha J_{k-1}^*(j)].$$

This expression states that the k-step cost-to-go can be represented as the expected value for the sum of immediate cost and the cost-to-go of the remaining $k \Leftrightarrow 1$ steps, discounted by α. To prove this, we observe that any policy π_k for the problem with initial state i and k stages to go is of the form $\pi_k = \{u, \pi_{k-1}\}$, where $u \in U(i)$ is the control at the first stage and π_{k-1} is the policy for the next $k \Leftrightarrow 1$ stages. Thus,

$$\begin{aligned} J_k^*(i) &= \min_{u \in U(i), \pi_{k-1}} \sum_j P_{ij}(u)[c(u) + \alpha J_{k-1}^{\pi_{k-1}}(j)] \\ &= \min_{u \in U(i)} \sum_j P_{ij}(u)[c(u) + \alpha \min_{\pi_{k-1}} J_{k-1}^{\pi_{k-1}}(j)] \\ &= \min_{u \in U(i)} \sum_j P_{ij}(u)[c(u) + \alpha J_{k-1}^*(j)]. \end{aligned}$$

The state transition model $P_{xy}(u)$ is a function of the current state i, the next possible state j, the action to choose u, and P_i calculated using backward inference. The next state $j = (i, (u, Z_u))$ and the corresponding $P_{xy}(u)$ may be

- $j = j_1$: if node u is faulty, $Z_u = 1$, $P_{ij}(u) = P_u$
- $j = j_0$: if node u is not faulty, $Z_u = 0$, $P_{ij}(u) = 1 \Leftrightarrow P_u$

So,

$$\begin{aligned} J_k^*(i) &= \min_{u \in U(i)} \{P_u[c(u) + \alpha J_{k-1}^*(j_1)] + (1 \Leftrightarrow P_u)[c(u) + \alpha J_{k-1}^*(j_0)]\} \\ &= \min_{u \in U(i)} [c(u) + P_u \alpha J_{k-1}^*(j_1) + (1 \Leftrightarrow P_u) \alpha J_{k-1}^*(j_0)] \\ &= \min_{u \in U(i)} [c(u) + \alpha \sum_j P_{ij}(u) J_{k-1}^*(j)]. \end{aligned}$$

It is easy to verify that

$$\begin{aligned} J_1^*(i) &= \min_{u \in U(i)} \sum_j P_{ij}(u) c(u) \\ &= \min_{u \in U(i)} c(u) \end{aligned}$$

Thus we obtain the MDP formulation pair for IDTA, as shown below,

$$\begin{cases} J_1^*(i) = \min_{u \in U(i)} c(u) \\ J_k^*(i) = \min_{u \in U(i)} [c(u) + \alpha \sum_j P_{ij}(u) J_{k-1}^*(j)] \end{cases}$$

and this can be solved by dynamic programming.

We would like to note that it might be possible that a fault (or multiple faults) can be identified within n tests; we don't have to wait until $J_n(i), \forall i$ have been obtained. The problem is incremental by nature and it can terminate well before n steps. So it might be desirable to design the formulation more carefully and include the STOP rules.

4.5 Proposed solutions for the MDP formulation

The optimal solution is an *optimal policy* $\pi^* = \{\mu_1^*, \ldots, \mu_n^*\}$, where $\mu_k^* : \mathcal{S} \to \mathcal{A}$ for each $k \in T$, that **minimizes** the expected total costs. Ideally, we expect closed-form expression solutions using value iterations or policy iterations [4]. Reinforcement learning techniques, like Q-learning, would be appropriate to obtain the approximations for the J-functions. We propose below three algorithms for the MDP formulation described in the last section.

In a **value iteration** algorithm, each possible action is tried for a state and the outcome of the best action is recorded as the value function for the state. Such iterations continue until convergence is achieved, as illustrated below:

- 1. Initialize $J(i)$ arbitrarily
- 2. Loop until convergence
  - Loop for each state i

$$J(i) \leftarrow \min_{u \in U(i)} [c(u) + \alpha \sum_j P_{ij}(u) J(j)]$$

  - End loop
- 3. End loop

In a **policy iteration** algorithm, on the other hand, it is the candidate policies themselves, not the value functions, that are updated through the iterations. In each iteration, we evaluate for each state the value functions

under this policy and then find for each state the best action in order to improve the previous policy. Such iterations continue until no improvement is possible, as shown below:

- 1. Choose an arbitrary policy π
- 2. Loop
  - Policy evaluation: compute for each state i the value function under π,

$$J_\pi(i) = c(\pi(i)) + \alpha \sum_j P_{ij}(\pi(i)) J_\pi(j)$$

  - Policy improvement: improve the policy at each state,

$$\pi'(i) \leftarrow argmax_u[c(u) + \alpha \sum_j P_{ij}(u) J_\pi(j)]$$

$$\pi = \pi'$$

- 3. Until no improvement is possible

It has been shown that both value iteration and policy iteration algorithms can terminate with arbitrarily good policy.

Note that in either value or policy iteration algorithm, *all* of the possible actions for *each* state are computed and compared (This is called "whole-sweeping"). Even if the updating is asynchronous, as illustrated in [4], the convergence still depends on such whole-sweeping. It is easy to imagine that, for a system with large number of states and/or candidate actions, the whole-sweeping algorithms will become intractable. So they should be used only when we are pretty sure that the state space is manageable.

In our problem, suppose the extracted belief network consists of n nodes, each of which assumes a binary value. Then the state space for the IDTA to face is as large as $2^n * n!$, which will be unthinkably large as n increases. For example, $n = 4$ leads to 384, $n = 5$ leads to 3840, $n = 6$ leads to 46080..., and such increase progresses exponentially. So for the cases with $n \geq 5$, instead of doing the whole-sweeping, it is more appropriate to take some approximation algorithms. The generic **Q-learning** algorithm for our problem is described below.

- 1. Initialize $Q(i, u)$ arbitrarily
- 2. Loop
  - For each state i, choose an action u, which leads the system to state j with immediate cost $c(u)$. Thus we obtain the learning instance $(i, u, j, c(u))$. The action may be selected using $\epsilon \Leftrightarrow greedy$ strategy, as described in section 2.3.4.
  - Update the Q-value:

$$Q(i, u) \leftarrow (1 \Leftrightarrow \gamma) Q(i, u) + \gamma \left(c(u) + \alpha \min_{u\prime} Q(j, u\prime) \right)$$

- 3. Until convergence

It has been proved that if each action is executed in each state an infinite number of times, and γ is decayed, the Q-values will converge to Q^*, by which an optimal policy can be acquired.

One may point out that such conditions again imply a sort of "whole-sweeping". However, the key idea behind Q-learning is: instead of finding

the *optimal* policy, it aims to find a reasonably *good* policy at a much lower cost. Thus a Q-learning agent doesn't have to try (simulate) every candidate actions; what it does is to try a couple of actions (using an ϵ-greedy scheme, for example) at a state and select the current possible action to perform. Such an action leads to the next state, where the same procedure will be repeated, and so on. So in this sense we say Q-learning entails only a *partial* sweeping of the state/action space.

Note that the above algorithm is based on the look-up table representation of the Q-values. In cases of large state spaces and/or action pools, such a table might require large amount of memory and/or storage space. So it is more desirable to use a compact representation scheme in such cases.

5 Conclusions

In this report, we describe an integrated, distributed fault management system for communication networks. The architecture is hierarchical and distributed, with probabilistic networks as the framework for knowledge representation and evidence inferencing. For fault identification and correction, mathematical formulations are presented and trouble-shooting strategies, both static and dynamic, are proposed.

References

[1] J. S. Baras, M. Ball, S. Gupta, P. Viswanathan and P. Shah, "Automated Network Fault Management", *MILCOM'97* , Monterey, CA, November 2-5, 1997

[2] W. R. Becraft and P. L. Lee, "An integrated Neural Network/Expert System Approach for Fault Diagnosis",*Computers Chem. Engng.* , vol. 17, no. 10, pp. 1001-1014, 1993

[3] A. Benveniste, M. Metivier, and P. Priouret, *Adaptive Algorithms and Stochastic Approximations,* Springer-Verlag, 1990

[4] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. I and II* , Athena Scientific, Belmont, MA, 1995

[5] D. P. Bertsekas, and J. N. Tsitsiklis, *Neuro-Dynamic Programming,* Athena Scientific, 1996

[6] J. Binder, D. Koller, S. Russell, K. Kanazawa, " Adaptive Probabilistic Networks with Hidden Variables." *Machine Learning*, in press, 1997.

[7] 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

[8] J. S. Breese and D. Koller, "Bayesian Networks and Decision-Theoretic Reasoning for Artificial Intelligence", Tutorial for AAAI'97, 1997

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

[10] E. Charniak, "Bayesian Networks without Tears", *AI Magazine,* 12, 4 , pp. 50-63, 1991

[11] J. Chow and J. Rushby, "Model-Based Reconfiguration: Diagnosis and Recovery", Tech. Rep. 4596, NASA Contractor Report, 1994

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

[13] G. F. Cooper and E. Herskovits, "A Bayesian Method for the Induction of Probabilistic Networks from Data", *Machine Learning,* 9, 309-347, 1992

[14] P. Dagum and M. Luby, "Approximately probabilistic reasoning in Bayesian belief networks is NP-hard", *Artificial Intelligence,*pp. 141-153, 1993

[15] N. Friedman, M. Goldszmidt, D. Heckerman, S. Russell, " Where is the Impact of Bayesian Networks in Learning? ." In Proc. Fifteenth International Joint Conference on Artificial Intelligence, Nagoya, Japan, 1997.

[16] D. Heckerman, J. S. Breese, and K. Rommelse, "Decision-Theoretic Troubleshooting", *Communications of the ACM,* vol. 38, pp. 49-57, 1995

[17] D. Heckerman, "A tutorial on learning with Bayesian networks", Microsoft Research Technical Report MSR-TR-94-09, 1996

[18] C. S. Hood, and C. Ji, "Probabilistic Network Fault Detection", *Global Comm. ,* pp. 1872-1876, 1996

[19] 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

[20] F. V. Jensen, *An Introduction to Bayesian Networks*, UCL, 1997

[21] J. Kalagnanam and M. Henrion, "A Comparison of Decision Analysis and Expert Rules for Sequential Diagnosis", in *Uncertainty in Artificial Intelligence 4,* (Amsterdam, The Netherlands), pp. 271-281, Elsevier Science Publishers B. V., 1990

[22] 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

[23] 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,*Series B 50 (2), pp. 157-224, 1988

[24] T. Magedanz, K. Rothermel, S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?"

[25] J. Pearl, "Fusion, Propagation, and Structuring in Belief Networks", *Artificial Intelligence,* vol. 29, pp. 241-288, 1986

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

[27] J. Pearl, "Bayesian Networks", to appear in *MIT Encyclopedia of the Cognitive Science* , 1997

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

[29] S. Singh, D. P. Bertsekas, "Reinforcement Learning for dynamic channel allocation in cellular telephone systems", submitted to NIPS 96, section: Applications

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

[31] P. Stone, M. Veloso, "Multiagent System: A Survey from a Machine Learning Perspective", Technical Report, Computer Science Department, Carnegie Mellon University, 1997

[32] R. S. Sutton, "Learning to Predict by the Methods of Temporal Differences", *Machine Learning*, vol. 3, pp. 9-44, 1988

[33] G. Tesauro, "Practical Issues in Temporal Difference Learning", *Machine Learning*, vol. 8, pp. 257-277, 1992

[34] G. Tesauro, "Temporal Difference Learning and TD-Gammon", *Communications of the ACM*, vol. 38, no. 3, pp. 58-68, 1995

[35] S. Thrun, "The role of exploration in learning control", in: D.A. White and D.A. Sofge, eds., *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, pp. 527-559, Van Nostrand Teinhold, New York, 1992

[36] J. N. Tsitsiklis and B. V. Roy, "An Analysis of Temporal-Difference Learning with Function Approximation", Technical Report LIDS-P-2322, Laboratory for Information and Decision Systems, MIT, 1996

[37] P. Viswanathan, "Automated Network Fault Management", Master's Thesis, CHSCN M.S. 96-3, University of Maryland, 1996

[38] C.J.C,H. Watkins, P. Dayan, "Q-learning", *Machine Learning*, 8, pp. 279-292, 1992

[39] M. Wooldridge, N. R. Jennings, "Intelligent Agents: Theory and Practice", submitted to *Knowledge Engineering Review* , 1995

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