VTT INFORMATION

RESEARCH REPORT TTE1-2001-37

INMAN

Intelligent Modeling Techniques in Telecommunications Network Management

Version 2.0

17.1.2002

Mikko Hiirsalmi

Version history

Version	Date	Author(s)	Reviewer	Description
1.0	31.10.2001	Mikko Hiirsalmi		First version
2.0	31.12.2001	Mikko Hiirsalmi		Final version.

Contact information

Mikko Hiirsalmi
VTT Information Technology
P.O. Box 1201, FIN-02044 VTT, Finland
Street Address: Tekniikantie 4 B, Espoo
Tel. +358 9 4561, fax +358 9 456 4537
Email: Mikko.Hiirsalmi@vtt.fi
Web: http://www.vtt.fi/tte/

Last modified on 17 January, 2002
H:\EUDORA.PRO\liitteet\rr0137-inman.doc

UDK
Key Words Bayesian network, Decision-theoretic troubleshooting, Network management

Abstract

As the size of communication networks keeps on growing, with more subscribers, faster connections and competing and co-operating technologies and the divergence of computers, data communications and telecommunications, the management of the resulting networks gets more important and time-critical. More advanced tools are needed to support this activity. In this report we have reviewed the published articles for the main application areas of intelligent methods in the domain of communications network management. With intelligent methods we mean various knowledge based reasoning methods capturing the domain expertise and also methods capable of learning from past experiences using statistical, data mining or artificial neural network methods and combinations of these methods. In the study we have found that intelligent methods have been most often applied to solve the alarm correlation task where the main target is to filter redundant data from the overwhelming alarm data stream in order to make the information understandable. Some work has also addressed repair management of faults combining fault diagnosis and repair scheduling. Other application areas cover proactive fault detection and the use of intelligent network management agents.

As better integration of alarm correlation, fault diagnosis and repair activity management still seems to be a problem, the applicability of decision-theoretic troubleshooting and Bayesian probabilistic reasoning has been studied more closely in this report. At first we describe the repair management needs. Then decision-theoretic troubleshooting theory is reviewed. Previous work using these methods in alarm management is also reviewed. We have modeled three small problems using Bayesian networks and described example troubleshooting sessions with these models to demonstrate the applicability of the models. A maintenance support terminal concept has been sketched to help the maintenance personel with decision-theoretic recommendations.

.

Contents

1 Introduction

As communication networks and distributed processing systems are gaining importance their reliable operation is getting vital to businesses. Networks are getting bigger and more complex. Communication networks have a legacy problem as they have been built over a long time and consist of very heterogeneous equipment which has to work together. Public telecommunication and various cellular phone networks serve as examples of telecommunications networks. Local-area networks and backbone networks serve as examples of data communications networks. These two types of networks are now rapidly converging so that voice data may be communicated over the Internet protocol (VoIP) and data, images and multimedia may be transfered over the telecommunications networks.

Network management is gaining importance. Previously each equipment manufacturer created management systems for their own equipment but in the last decade network management standards (called TMN = Telecommunications Management Network) were specified making it easier to build unified network management systems for heterogenous equipment. These standards define five functional areas of network management (FCAPS):

- Fault management
- Configuration management
- Accounting management
- Performance management
- Security management.

Fault management consists of monitoring, detection, diagnosis and correction of anomalous events that occur in the network. Currently used systems mainly deal with monitoring the events (alarms and state changes) generated by the underlying equipment. Most systems also provide tools to perform alarm correlation in order to filter the most relevant information from the generated events. However, troubleshooting and repair are still broadly done manually by maintenance staff. Typically trouble tickets are generated by the network management system to the maintenance staff who has the duty of correcting the problems.

Configuration management deals with maintaining a model of the network, planning for changes (extensions) and implementing them.

Accounting management deals with keeping an accurate record of the various charges for the services. It takes care of charging the customers and of distributing the payments to the associated partners.

Performance management deals with monitoring the performance of the network and identifying deviations from normal behaviour and detecting bottlenecks in the design.

Security management takes care of the security administration of the network. It could involve intrusion detection and fraud detection.

In modern network elements network management information is increasingly stored in a distributed manner locally with the network elements into MIB databases. These databases contain all relevant configuration data and the dynamic state data (measurements and alarms) in a standardized format.

An effective fault management system should meet the following requirements [Haj00a]:

1) It should scale up well to growing networks.

2) It should perform non-intrusively.

   The management activity should not interfere with normal operations of the network. It must only intervene when necessary. Excessive polling wastes bandwidth that could be used for other important services.

3) It should be robust.

   Management applications should be able to perform even when the network is not fully operational as management is mostly needed in abnormal situations, e.g. when connections are broken.

Simple Network Management Protocol (SNMP) provides an example of a network management system based on a platform-centric approach. Its organizational model specifies that the active entities in the network are divided into two types: Network Management Stations (NMS) and SNMP agents. NMS is typically a stand-alone machine running the management application. The SNMP agent is a software process that runs on all managed resources and maintains a set of parameters that reflect the state of the resource. The SNMP information model specifies that the managed resources are abstracted as managed objects. Each of these holds the parameters necessary to the management functions. Internal details are hidden. The SNMP communication model specifies that the NMS may query state of variables (GET-messages) and set variable values (SET-messages). SNMP agents notify unusual events using the TRAP capability.

NMS handles all the monitoring and decision-making tasks and the SNMP agents are just simple software processes that perform very limited processing on the MIB. This becomes difficult with the increasing sizes of the networks. The NMS has to poll the agents frequently for network statistics which consumes bandwidth and computation time for the transfer and processing of the data mass. Fault management is especially needed when the normal operation of the network is disabled. Local autonomous recovery processing capability would be desirable. This is a problem with SNMP as all the decisions are done centrally in the NMS – local activity is not supported.

2 Research directions in applying intelligent techniques to network management

Many rule-based systems, knowledge-based frame systems and expert systems shells have been used to develop prototype fault diagnosis systems in the 1980's and early 1990's also in the network management domain. Often the execution performance was considered a problem especially in domains dealing with high rates of events.

As alarm correlation has been found to be a very useful application for helping the operator to make sense of a storm of related alarm events and it is easier than complete fault diagnosis many systems have been developed to ease this task. Some model-based reasoning systems have been developed for fault diagnosis.

Recently proactive fault detection systems have been developed trying to predict developing problems in time to make adjustments in order to correct the situation before systems turn inoperable.

As network management domain naturally consists of a distributed environment agents technology has been developed to distribute control into the network elements. However, standardization mostly deals with distributed management centers.

In operational use much work still needs to be done to integrate the various, often heterogenous legacy networks and management systems together. Name space modeling and topology development needs to be done to make the alarms homogenous so that cross-domain correlation can take place.

2.1 Alarm correlation

Alarm correlation deals with reducing unnecessary alarm events and with enhancing the information content of the alarms. Such systems are especially important when real problems emerge as then a storm of events is often generated and the root-cause of failure is easily lost in the mass of events. Many approaches have been developed to perform some sort of alarm filtering or correlation. Commercial systems are also available. However, research still continues on this topic.

The modeling needed depends heavily on the subproblem being solved. Some network elements may produce transient alarms that are turned on and always cancelled after a short while. These can be filtered by a delay timer and a counter system. On the other hand, a broken cable or link may cause a huge amount of alarms coming from distant network elements that may even be controlled by separate systems. In this case the topological connections of the network need to be modeled in a suitable level (physical connection, protocol layer, application layer) in order to be able to correlate the alarms and diagnose the root cause of failure.

2.2 Decision-theoretic sequential troubleshooting

Early diagnostic systems were directed toward detecting the most likely fault hypothesis and did not directly address the problem of supporting the repair process. Recently,

attempts have been made to integrate diagnosis and repair [Frie92], [Bree96], [Jens01]. The aim is to create optimal (minimal expected cost) repair plans for the faulty system. Such plans consist of observations and component-repair actions.

The basic idea of sequential troubleshooting is to allow a repair agent the possibility to gather further information on the state of the system (called observations) or to perform repair actions of the components most likely to be broken. Decisions are made one at a time and between the decisions the agent has the chance of observing the environment (monitoring new evidence, observing the effect of the repair actions and making new decisions). Each action is considered to cause costs (measured in money or time) and one has to try to determine an optimal sequence of actions.

Decision models have been created for situations where we know the potential faults and have been able to assign failure probabilities to them (conditioned on the gathered monitoring evidence) and have some observation actions available (questions of the configuration, probes, monitoring samples) and also the choice of repairing or replacing the failed components (repair costs associated).

Under various assumptions optimality criteria have been determined for sequencing the actions. Often one aims at minimizing the expected cost of repair of the system. Some papers have also suggested the importance of minimizing the risk of costs (they typically take into account the variance of the failure probabilities and repair costs) [Shay00]. However, the calculations get much more difficult.

Decision-theoretic sequential troubleshooting systems have been reported by Microsoft to assist the troubleshooting of printer problems [Heck95]. Microsoft has also created prototypes for helping with operating system problems and information retrieval in help systems (wizards). Hewlett-Packard has been co-operating with a Bayesian network tool pioneer, Hugin, and Aalborg University in creating printing system troubleshooters for HP products [Jens01].

In the domain of network management, AT&T Bell Labs has done some experiments with probabilistic reasoning [Huar96] and Saitama University in Japan has used Bayesian networks and decision theoretic troubleshooting in their network management agents [Haj00a].

2.3 Proactive fault detection

Recently some attempts have been made to create network anomaly detectors that are able to detect soft network faults (ie. performance degradations) before they lead to concrete faults or problems. This enables one to fix the problems before they actually occur [Ho99], [Hood97]. Such systems are typically based on directly observing the available measurement variables and building models for the normal behaviour of the system. Deviations from the normal behaviour can then be detected with some type of abnormality detector. In some papers it has been reported that the parameters of video and IP (Internet Protocol) traffic can be successfully modeled with certain wavelet models. Separate traffic models are typically created for different time periods (such as for each hour of a typical week) as the characteristics tend to vary according to the time of the day and the day of the week.

One could consider the Self-Organizing Map method as a potential abnormality detector once the input features have been suitably preprocessed. For example, the wavelet coefficients could be used as features for the SOM.

3 Using decision-theoretic troubleshooting in telecommunications fault management

In this chapter we describe some of our experiments with decision-theoretic troubleshooting in the domain of telecommunications system diagnosis and repair. At first we briefly describe the fault management function of TMN (Telecommunications Management Network) formalism. Then we motivate the reader on the need to integrate the alarm correlation and fault diagnosis activities with the repair activities and suggest that decision-theoretic troubleshooting provides a theoretical framework for accomplishing this. Next we introduce the reader to the decision-theoretic troubleshooting theory and terminology. In the next chapter previous work applying probabilistic reasoning and decision-theoretic reasoning in the fault management area is reviewed. Next we describe some of our experiments with the decision-theoretic troubleshooting methodology in this domain and give some guidelines on how to specify such models. Finally the gained experiences are discussed.

3.1 On the fault management function

Network fault management involves monitoring, detection, diagnosis and repair of malfunctions in the network and its control subsystem. These activities are still broadly handled manually by operation staff. There is a need to reduce operations cost by using management software to simplify and automate the management tasks.

A fault is an abnormal operation that significantly degrades performance of an active entity in the network or disrupts communication. All errors are not faults as protocols can mostly handle them. Generally faults may be indicated by an abnormally high error rate.

In fault management different error rates and measures are commonly stored in MIB (Management Information Base) records for each managed object. Typically these values are scalar valued counters. MIB variables are often redundant partly measuring similar behaviour from slightly different perspectives. Not all variables need to be tested in order to detect abnormalities.

An observation is a test over MIB variables. It may be based on the nominal values of the MIB variables or a function may be applied to the variable, like taking the time difference in order to gain the rate of change over time of the counter.

An alarm is an event generated asynchronously whenever the value of some quality indicator crosses a predefined threshold (either positively or negatively).

Detection mechanisms are usually implemented real-time and have been embedded with the network elements (network protocols and devices). Finite-state machines may be used to detect protocol errors. Fault localization is typically supported by algorithms that compute a possible set of faults. Fault identification involves testing the hypothetical faulty components. Repair is achieved by taking corrective actions.

The diagnosis process is sequential by nature. At any stage there are many observations, tests and repairs that can be applied. It is useful to take advantage of the information

gathered on the way by testing propositions that might reduce diagnosis costs or give us higher confidence in potential failures.

3.2 Motivation to apply decision-theoretic troubleshooting

In current alarm management systems the main focus is on detecting the most likely fault candidates and generating trouble tickets (repair orders) of these faults for the maintenance staff to repair. The large volume of alarm messages is filtered using alarm correlation tools that are typically based on detecting commonly occurring patterns using rule bases and combining these alarms into more refined fault hypothesis. To our knowledge no knowledge based support is provided for actually repairing the network. The maintenance persons only have the trouble tickets at their disposal and many persons may do partial tests to gather further information or to test possible repair actions during the correction process before actual system recovery takes place. If the initial alarms are falsely interpreted the correction of the faults may take a long time and involve many test and repair operations by multiple persons.

We claim that it would be useful to combine alarm correlation and fault diagnosis with the repair operations. By doing this a list of possible fault hypothesis could be ordered by their likelihood and the supporting evidence could be made available also to the maintenance personel via a troubleshooting guidance system. This system could also contain knowledge about suitable repair operations and the associated cost estimates (based on the needed time, money, supplies etc.) for the different fault types. The support system could suggest the most cost efficient repair and observation actions to follow based on all the available evidence. We believe that such support leads to more efficient repair sequences, harmonizes the repair operations, supports knowledge transfer between the involved personel and could be used for gathering feedback on the success of the applied repair operations.

Decision-theoretic troubleshooting provides a probabilistically oriented framework for building the core operations outlined above. Its potential is explored in the following chapters.

3.3 Introduction to the Value-of-Information (VOI) principle

While making decisions under uncertainty there is a quest for more information to reduce the uncertainty. Gathering and accessing information is usually not free but creates costs. Therefore one would like to be able to evaluate beforehand whether it is worthwhile to consult an information source and in which order should the data sources be consulted. This data request problem has been studied formally in decision theory where utilities of possible actions are used to guide the decisions. Also utility-free measures exist and include Shannon's measure of mutual information and variance.

As the range of possible actions to perform is often large one has to set a bound to the search space. Usually myopic data requests are used where one at each stage searches for the best next action to perform. Also the search may be reduced by evaluating the state in terms of the probability distribution of a hypothesis (the variable of interest, the one that one needs to have accurate knowledge of). [Jens94]

The expected value of performing test T is:

$$EV(T) = \Sigma_{t \in T} V (P(H \mid t) P(t)$$

The expected benefit is:

$$EV(T) = EV(T) – V(P(H))$$

The expected profit is:

$$EP(T) = EB(T) - C_T$$

V = value function that maps the given probability to a real-numbered value.
C_T = the cost of accessing information source T.

Different value functions may be used. Entropy of a distribution over H is defined as:

$$ENT (P(H)) = - \Sigma_{h \in H} P (h) \log_2 (P(h))$$
$$V(P(H)) = - ENT (P(H))$$

Entropy measures how much the probability mass is scattered around on the states. When H has n states and the distribution is even the entropy gets its maximum value $\log_2 n$. When the probability mass is concentrated in one state the entropy is 0.

When the states of H are numeric a variance measure may be used. Small variances are preferred.

$$V(P(H)) = - \Sigma_{h \in H} (h - \Sigma_{h' \in H} h' P(h'))^2 P(h)$$

In some cases the value of the hypothesis is due to a set A of actions which are taken dependent on the state of the hypothesis. The expected utility is then:

$$EU(a) = \Sigma_{h \in H} U(h,a) P(h|a)$$
$$V(P(H)) = \max_{a \in A} EU(a)$$

Also other types of value functions have been studied.

In some cases the myopic approach is not enough and one should consider more than one information request at the same time. The search space gets exponentially larger and typically this is tried only after the myopic approach has failed to suggest any actions and often a simplifying assumption of the actions being conditional independent on each other given the hypothesis.

3.4 Introduction to decision-theoretic troubleshooting

In traditional diagnostic systems the main objective has been to find out the set of fault hypothesis that is consistent with the observations and arrange these hypothesis according to their likelihood. In these systems the repair process has usually been assumed to be a simple two-phase process where one is first charged with finding the most plausible diagnosis candidate and then repairs the system according to these diagnostic results. Attention has later turned to integrating diagnosis and repair together in order to restore

the system to its functioning state as cost efficiently as possible [Frie92]. An alternating sequence of diagnosis and repair actions is needed to accomplish this task. This integration task has been approached using both logical model-based and decision-theoretic cost-based approaches.

In [Frie92] a model-based diagnosis and repair process has been defined and a temporal framework has been introduced where one models the system components, faults, observations and repair actions in a logic-based formulation. Multiple system purposes may be defined, i.e. for emergency recovery and for regular maintenance. Different recovery actions may be selected to restore each purpose based on the same system formulation. The sequencing of the actions uses utility computations based on costs of downtime and costs of observation and repair actions.

Decision-theoretic formulations for selecting diagnostic tests and for guiding the repair process have been formulated in [Kala90], [Heck95], [Huar96] and [Hajj98]. The primary objective of the troubleshooting session is to repair the device not just determine what is wrong. The main use of these formulations is to provide the diagnostician help with troubleshooting the system by providing him with a recommendation list including the most cost efficient actions to perform next. Possible actions include asking additional questions about the behaviour of the device, testing or repairing individual components or calling for outside help (typically modeled as an expensive service call). In order to use a decision-theoretic approach the failure probabilities of the components conditioned on the available evidence and the costs for observing new evidence, for testing and for repairing a component and the costs for calling service need to be defined. Bayesian network models are well suited for computing the marginal probabilities of dependent variables conditioned on the available evidence. Fast algorithms exist for performing probability propagation with relatively sparse discrete Bayesian networks. This computational boost has revived interest in decision-theoretic troubleshooting methods.

Making a single fault assumption (only one component is believed to have failed at a given time, faults are mutually exclusive) the expected cost of repairing the system (ECR) using a test sequence (c_1,..., c_k) may be formulated in the following way:

$$ECR(c_1, ..., c_k) =$$

$$C_1^o + p_1 (C_1^r + C^p)] + (1 - p_1) [C_2^o + p_2 / (1 - p_1) (C_2^r + C^p)] +$$

$$(1- p_1 - p_2) [C_3^o + p_3 / (1 - p_1 - p_2) (C_3^r + C^p)] + ... +$$

$$(\sum_{j=k+1...n} p_j) C^S$$

$$= \sum_{i=1...k} [(1 - \sum_{j=1...i-1} p_j) C_i^o + p_i (C_i^r + C^p)] + (\sum_{j=k+1...n} p_j) C^S$$

The c_i stands for the component i.
The p_i stands for the failure probability of component i.
The C_i^o stands for the observation cost of component i.
The C_i^r stands for the repair cost of component i.
The C^p stands for the observation cost of the problem defining node of the whole system.
The C^S stands for the repair cost of the whole system (or the cost of a service call).

The formula has been formed by summing the expected costs of the component repairs. In this formulation only binary states have been assumed for the nodes. The first component

is at first observed with the cost C_1^o. If the component is found faulty (with probability p_1) it needs to be repaired immediately and then the system status needs to be observed with total cost ($C_1^r + C^p$). If the component is found normal (with probability 1- p_1) the testing sequence needs to be continued with the next component C_2. This sequence is continued until the last component is tested or a service call is made (it has been modeled here to take place after testing component C_k).

The most optimal position of the service call may be identified by evaluating the minimum of the expected cost of repair including k components (k=0,...,n):

$$n_s = \min_k [ECR (c_1, ..., c_k)]$$

An optimality criterium may be formulated for the optimal order of the actions by considering a case where two actions have changed places (i=k and j = k+1). The difference between the costs is then:

$$ECR(c_1, ..., c_n) - ECR(c_1, ..., c_{k-1}, c_{k+1}, c_k..., c_n) =$$

$$(1 - \sum_{j=1...k-1} p_j) C_k^o + (1 - \sum_{j=1...k} p_j) C_{k+1}^o -$$

$$(1 - \sum_{j=1...k} p_j + p_k) C_{k+1}^o - (1 - \sum_{j=1...k-1} p_j - p_{k+1}) C_k^o =$$

$$p_{k+1} C_k^o - p_k C_{k+1}^o$$

In order for the original order (c_1, ..., c_n) to be superior (k precedes k + 1) this difference needs to be negative which leads to the following inequality:

$$p_k / C_k^o > p_{k+1} / C_{k+1}^o$$

The term p_k / C_k^o is called the efficiency of the action k. The optimal observation-repair strategy is therefore produced by ordering the components according to the decreasing efficiency.

The optimal plan can be approximated using the following algorithm:

1. Update the probabilities of the component faults given the current state of information.
2. Observe the as yet unobserved component with the highest efficiency (ratio p_i / C_i^o).
3. If the component is faulty then replace it.
4. Terminate if the device is working after replacing the faulty component.
5. Otherwise go to step 1.

This algorithm also supports repairing multiple faults but the plan may be suboptimal because a single fault was assumed while computing the efficiencies used to select the next action to perform.

Many assumptions have been made while formulating the above ECR in order to simplify the problem and to keep the complexity time polynomial in terms of the number of components:

1. Only one problem defining node (representing the functional status of the device) is allowed for a troubleshooting session.

A separate diagnostic network may be defined for different tasks or multiple problem defining nodes may be defined but only one may be instantiated during a session.

2. The device is faulty at the onset of the troubleshooting session.
   Typically this assumption is acceptable if the session is started after some abnormal conditions have been detected. If the system is being used by an automated system to do preventative maintenance the assumption may not be valid. In this case the problem-defining node should be tested before carrying out any repairs.
3. Only a single fault is assumed.
   This assumption can typically be made when the system is being monitored regularly. However, in the above discussion also multiple faults have been supported but the optimality can not be guaranteed.
4. Immediately after any component repair the problem-defining node is observed with cost C^p.
   When the cost of testing the status of the system is expensive (like in testing a jet engine) it could be better to fix multiple components at the same time before testing. This assumption is also problematic in situations where the corrective actions have a delayed impact as the status can not be observed reliably before the delay time expires.
5. Components are either observable or unobservable. A component that is observed to be abnormal must be repaired immediately.
   Unobservable components can only be repaired. They can be modeled by setting the repair cost to null ($C_i^r = 0$) and the observation cost equal to the cost of repairing the component. In the case of multiple faults the immediate repair principle may not be optimal but it is a good approximation.
6. Observations of the status of the component are assumed to be unambiguous. Also the repair operations are assumed to be reliable (they never fail).
   Perfect reliability is often difficult to achieve and retries should be allowed.
7. Costs are independent of previous repair and observation actions.
   The validity of this assumption is domain dependent. However, in many cases there are preliminary operations required that are common with many repair operations (e.g. opening a cover or having to disassemble some equipment before accessing the component) that should be also modeled.

Additional observations gathering more evidence (nonbase observations) could be allowed if the expected cost of repair is lowered based on the expected new data. In order to limit the search space one needs to make a myopic approximation where only one nonbase observation can be made before the observation-repair sequence. This decision is made after each action and therefore multiple nonbase observations may be made before repairs. The expected cost of observing o_i with background information I is:

$$ECO(I, o_i) = C_i^o + \sum_{k=1\ldots ri} P(o_i = k \mid I)\, ECR(I \cup \{ o_i = k \})$$

If $ECR(I) < ECO(I, o_i)$ for every nonbase observation o_i then no nonbase observation is made at this point of the troubleshooting session. Otherwise the nonbase observation (o_i) with the smallest $ECO(I, o_i)$ is chosen to be performed.

From a modeling point of view the assumption of independent costs for actions is problematic as in some cases the costs of the repair actions are clearly dependent. While performing such a dependent action one should also consider the possibility of performing the dependent actions at the same time. Also delays in the process may cause delays in the chance for observing the new system status. In such cases performing many actions at the same time should be considered. It seems that in the network management domain such problems are not common.

The assumptions of perfect observations and repair actions may also cause problems. It should be possible to allow some uncertainty in the status of the repaired component possibly allowing one to retry the repair action. Also the observation of the component status may have some uncertainty involved.

3.5 Previous work using probabilistic reasoning and decision theoretic troubleshooting in telecommunications alarm management

In [Deng93] the authors use Bayesian networks to fault diagnosis in a network management problem. The fault diagnosis of a linear lightwave network (LLN) is investigated. The linear lightwave network constitutes the physical layer of ACORN gigabit research network testbed. It is based on establishing controllable transparent optical paths between network users. Its key element is a controllable linear divider/combiner (LDC) which can be used to create optical paths (routes) on demand. The routes run through the LDCs from its inputs to its outputs according to its dynamically configurable routing table.

Bayesian networks are used as the knowledge base language to represent the diagnostic knowledge. The used model is simple enough to be automatically updated from the LLN configuration tables. Therefore topological changes do not require manual model changes. The incoming and outgoing signal levels are measured regularly. When abnormalities are detected in the outgoing signal levels a diagnostic session is started where one tries to identify the faulty LDC components. In order to identify these components additional (costly) power measurements can be made measuring the incoming and outgoing power levels for each LDC route. Bayesian belief propagation is used to find the most likely failed components and focusing additional tests to them. This way only part of the components need to be tested.

Much of the reported work dealt with basic Bayesian network belief propagation algorithms. Specialized decision algorithms were created to diagnose networks having t simultaneous faults (t-fault diagnosis system). More powerful belief propagation algorithms are now available than were used in this paper. The dynamic configuration ideas were interesting.

In [Huar96] a test system applying Bayesian belief networks and decision-theoretic troubleshooting to fault isolation in broadband networks has been described. During a troubleshooting session the program recommends a sequence of tests believed to be the most efficient in the current situation based on the fault probabilities conditioned on gathered evidence about the state of the network (alarms and test results). The prototype has been developed for a Gigabit testbed network, Xunet. [Fras92]

Knowledge engineering has been done together with XUNET designers and managers. Initially they identified a set of faults that occur regularly and for which the causality was known. Then individual belief networks were composed for the identified subproblems and at the last step a global belief network was contructed by combining the subnetworks appropriately. Six example problems were considered and for four of them Bayesian network models were constructed.

The guiding design principle was to identify problems with regularly occuring faults as the dynamics was sufficiently well understood for them. While modeling much of the details were ignored in order to keep the task manageable. At the next step the prior and conditional probability matrices were estimated for the subnetworks. Qualitative adjectives and adverbs were assigned to major probability intervals in order to ease the assessment task. During this work some topology adjustments were performed (splitting nodes having too many connections and clustering some other nodes). Also some consistency tests were performed. Finally the subnetworks were combined into a single large belief network containing 30 – 40 nodes.

The belief network was used to combine gathered evidence in order to update the beliefs of the fault hypothesis. A decision engine was added capable of suggesting which evidence to gather next (based on dynamic programming). During a decision-theoretic troubleshooting session the network manager iterates between querying recommendations on the best actions to perform and making decisions. The system derives a recommendation list of most useful actions to perform at the current situation. Once new evidence is gathered the situation is re-evaluated and the session is continued iteratively until a good enough fault hypothesis has been identified.

The decision-theoretic approach was evaluated by simulating the success of different decision making methods (against a heuristic troubleshooter, an omniscient troubleshooter and a static troubleshooter). The decision-theoretic approach was found to produce significant savings compared with the best static troubleshooters. The best heuristic troubleshooters (with 4-step lookahead) performed almost as good as the decision-theoretic approach in this test case.

The authors suggest that in general belief network approach is best suited for cases where the dynamics is not well understood, e.g. cases where several layers of network protocols are involved.

In [Haj00a] a framework is suggested for automating fault management using distributed software agents. The main idea is to design intelligent agents that can perform advanced reasoning activities on the local network domain. Together these agents are able to implement the overall fault management function. Bayesian network modeling is used to model the communications network domain. Each agent is capable of detecting abnormal situations, correlating the gathered evidence and selectively seeking to derive a better explanation of the alarms generated in its domain. The agents have the capability of carrying out local recovery actions.

The network intelligent agents (NIA) consist of four modules: a knowledge base, a reasoning engine, a learning engine and a communication engine. The reasoning engine determines the most likely fault causing the abnormal behaviour indicated by the gathered alarms. It is also responsible for invoking the appropriate action to repair the fault or to report it to the Network management System (NMS). The knowledge base is used to

model the managed objects, their possible errors, available measurement variables (MIB) and the associated faults. The propagation patterns of the faults are modeled using Bayesian network dependency models connecting the faults and the different fault patterns associated with different measurements. The propagation patterns are partly dependent on the dynamic physical and logical configuration of the network. It is claimed that automatic network topology discovery would be beneficial for constructing and maintaining the models. No implemented and generally applicable approaches were presented for the learning task, just suggestions for future research.

The reasoning engine is activated by alarms and the activity continues until the most likely fault has been detected and the fault has been recovered. The process involves an iterative sequence of situation assessments followed by selecting further observations as described in Figure 3.5-1.

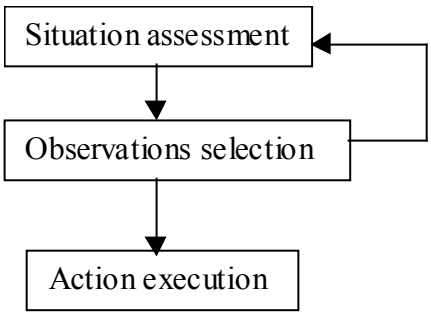


Figure 3.5-1: Reasoning cycle of the agent

Situation assessment takes care of integrating the observed evidence into an overall explanation of the state of the local network. In a Bayesian probabilistic environment this amounts to estimate the probability distribution of the different network faults given the raised alarms. The Bayesian network formalism is well suited for such inference.

The Bayesian network models of the local agents are loosely connected with the models of the neighbouring agents with as few common nodes as possible. Such a distributed inference with Bayesian networks is further studied in [Xian96].

Typically marginal probabilities of the fault hypothesis given the initial evidence are not accurate enough for identifying proper repair actions and further evidence needs to be gathered. Relevant variables need to be queried for necessary information in a goal-directed manner. Observations are selected based on their degree of informativeness.

A mutual information criterion, I(P,O), is used to select the most influential observation.

$I(P,O) = H(P) - H(P|O)$ mutual information measure

$H(P) = - \Sigma_j P(F_j) \log P(F_j)$ an entropy measure

$H(P \mid O) = - \Sigma_i \Sigma_j P(F_j \mid o_i) \log P(F_j \mid o_i)$ an conditional entropy measure

The mutual information criterion is non-negative. It becomes null when O does not contribute to any reduction in the uncertainty of the NIA. The entropy measure is largest

when all the faults are equally probable. It becomes null when one of the faults is certain (= 1). So a smaller number indicates a probabilistically better known situation.

The observations are evaluated myopically by only considering the selection of one observation at a time. The observation with the highest mutual information score is added to the list of selected observations (Θ) and then one detects the observations that are independent of F given Θ. These independent observations are deleted from the network. The search is continued until no further observations can be made.

Once a fault has been localized recovery actions can be taken locally. They can range from conventional notifications of faults to the NMS using e.g. SNMP traps to local recovery actions like a sequence a discrete changes in the states of the managed objects.

The authors claim that the described methods have been prototyped in a working prototype implementation. A trouble ticket database has been used to generate the Bayesian network models. One of the identified problems with Bayesian network modeling is the availability of experts capable of expressing the dependencies. The authors have also researched continuous network monitoring automatically detect performance degradation in IP-Networks [Haj00b].

3.6 Fault management experiments with decision-theoretic troubleshooting

3.6.1 Overview

In this chapter we describe some of our experiments with decision-theoretic troubleshooting and Bayesian network modeling in the domain of fault management. We start with guidelines for modeling using these methods. At first a Bayesian network for diagnosing TCP connectivity problems is presented. An example session of decision-theoretic troubleshooting is first presented and then a session using Value-of-Information criterion for recommending actions is presented. This first experiment deals with a local troubleshooting problem. Additionally two experiments are presented dealing with more global views of the network: the first one diagnoses microwave link problems in the GSM access network, the second deals with a dynamically configurable diagnosis model based on the routing table information. These examples have been produced to demonstrate the capabilities provided by probabilistic reasoning and decision-theoretic troubleshooting. The models have not been validated by domain experts. The experimental sessions have been carried out with a research tool called MSBNx (see Appendix A for more details). Finally we discuss the experiences and the applicability of these modelling techniques.

3.6.2 Guidelines for applying Bayesian network modeling and decision-theoretic troubleshooting

Selection of a promising domain requiring uncertain reasoning and giving a high payback for better solutions and where domain knowledge is available is the cornerstone for successful knowledge based system development. Also one should focus ones attention and only model the necessary entities in the domain.

At first one has to identify the set of relevant entities that have a clear grounding in the problem domain. These should be descriptively named to allow easy communication with

domain experts. Next one specifies discrete states for the variables – these states have to be suitable for the planned inferences. Often the used entities have a continuous valued domain. Most Bayesian network methods support only variables with discrete states. Therefore the continuous values have to be discretized in a meaningful way.

If the modeling entities are not directly available one has to define how these values can be computed for decision making. Some preprocessing may need to be done to map the available objects to the ones used in the model. New variables may be formed by functional transformations involving more than one raw measurements or by differentiating the values to get the rate of change estimates. New variables may also be assigned to past values of the entities by lagging the data with the desired amount e.g. ifInOctets(-10 mins) is the counter value 10 minutes ago. These steps need to be performed by an intermediary driver system controlling the inference.

Next dependencies between the selected variables are identified. This is usually done with a graphical graph editor by drawing directed arcs between the variables, typically following the causal order from the cause to the consequence. The causal direction is recommended as it is usually easier to define the conditional probabilities in this direction. Alternatively a causal structure learning algorithm may be used to datamine a preprocessed measurement database of the selected variables. The generated graph can be later modified manually. In order to get real computational benefits from using Bayesian network inference algorithms mutual independencies have to be identified between the chosen variables.

Finally the quantitative probability matrices (both prior and conditional ones) need to be defined. If data is available initial matrices can be estimated statistically from the data. Expert justification is typically needed. There are ways to combine the expert opinions with the data-based estimates therefore adjusting the expert opinions with data. Simplifying assumptions can be made: The parents may be considered mutually independent and the dependence of each one can be estimated separately (conditional independence of the parent nodes). Alternatively the state space may be divided into regions with decision trees by iteratively dividing the space according to the states of the parents. This is called asymmetric evaluation.

When using decision-theoretic troubleshooting one needs to estimate observation and repair costs for the repairable components and observation costs for informational nodes. Also a repair cost needs to be estimated for replacing the whole system (a service call cost). The troubleshooting engine does not understand the actual repair procedures – it makes recommendations based on the given cost estimates. All costs have to be summarized in the single number per component.

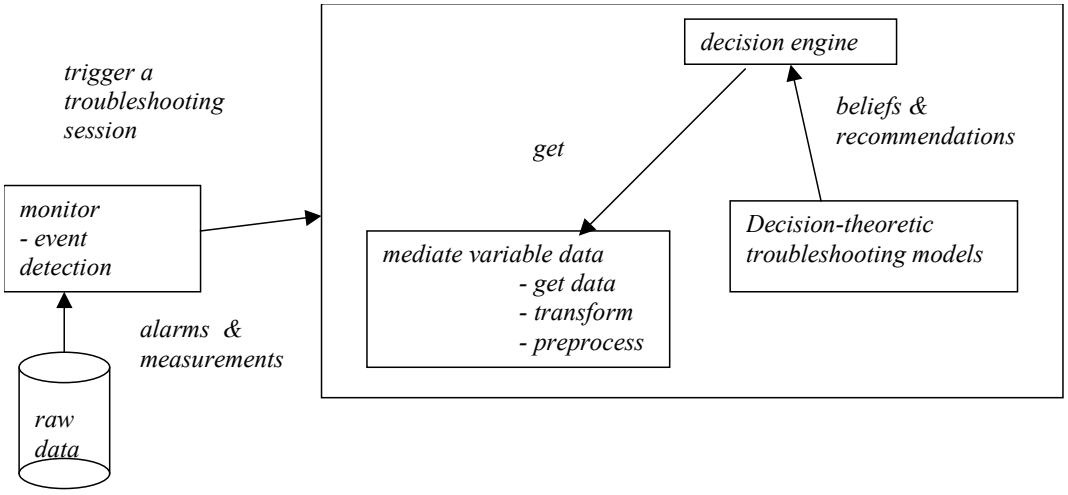


Figure 3.6.2 A schematic diagram showing how to integrate the troubleshooting engine with the controlled process.

An intermediary driver system is needed to control the decision-theoretic troubleshooting session. The session may be initiated by an event monitoring system when it detects unexplained abnormalities. A decision engine program loads the appropriate diagnostic model and starts the troubleshooting session by providing the diagnostician a recommmendation list if running interactively or by automatically initiating the best troubleshooting action if running in automated mode. A data mediator takes care of retrieving the variable states that the decision engine queries from the raw database. It takes care of the required data transformations and preprocessing. Some form of user interface is typically required to communicate with the human diagnostician. See Figure 3.6.2 for a schematic diagram of the integration.

3.6.3 A decision-theoretic troubleshooting session dealing with TCP problems

This example deals with diagnosing the sources of TCP (Transmission Control Protocol) problems in the router interface of a host. It is based on a previously published network [Haj00a]. The Bayesian network model contains four primary problem types: Line_UP (indicates whether the communication line is working), Interface_Problem (indicates whether the interface hardware has problems), BufferSize_Too_Small (indicates whether the TCP buffer is too small) and CPU_overloaded (indicates whether the CPU on the connection card has been overloaded). Most of the evidence nodes included into the model are locally stored MIB counters that count incoming, outgoing and rejected packet levels in the host interface. Instant temporal behaviour has been modeled with the rate nodes (Input_frame_rate and Delivers_datagram_rate) that measure the rate of change (time differenced) of the appropriate counter. The used MIB variable definitions have been summarized in Appendix B. The dependency structure of the Bayesian network is depected in Figure 3.6.3-1. The textual description of the used Bayesian network has been included as Appendix C.

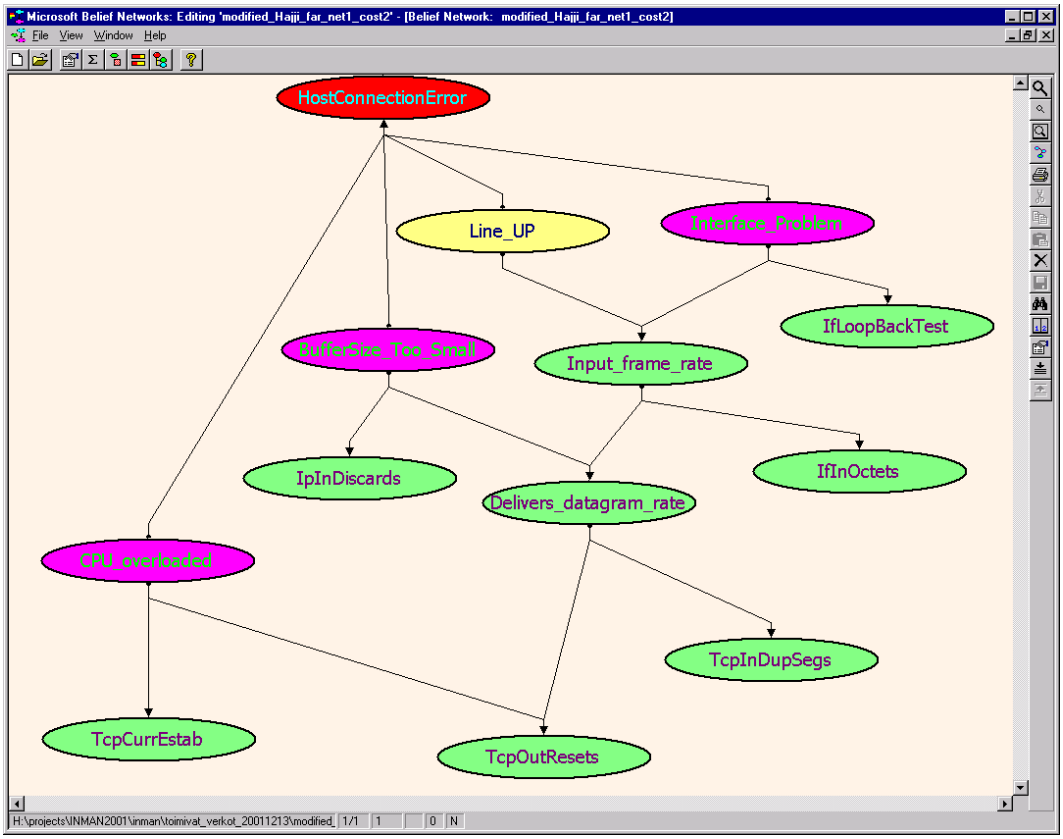


Figure 3.6.3-1. A troubleshooting Bayesian network model for TCP problems of a host interface. (In this figure, informational nodes are marked with the green/light grey color, unobservable but repairable nodes have the violet/dark grey color and repairable and observable nodes have the yellow/white color, the problem identifying node has the red/black color).

In [Haj00a] this network serves as a Bayesian knowledge base of a single network intelligent agent (NIA) which is connected to other local NIAs according to the physical configuration of the underlying local area network. This host NIA is connected to a router NIA and shares a common replicated node with that model (input_frame_rate in this model is equal to the output_frame_rate in the router model). These models could be connected into a large network model for the whole LAN but the model would still be useable and inferences could be performed efficiently as the nodes are not too densely connected.

The qualitative dependency structure of the diagram has been slightly modified to better follow the causality principle. Therefore evidence node TcpCurrEstab has been made a child of fault node CPU_overloaded.

Some modifications have also been made to conform with the MSBNx tool (see Appendix A for details) and decision-theoretic modeling requirements: All the faults have been made root nodes. A special problem defining node has been added as a child of all the possible faults. Observation costs have been specified for all the observable nodes and repair costs for all the nodes that can be fixed or replaced. Also a system level repair cost has been defined. In this example a cost of 5 units is allocated to observing most of

the MIB variables (the loopback test has a cost of 25 units) and a cost of 7 units to the observation of the rate-of-change variables. The primary faults were assigned repair costs of 70 units for the bufferSize_Too_Small, 100 units for CPU_overloaded, 200 units for the Interface_Problem and 30 units for the Line_UP problem. The observation of Line_UP status was assigned the cost of 15 units. The replacement of the whole system was assigned the cost of 1000 units.

All nodes have been assigned discrete states (mostly 2 but some have 3). An intermediary system needs to assign the continuous values to the appropriate discrete states. All the required probability matrices (the prior and the conditional probabilities) have been manually defined for this demonstration and do not reflect real beliefs tuned to expert knowledge or measurement data. Conditional independence assumption is used in the problem defining node – it is assumed that any failure can cause the problem-defining node to get into the error state. Also there is a small chance of being in the abnormal state even if the known faults are all normal (the leak probability) caused by the chance of unmodeled errors. While defining the conditional probabilities of the node input_frame_rate we have used asymmetric evaluation mode (see Figure 3.6.3-2) where we have defined that when the line is not up (Line_UP = No) the probabilities are independent of the Interface_Problem states. In the case that the line is up the probabilities are normally conditioned on the states of the node Interface_Problem. The following prior probabilities were assigned to the primary faults: for Line_UP 0.02, for node BufferSize_Too_Small 0.01, for CPU_overloaded 0.004 and for Interface_Problem 0.001.

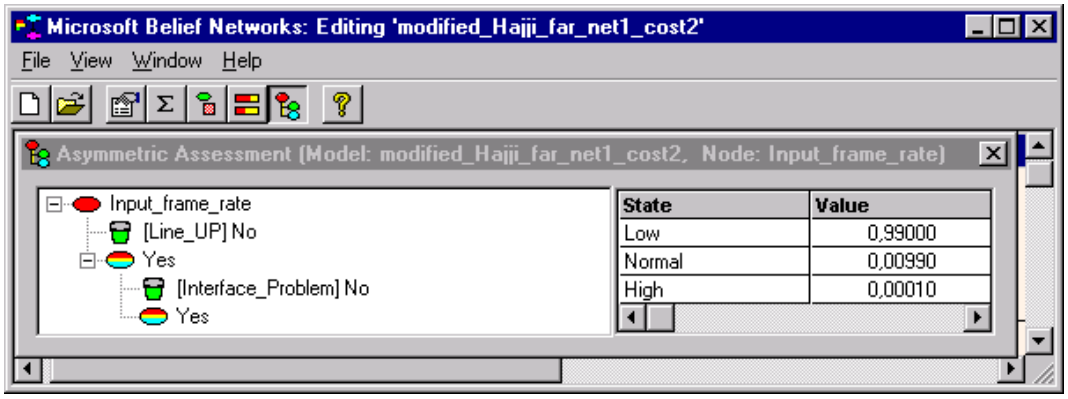


Figure 3.6.3-2. Asymmetric evaluation of the conditional probabilities when the state of Line_UP is No.

The following troubleshooting session is initiated by assigning the problem-defining node to an abnormal state (HostConnectionError = Yes). In this example session we have also added the evidence that the fraction of TCP segments being reset is high (TcpOutResets = High). In Figure 3.6.3-3 the marginal probabilities of the different nodes are shown given the evidence. One can see that the fault probabilities are still very small. In this phase the CPU_overloaded is the most probable fault.

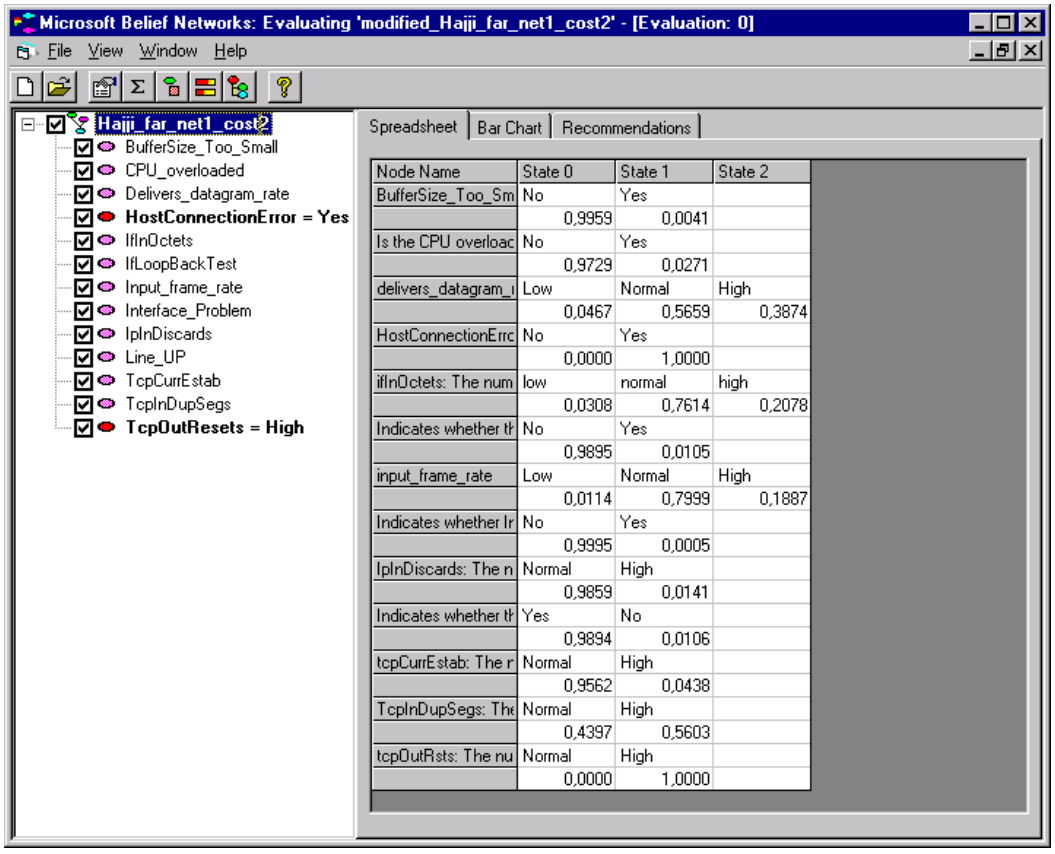


Figure 3.6.3-3. Marginal probabilities for the nodes given the assigned evidence.

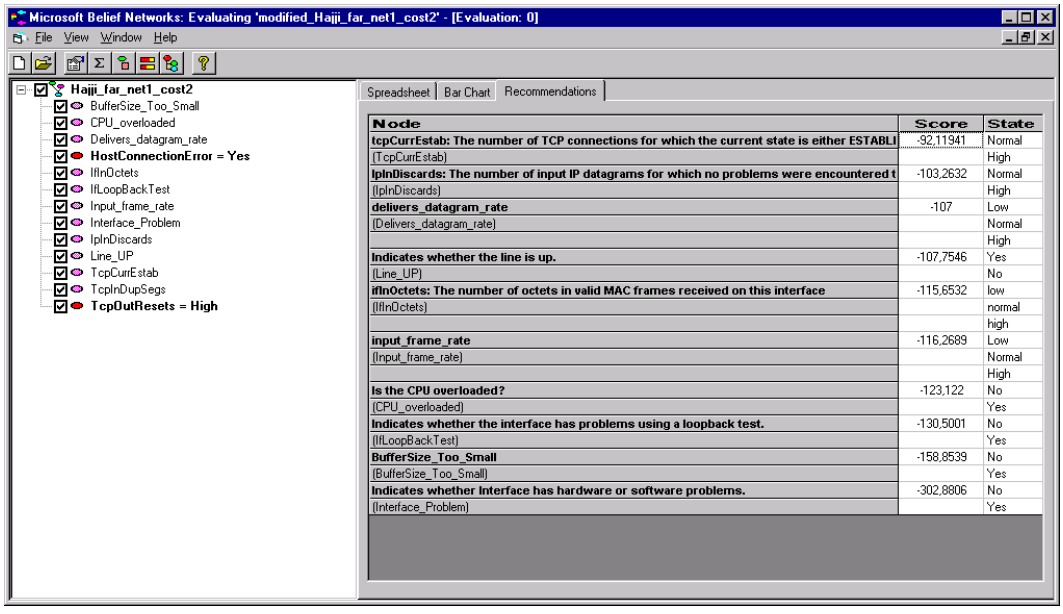


Figure 3.6.3-4. A recommendation list of actions to perform next.

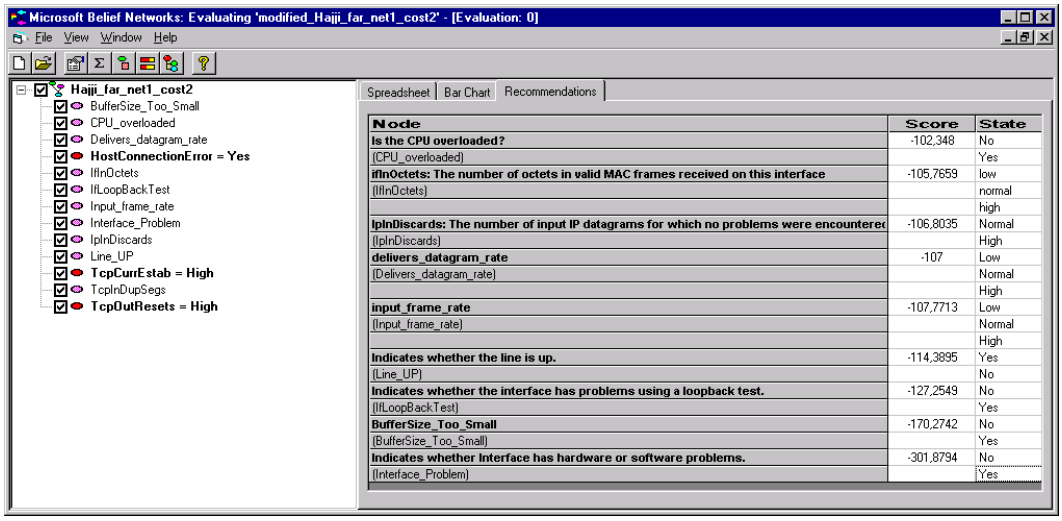


Figure 3.6.3-5. Marginal probabilities for the nodes given the new evidence (TcpCurrEstab = High).

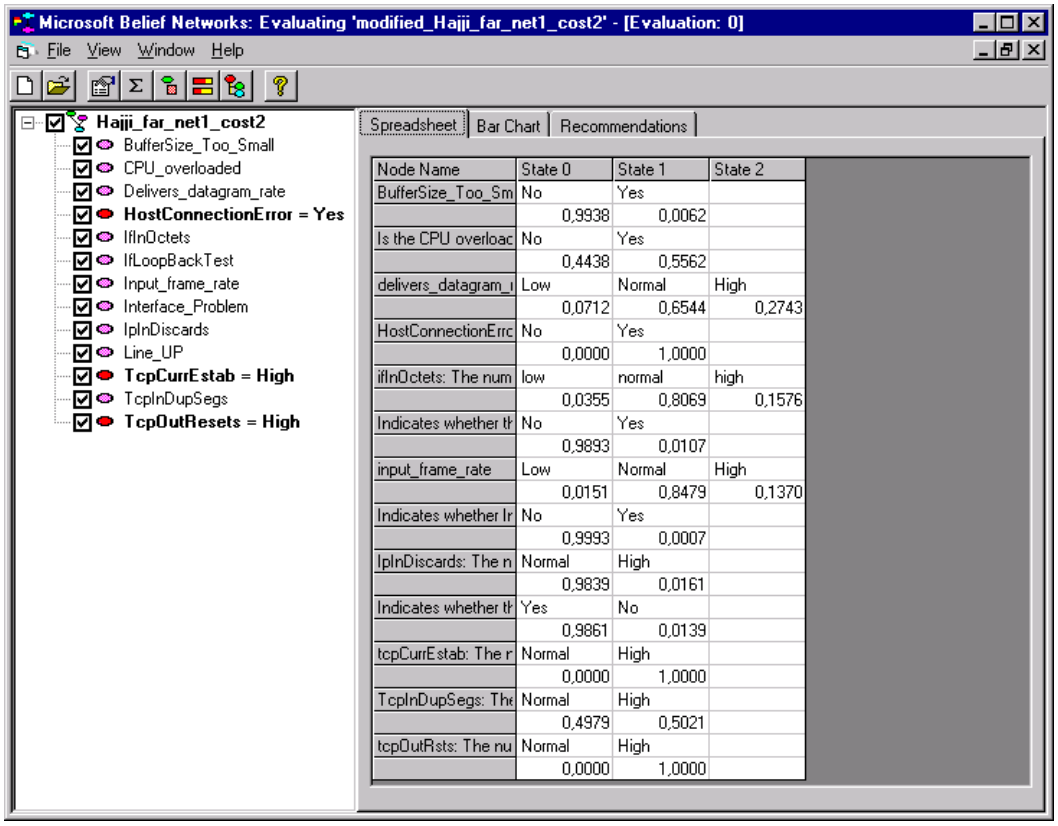


Figure 3.6.3-6. A recommendation list of actions to perform next given the new evidence (TcpCurrEstab = High).

In Figure 3.6.3-4 the most cost efficient actions to perform next are listed. It is suggested that one should first check the status of node TcpCurrEstab as it gives much information on the fault hypothesis CPU_overloaded. The suggested node is inspected and its state is found to be High. In Figure 3.6.3-5 we see that the next recommendation is to try to fix the CPU_overloaded problem. Its fault probability has risen 20-fold from 0.027 to 0.56 as indicated in Figure 3.6.3-6.

In Figure 3.6.3-7 we see that as the Cpu was not found overloaded the next recommended action is to observe the node IpInDiscards which gives strong evidence on the failure BufferSize_Too_Small. In Figure 3.6.3-8 and Figure 3.6.3-9 we see that when we observed IpInDiscards to be High the belief in BufferSize_Too_Small increased 70 fold (0.004 -> 0.287). It is also recommended to be fixed as the next action (see Figure 3.6.3-10). Now this node is found to have a fault and the troubleshooting session terminates.

From this example session one can see that the provided recommendations seem to be reasonable and focus the troubleshooting session at each step to the most cost efficient direction. The user interface of the used tool is rather basic but provides the necessary information for testing.

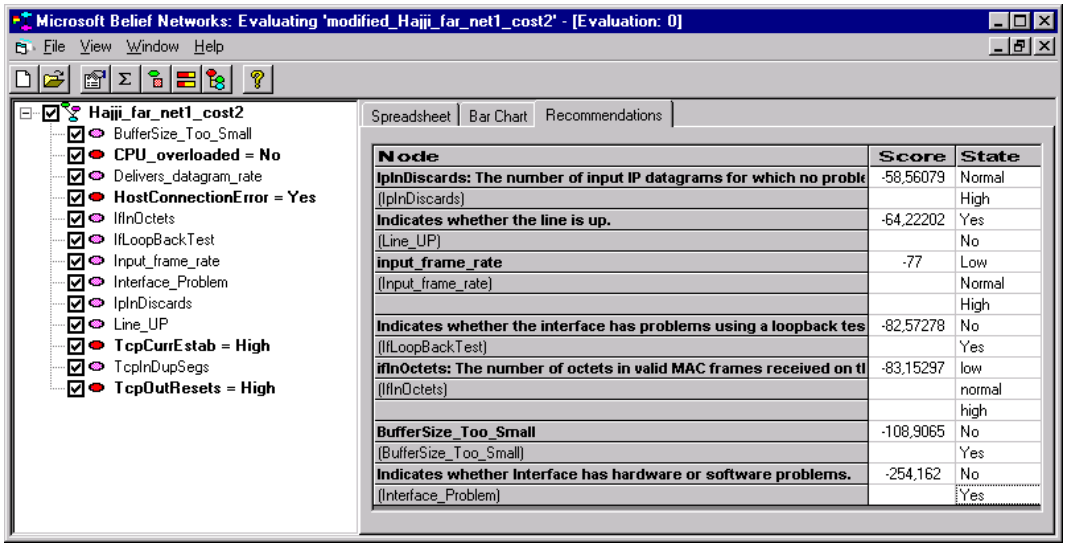
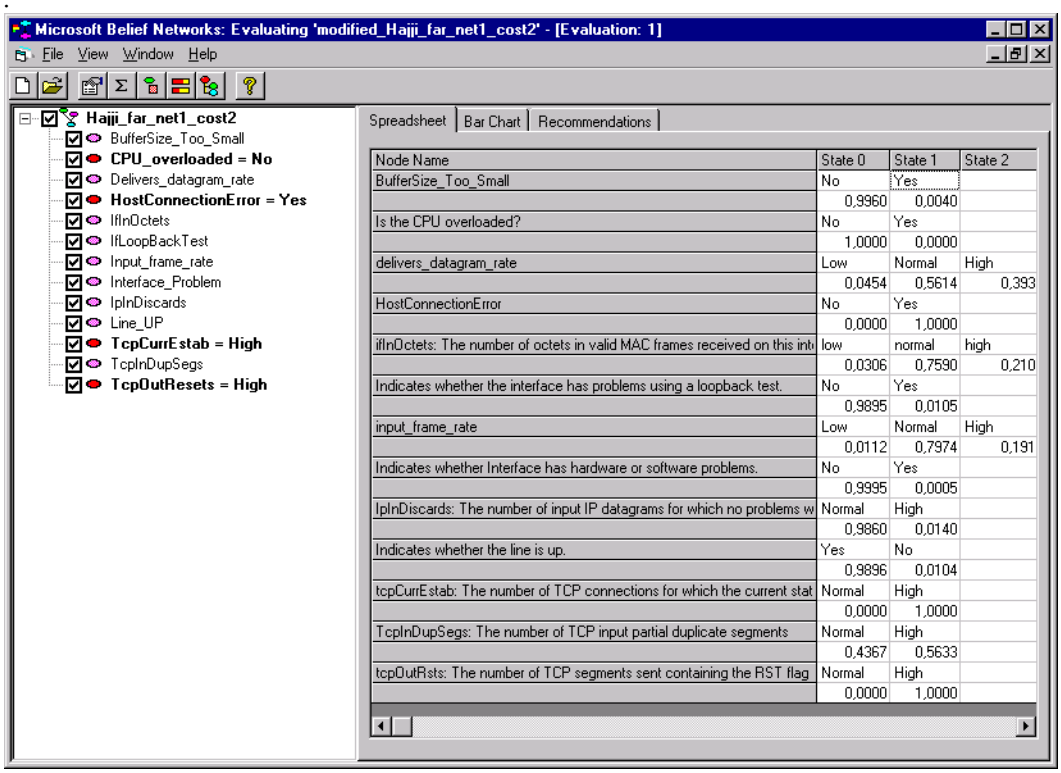


Figure 3.6.3-7. A recommendation list of actions to perform next given the new evidence (CPU_overloaded = No).

.



Node Name	State 0	State 1	State 2
BufferSize_Too_Small	No	Yes	
	0,9960	0,0040	
Is the CPU overloaded?	No	Yes	
	1,0000	0,0000	
delivers_datagram_rate	Low	Normal	High
	0,0454	0,5614	0,393
HostConnectionError	No	Yes	
	0,0000	1,0000	
ifInOctets: The number of octets in valid MAC frames received on this int	low	normal	high
	0,0306	0,7590	0,210
Indicates whether the interface has problems using a loopback test.	No	Yes	
	0,9895	0,0105	
input_frame_rate	Low	Normal	High
	0,0112	0,7974	0,191
Indicates whether Interface has hardware or software problems.	No	Yes	
	0,9995	0,0005	
IpInDiscards: The number of input IP datagrams for which no problems w	Normal	High	
	0,9860	0,0140	
Indicates whether the line is up.	Yes	No	
	0,9896	0,0104	
tcpCurrEstab: The number of TCP connections for which the current stat	Normal	High	
	0,0000	1,0000	
TcpInDupSegs: The number of TCP input partial duplicate segments	Normal	High	
	0,4367	0,5633	
tcpOutRsts: The number of TCP segments sent containing the RST flag	Normal	High	
	0,0000	1,0000	

Figure 3.6.3-8. . Marginal probabilities for the nodes given the new evidence (CPU_overloaded = No).

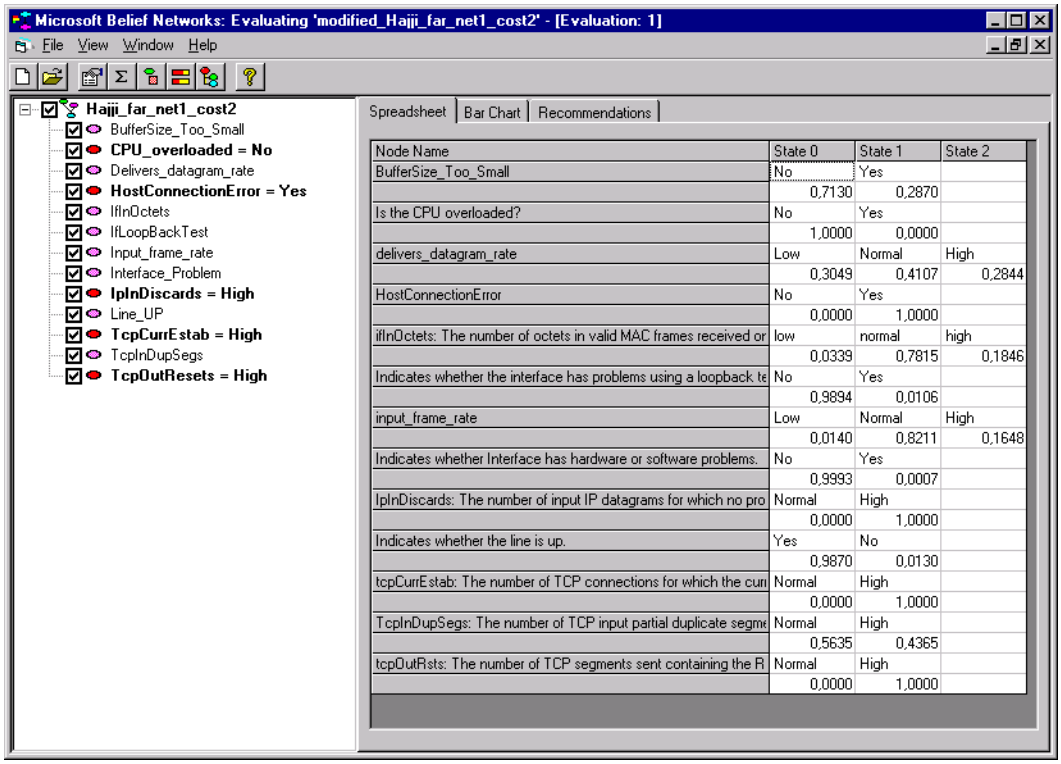


Figure 3.6.3-9. Marginal probabilities for the nodes given the new evidence (IpInDiscards = High).

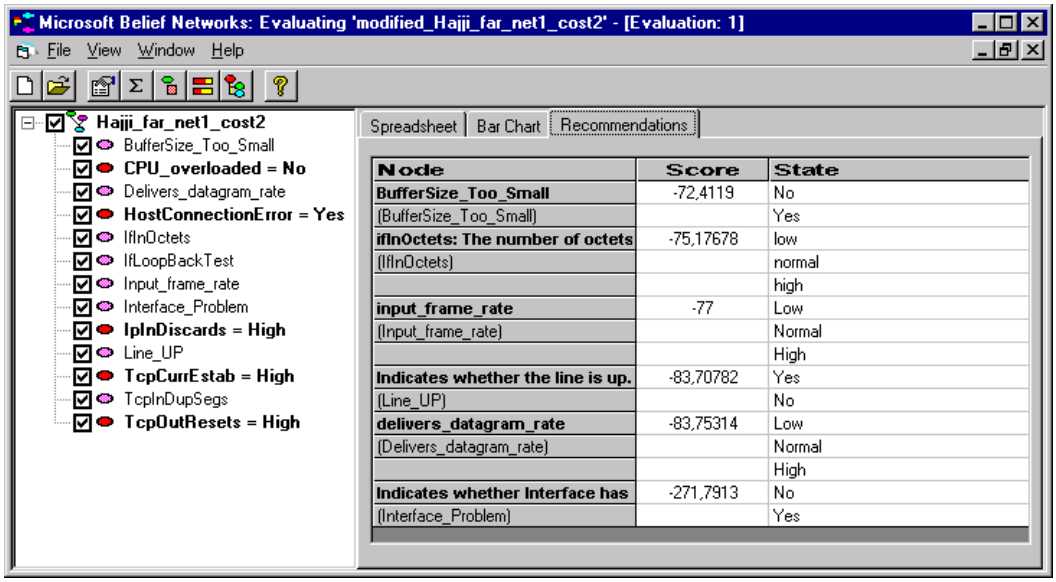


Figure 3.6.3-10. A recommendation list of actions to perform next given the new evidence (IpInDiscards = High).

3.6.4 A Value-of-Information (VOI) guided diagnostic session dealing with TCP problems

This example is similar to the previous one (see Figure 3.6.4-1). The probability matrices are the same. The only modifications deal with the MSBNx tool requirements: the problem defining node has been removed, the primary fault nodes have been labeled hypothesis nodes and all other nodes informational. The aim of this session is to show how Value-of-Information (VOI) computations can be used to make recommendations for the next evidence to gather. The textual description of the used Bayesian network has been included as Appendix D.

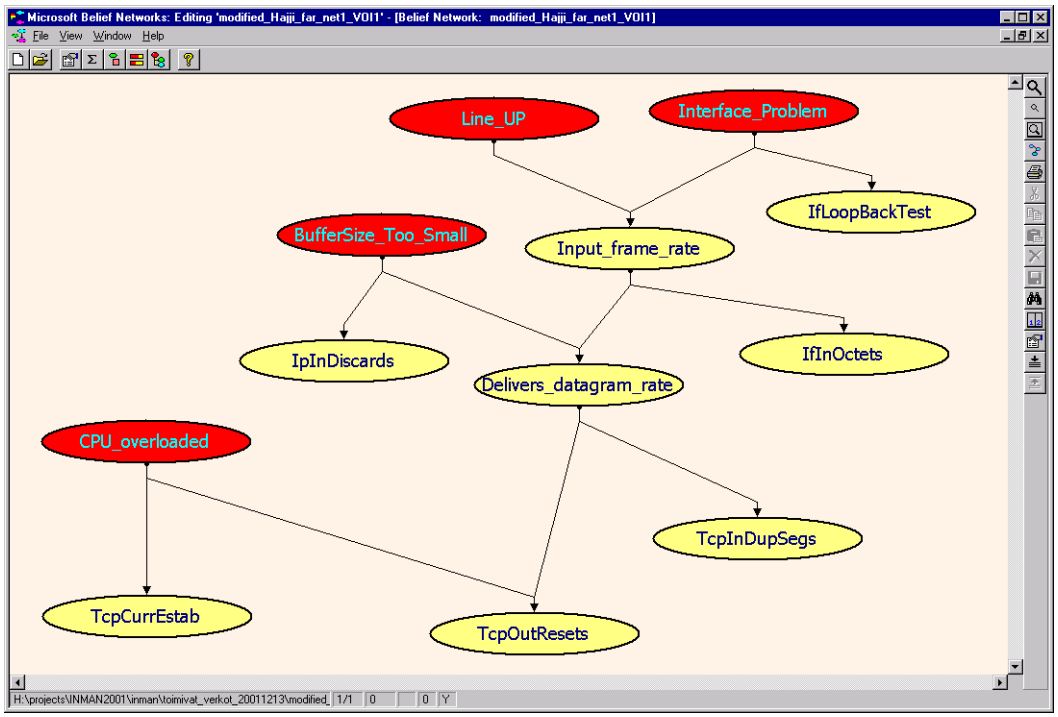


Figure 3.6.4-1. A Bayesian network model for diagnosing TCP problems of a host interface. (In this figure, informational nodes are marked with the yellow/white color, hypothesis nodes have the red/dark grey color).

In Figure 3.6.4-2 we have plotted the initial probabilities for the nodes before any evidence has been made available. In Figure 3.6.4-3 we notice that the input_frame_rate is considered the most valuable next observation as it can separate Line_UP and Interface_Problem faults. In Figure 3.6.4-4 the new node probabilities indicate that the fault hypothesis, Line_UP and Interface_Problem have been eliminated because high input_frame_rate is very rare with these faults. In Figure 3.6.4-5 the new best recommendation is to test node IpInDiscards which gives evidence on the likelihood of BufferSize_Too_Small fault. In Figure 3.6.4-6 we notice that the new evidence with high IP packet discarding level supports the fault hypothesis BufferSize_Too_Small whose probability has risen 50-fold (0.01 -> 0.50). According to Figure 3.6.4-7 the next observable node would be node Delivers_datagram_rate as it gives further knowledge on the previous fault hypothesis. This node is found to be in the Low state which makes the

fault hypothesis BufferSize_Too_Small almost certain as indicated in Figure 3.6.4-8. This was accomplished with only three observations from eight possible ones.

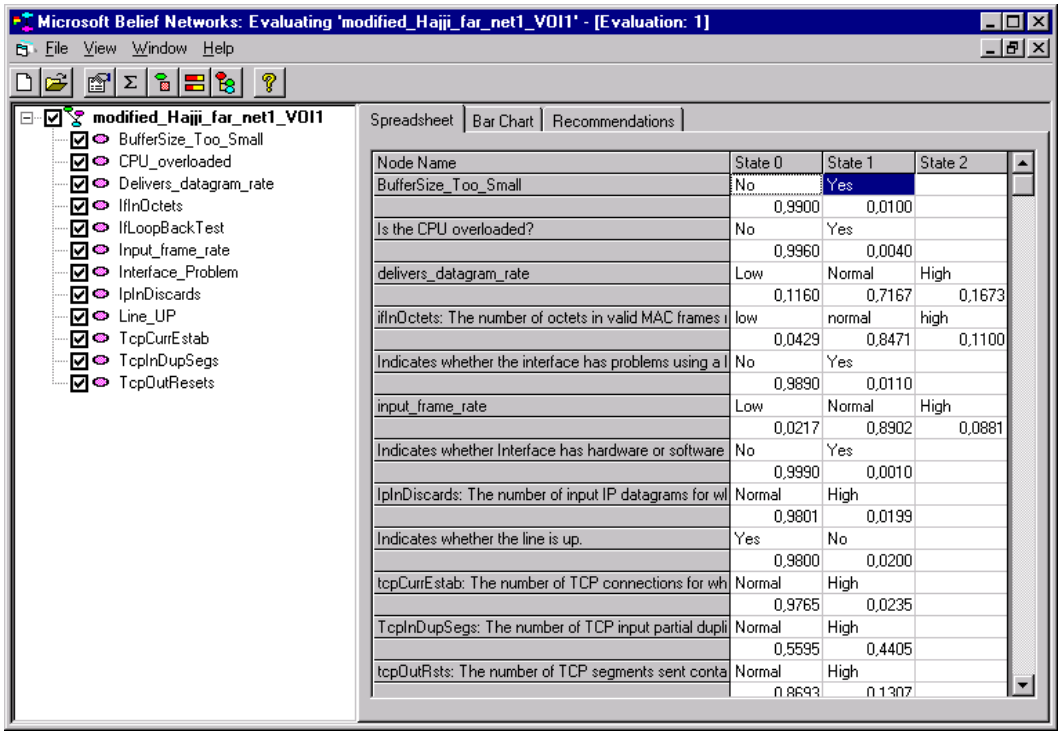


Figure 3.6.4-2. Marginal probabilities before any evidence has been inserted.

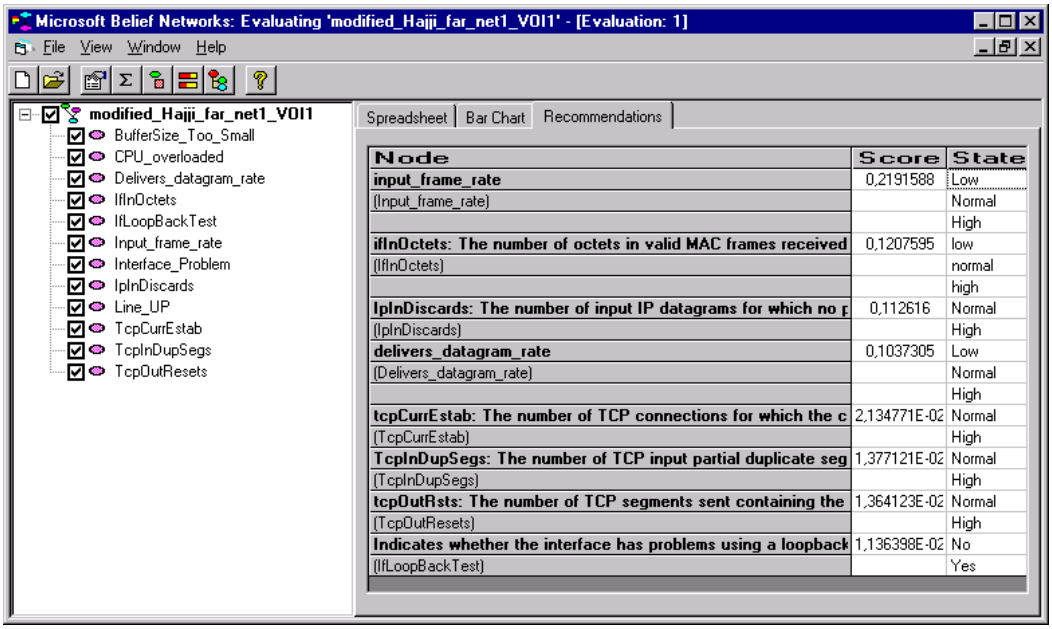


Figure 3.6.4-3. Recommendations for the best observations to make in order to make diagnosis more accurate.

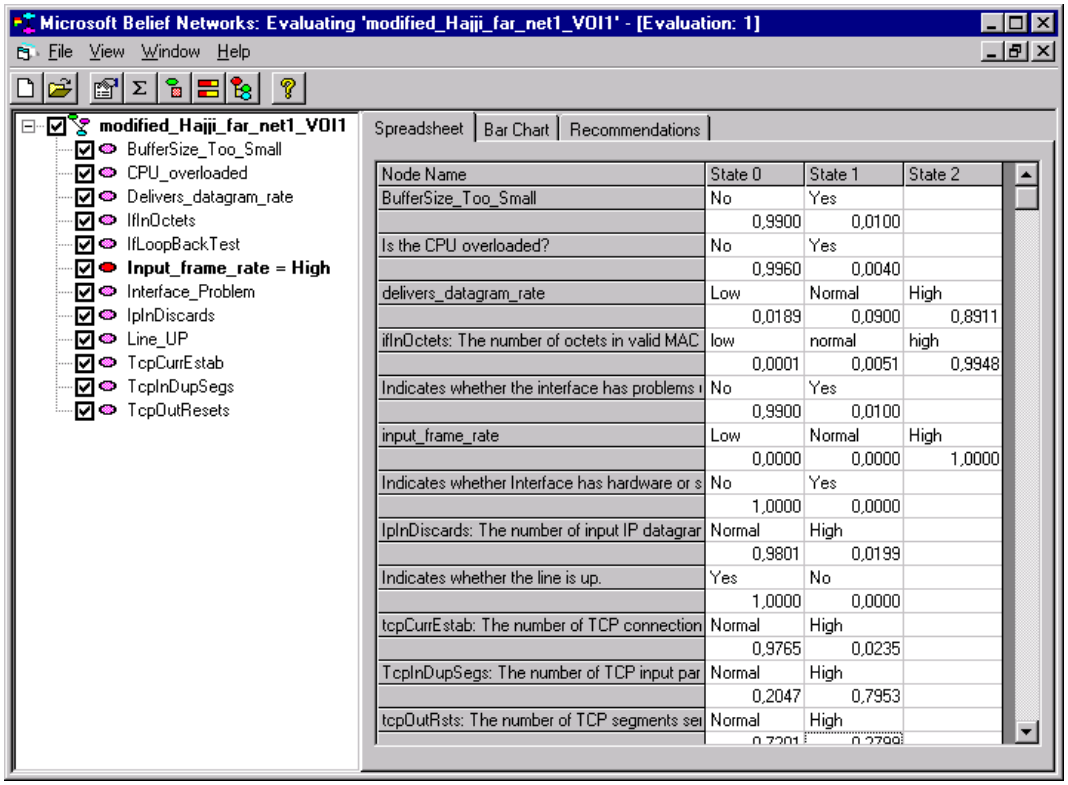


Figure 3.6.4-4. Marginal probabilities given the new evidence (Input_frame_rate = High).

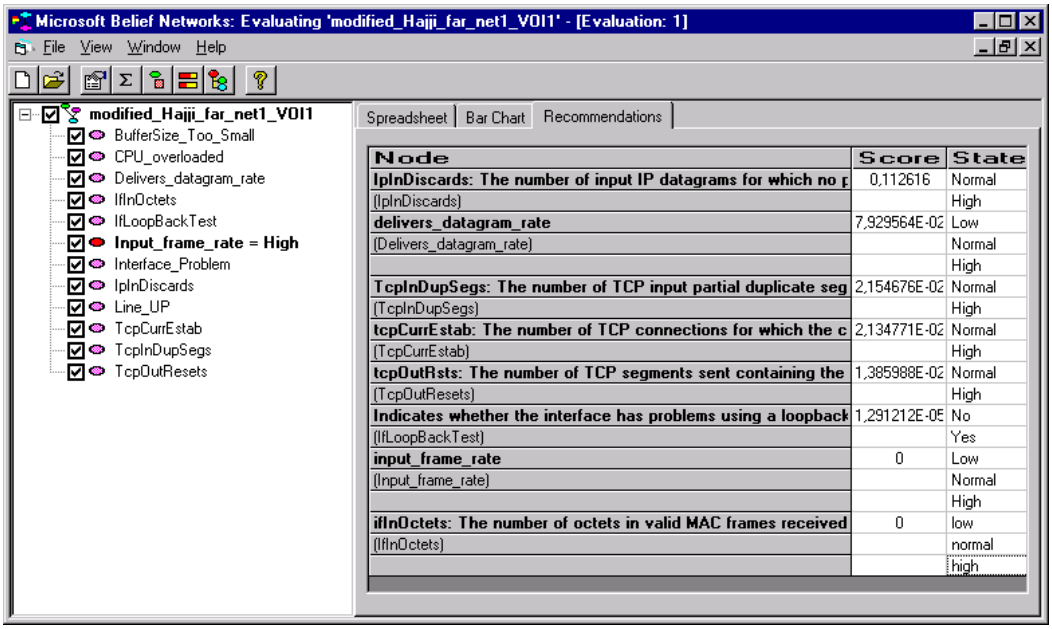


Figure 3.6.4-5. Recommendations for the best observations to make in order to make diagnosis more accurate.

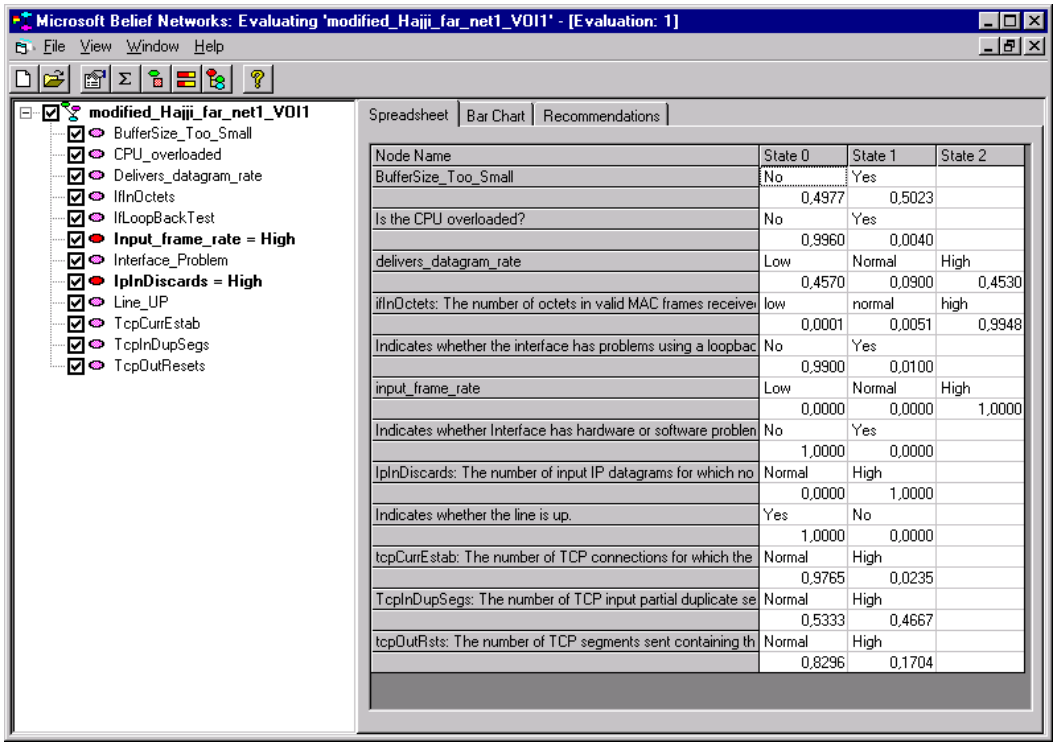


Figure 3.6.4-6. Marginal probabilities given the new evidence (IpInDiscards = High).

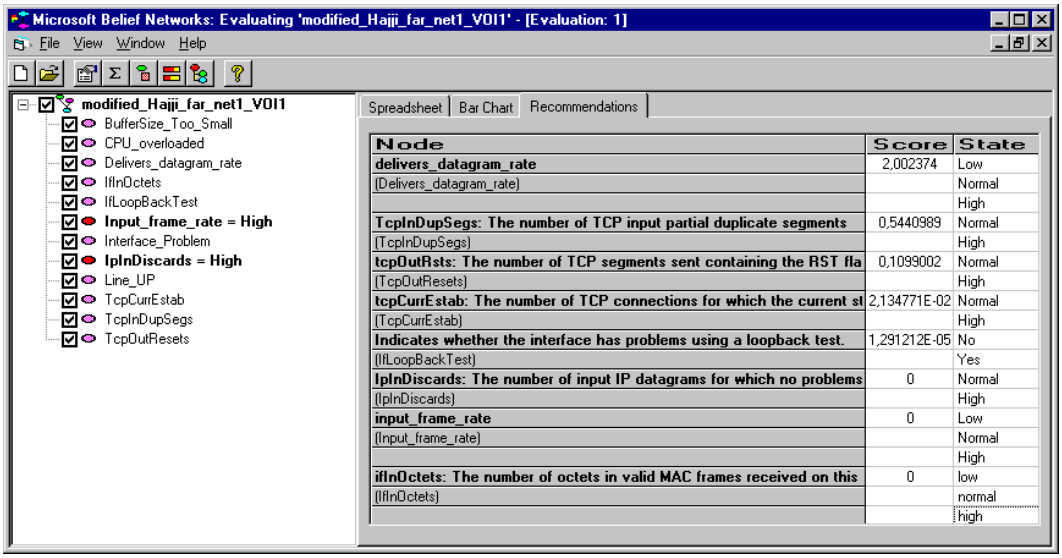
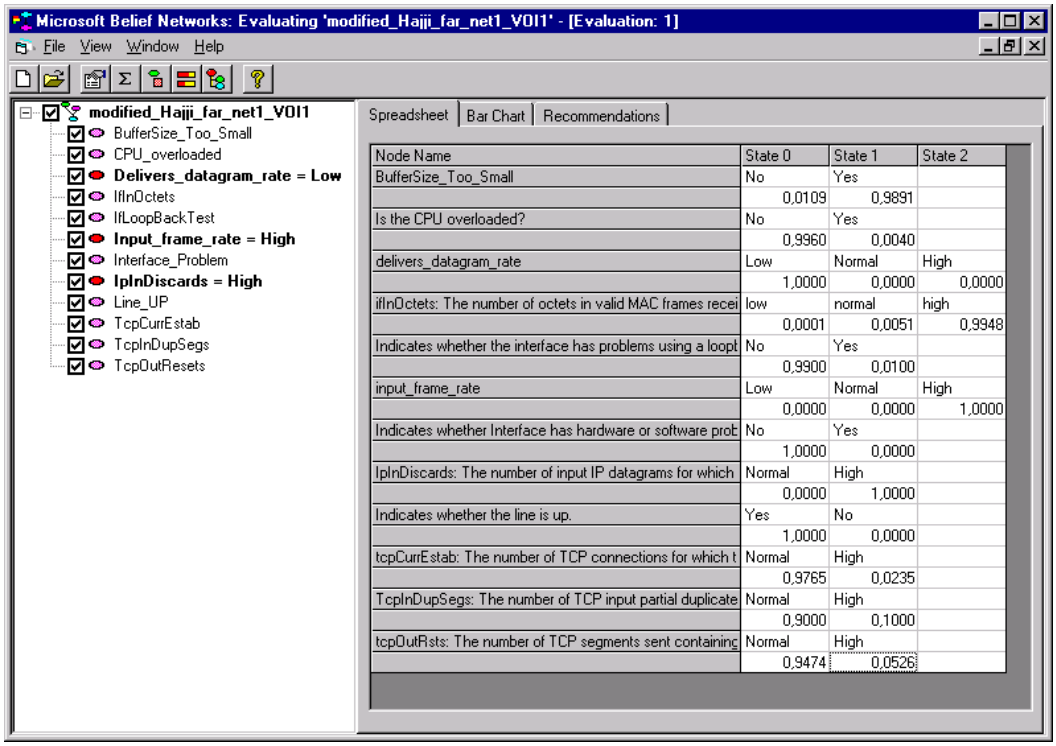


Figure 3.6.4-7. Recommendations for the next observation given the new evidence (IpInDiscards = High).



Node Name	State 0	State 1	State 2
BufferSize_Too_Small	No	Yes	
	0,0109	0,9891	
Is the CPU overloaded?	No	Yes	
	0,9960	0,0040	
delivers_datagram_rate	Low	Normal	High
	1,0000	0,0000	0,0000
ifInOctets: The number of octets in valid MAC frames recei	low	normal	high
	0,0001	0,0051	0,9948
Indicates whether the interface has problems using a loopt	No	Yes	
	0,9900	0,0100	
input_frame_rate	Low	Normal	High
	0,0000	0,0000	1,0000
Indicates whether Interface has hardware or software prot	No	Yes	
	1,0000	0,0000	
IpInDiscards: The number of input IP datagrams for which	Normal	High	
	0,0000	1,0000	
Indicates whether the line is up.	Yes	No	
	1,0000	0,0000	
tcpCurrEstab: The number of TCP connections for which t	Normal	High	
	0,9765	0,0235	
TcpInDupSegs: The number of TCP input partial duplicate	Normal	High	
	0,9000	0,1000	
tcpOutRsts: The number of TCP segments sent containing	Normal	High	
	0,9474	0,0526	

Figure 3.6.4-8. Marginal probabilities given the new evidence (Delivers_datagram_rate = Low).

3.6.5 Diagnosing microwave link problems in a GSM access network

In this example we briefly study an alarm correlation example concerning a GSM access network which has been described in [Wiet97]. Here the Base Station Controller (BSC) takes care of interfacing a set of Base Stations (BS) to the traditional switched network. The BSs are responsible for the mobile phone traffic at their cell area. These stations are typically connected to the BSC via cable links or microwave links (MWL). The network topology is logically a star and physically a tree where the traffic to several base stations is distributed over a chain of microwave links and leased lines (see Figure 3.6.5-1).

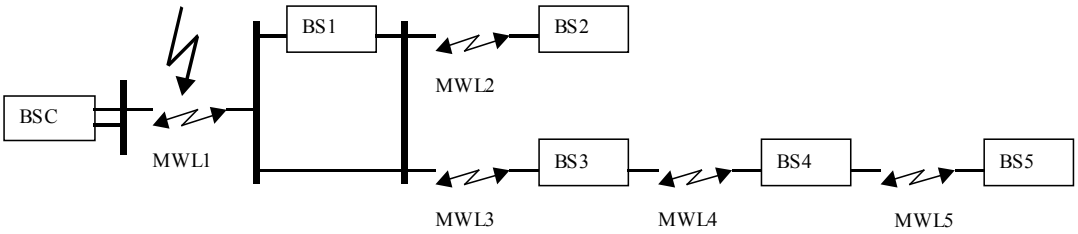


Figure 3.6.5-1 Topology of the studied GSM access network

When a link fails a number of alarms is generated and passed to the OMC (Operations and Maintenance Center). One has to reason based on the network topology to correlate the alarms and detect the root cause of the failure. As the topology is subject to frequent changes the models should be easy to adapt to the changed configuration.

In the given example the network elements send various types of alarms upon detecting abnormal behaviour. When a microwave link fails the BSC can send alarms like incoming_signal_missing, D_channel_failure, BER_over_limit etc and the BS may send PCM_failure, LAPD_failure, BCCH_missing when the connection to the BSC is broken. Some of the alarms may be missing or additional alarms may be included constituting noise.

Next we shall show how to model the situation with a Bayesian network. In this case we only model the effect of the microwave links on the BS and BSC alarms (only one type of alarm is modeled for both network elements: whether the connection to the nearest BS is broken for BSC and whether the upstream BS or BSC connection is broken for the BS) (see Figure 3.6.5-2). The textual description of the used Bayesian network has been included as Appendix E. According to the model the failure of an upstream microwave link causes all downstream base stations to alarm. The BSC alarms are explained by the connection to all the base stations getting disconnected – only microwave link 1 can explain that.

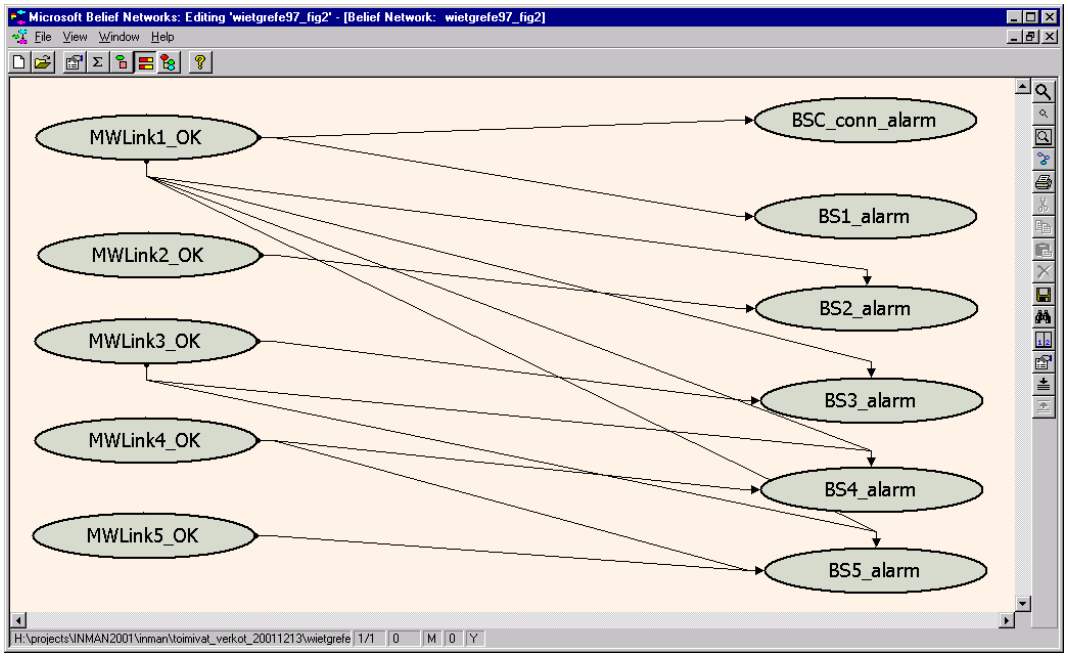
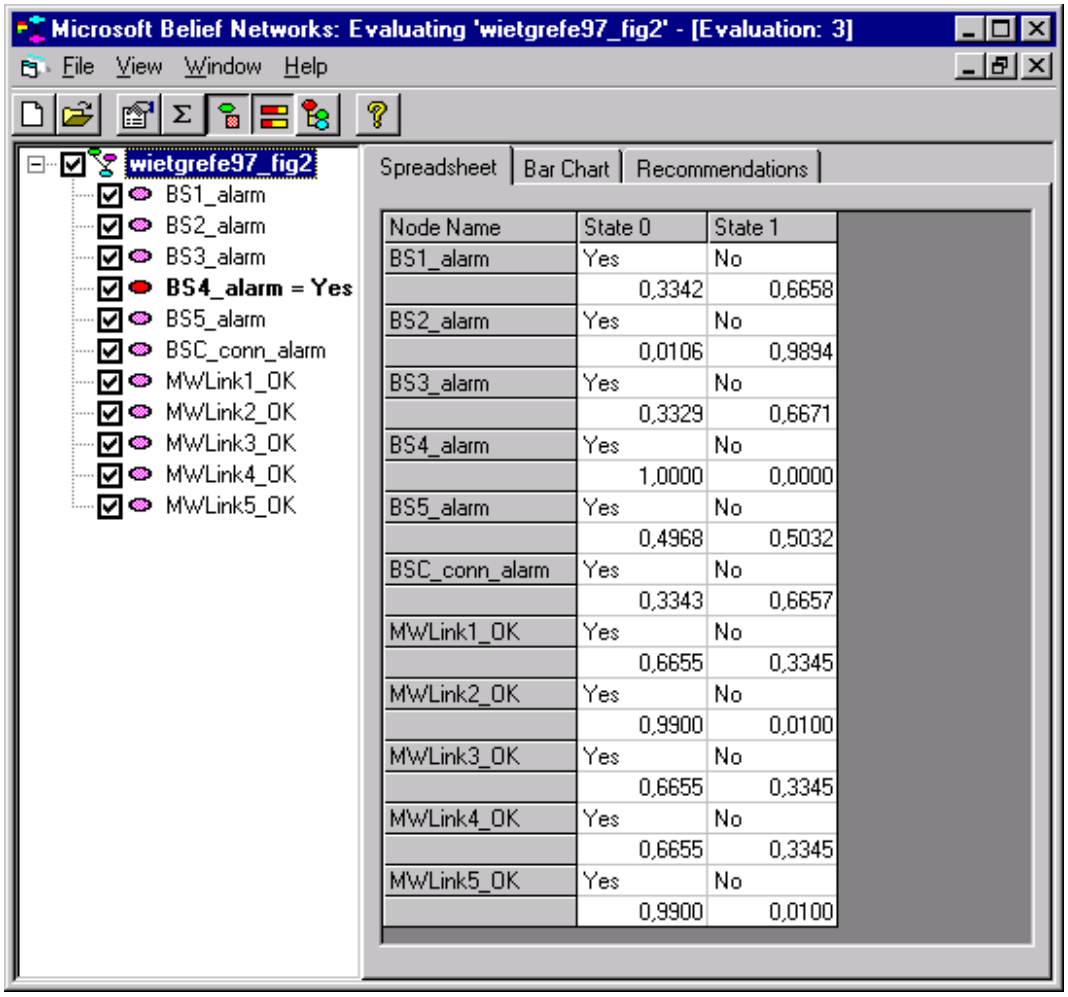


Figure 3.6.5-2 A Bayesian network for detecting the failed microwave link based on the BS and BSC alarms received

We test the model by inserting some alarms into the system. At first (see Figure 3.6.5-3) we observe that BS4 is alarming which raises suspicion of a fault in microwave links 4, 3 and 1 as each of them may explain this alarm. Also the expectation of an alarm in BS 1, 3 and 5 has risen because of the suspected faults.



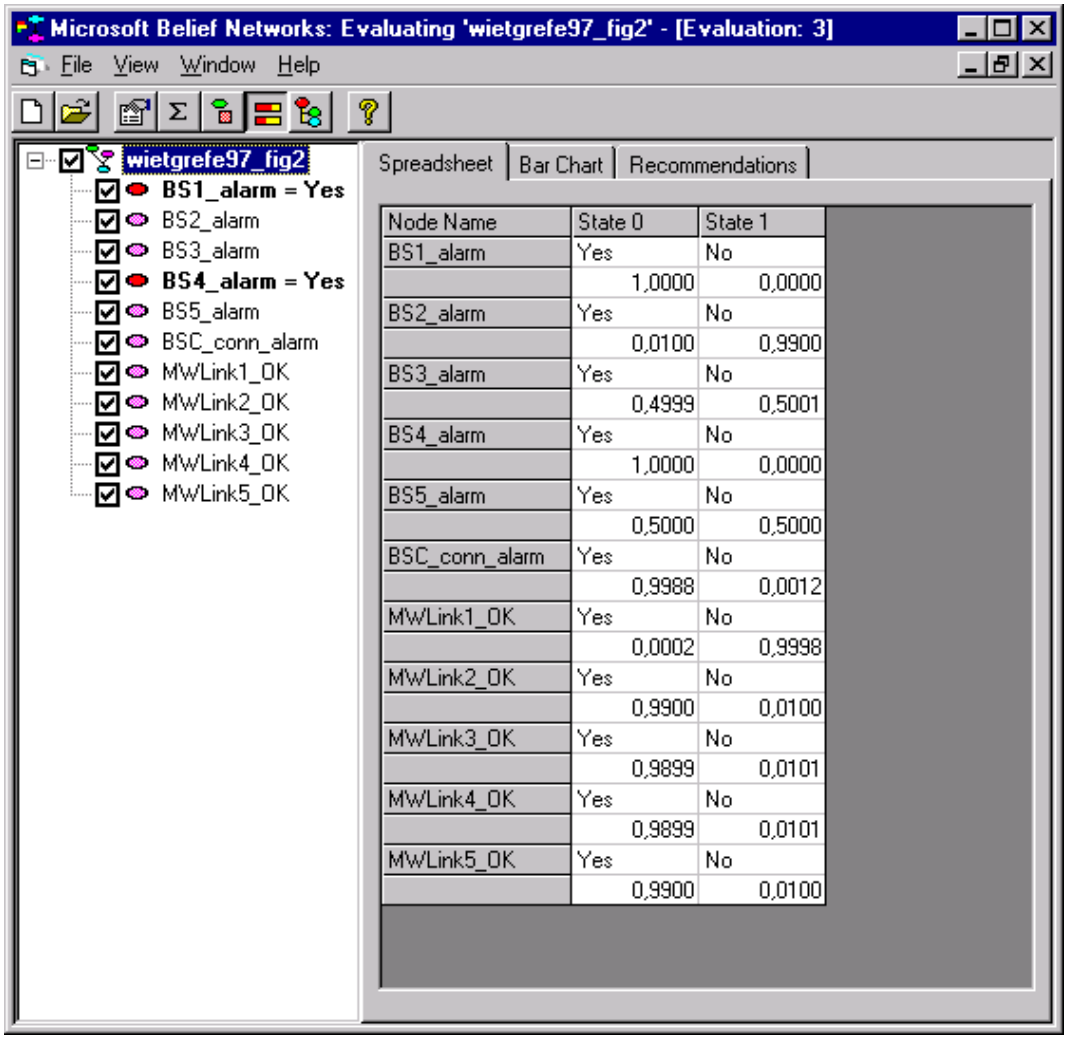
Node Name	State 0	State 1
BS1_alarm	Yes	No
	0,3342	0,6658
BS2_alarm	Yes	No
	0,0106	0,9894
BS3_alarm	Yes	No
	0,3329	0,6671
BS4_alarm	Yes	No
	1,0000	0,0000
BS5_alarm	Yes	No
	0,4968	0,5032
BSC_conn_alarm	Yes	No
	0,3343	0,6657
MWLink1_OK	Yes	No
	0,6655	0,3345
MWLink2_OK	Yes	No
	0,9900	0,0100
MWLink3_OK	Yes	No
	0,6655	0,3345
MWLink4_OK	Yes	No
	0,6655	0,3345
MWLink5_OK	Yes	No
	0,9900	0,0100

Figure 3.6.5-3 Marginal probabilities of the nodes given that BS4 is alarming.

Next we insert new evidence (BS1 is alarming) which makes the failure of microwave link 1 almost certain given the evidence as it is the only fault hypothesis that can explain both of the alarms. See Figure 3.6.5-4.

Adding more alarm types to the base stations could be easily modeled by making the BSn_alarm node a parent node with its children being the associated alarm types. The conditional probabilities would be defined so that observing more alarm types makes one more certain on the alarming status of the base station. This would also allow some noise in the observations.

All the base stations may be reasonably modeled with similar conditional probability tables using the conditional independence assumption which makes it possible to dynamically adjust the diagnostic model based on the changed topology.



Node Name	State 0	State 1
BS1_alarm	Yes	No
	1,0000	0,0000
BS2_alarm	Yes	No
	0,0100	0,9900
BS3_alarm	Yes	No
	0,4999	0,5001
BS4_alarm	Yes	No
	1,0000	0,0000
BS5_alarm	Yes	No
	0,5000	0,5000
BSC_conn_alarm	Yes	No
	0,9988	0,0012
MWLink1_OK	Yes	No
	0,0002	0,9998
MWLink2_OK	Yes	No
	0,9900	0,0100
MWLink3_OK	Yes	No
	0,9899	0,0101
MWLink4_OK	Yes	No
	0,9899	0,0101
MWLink5_OK	Yes	No
	0,9900	0,0100

Figure 3.6.5-4 Marginal probabilities of the nodes given that both BS1 and BS4 are alarming

3.6.6 A diagnostic model for a Linear Lightwave Network (LLN) case

The following example exemplifies how network topology information (routing information in this case) may be used to dynamically update a diagnostic Bayesian network when changes are made. This is possible as the model is composed of many similar submodels that have similar probability tables which allows automated network construction.

Our example is based on the LLN (Linear Lightwave Network) problem described in [Deng93]. In this case controllable transparent optical paths are established among network users by routing them through linear divider/combiner units (LDC). Many such key elements are combined together to implement the desired routing function. The diagnostic task is to detect the most likely failed LDCs when abnormality is detected in the output signal levels of the network. While monitoring the network only incoming and outgoing signals are measured. If failures are detected the inputs and outputs of each LDC

can also be measured but this is a costly operation. The aim is to detect the failures while minimizing the number of required additional tests.

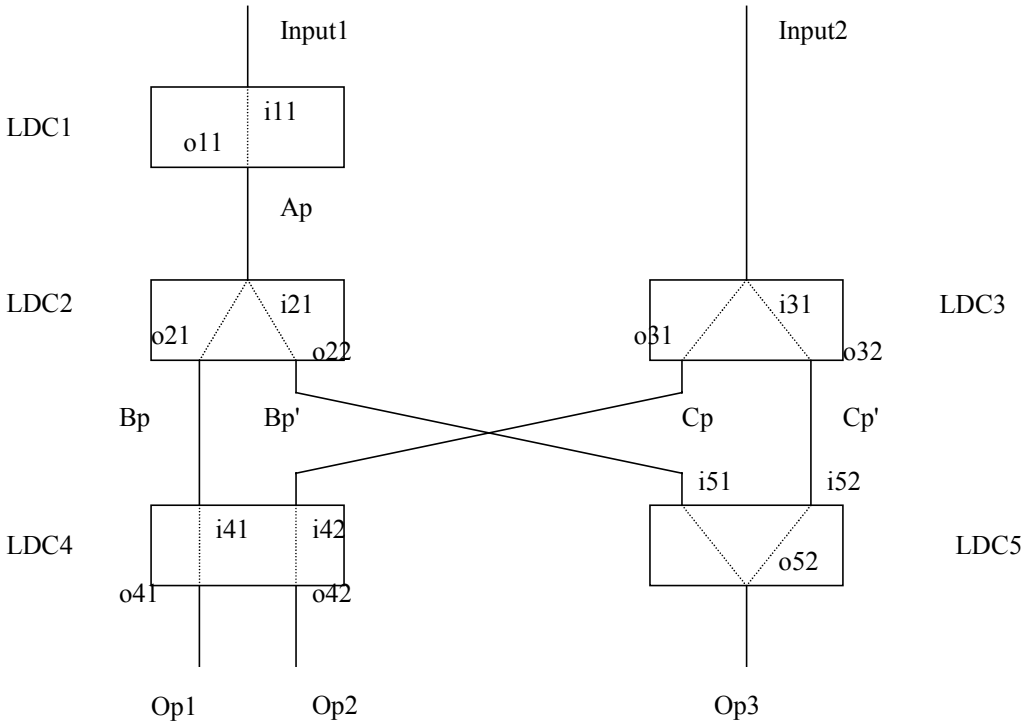


Figure 3.6.6-1 The example LLN network

Our example network is simple (see Figure 3.6.6-1). The LDC units are described with the rectangles. Different routes through them are marked with dashed lines. The route information has been stored in the LLN routing tables. The output of the LDC is considered erroneous when either the input is abnormal or the LDC unit has failed. The inputs and the outputs are measured during the normal operation of the network. There states are categorized to normal or abnormal. If problems are detected new measurements may be performed by exploring the intermediary power levels (Ap, Bp, Cp). The task is to detect the failing LDC units with as few additional tests as possible.

A Bayesian network model has been created for the problem (see Figure 3.6.6-2). The textual description of the used Bayesian network has been included as Appendix F. The prior probabilities of failure for all the inputs have been estimated to be 0.0001 and for the LDC failures 0.001. In [Deng93] deterministic conditional probabilities were used but we have added some uncertainty to them in order to allow unmodeled failure sources. The conditional probabilities for the outputs of the LDCs have been defined as conditionally independent from the parents with a chance of failure 0.999 in case the input or the LDC has failed. The leak probability for failure is 0.001 (having an abnormal output although all the inputs and the LDC are normal). This allows one to dynamically build the LLN diagnostic models based on the routing table information.

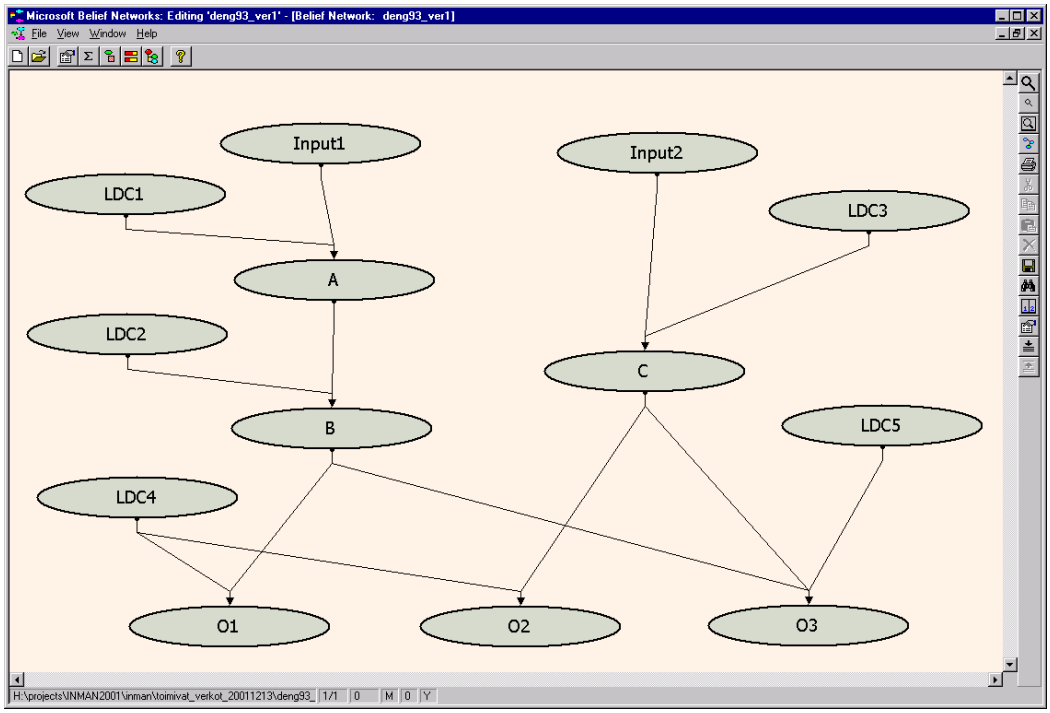
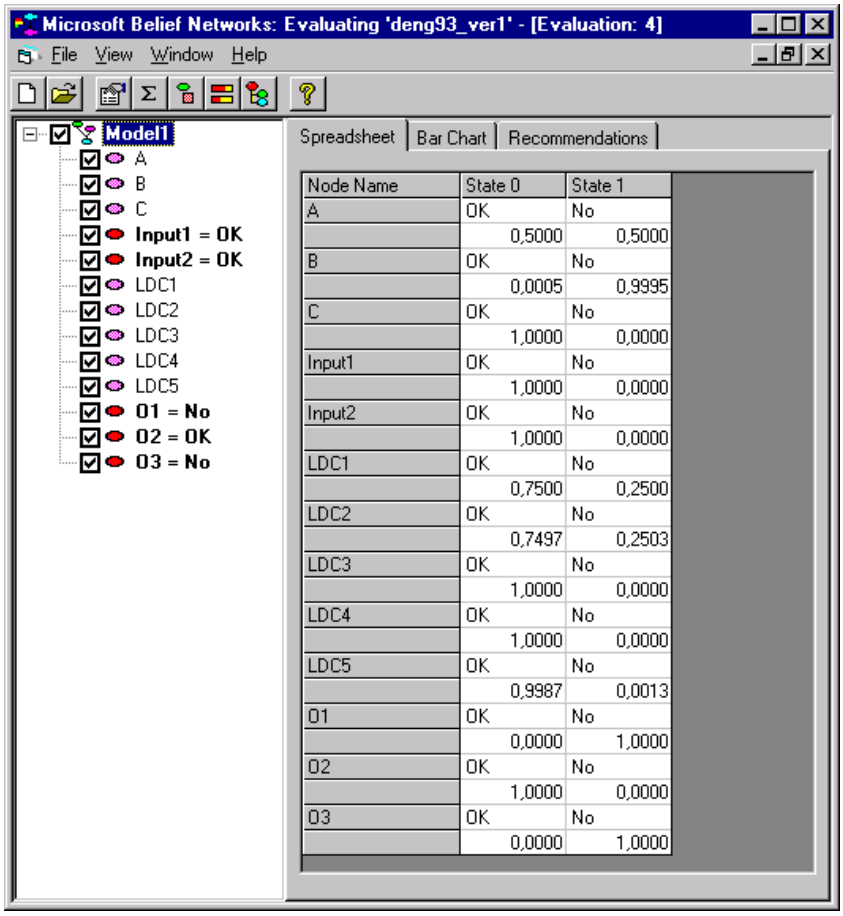


Figure 3.6.6-2 A Bayesian network model for the LLN example.

As an example we consider the case where all the inputs have been found to be normal and the outputs O1 and O3 have been measured to be faulty and the output O2 normal. In Figure 3.6.6-3 we have plotted the marginal probabilities given this initial evidence. We can see that LDC4 and signal level C are believed to be normal as they are direct parents of the normal output O2 which almost certainly requires them to be normal. On the other hand signal level B is almost certainly believed to be abnormal as it would explain both abnormal outputs. Both LDC1 and LDC2 are good candidates for the failures. Knowing the signal level A (which is now completely undecided) would tell us whether LDC1 is faulty. In Figure 3.6.6-4 we see that the Value-of-Information based recommendation is to observe node A.

In Figure 3.6.6-5 we see that the new observation (A=OK) makes us certain on LDC1's normal operation and doubles our concern on LDC2 being faulty. (The relatively large remaining uncertainty on LDC2 is due to the small prior probability of LDC2 being failed (0.001) and on the possibility of an output power level being abnormal even when the LDC and the inputs are normal (leak probability = 0.001) which cancel each others effect in this case.)



Node Name	State 0	State 1
A	OK	No
	0,5000	0,5000
B	OK	No
	0,0005	0,9995
C	OK	No
	1,0000	0,0000
Input1	OK	No
	1,0000	0,0000
Input2	OK	No
	1,0000	0,0000
LDC1	OK	No
	0,7500	0,2500
LDC2	OK	No
	0,7497	0,2503
LDC3	OK	No
	1,0000	0,0000
LDC4	OK	No
	1,0000	0,0000
LDC5	OK	No
	0,9987	0,0013
O1	OK	No
	0,0000	1,0000
O2	OK	No
	1,0000	0,0000
O3	OK	No
	0,0000	1,0000

Figure 3.6.6-3 Marginal probabilities after entering the input and output node states.

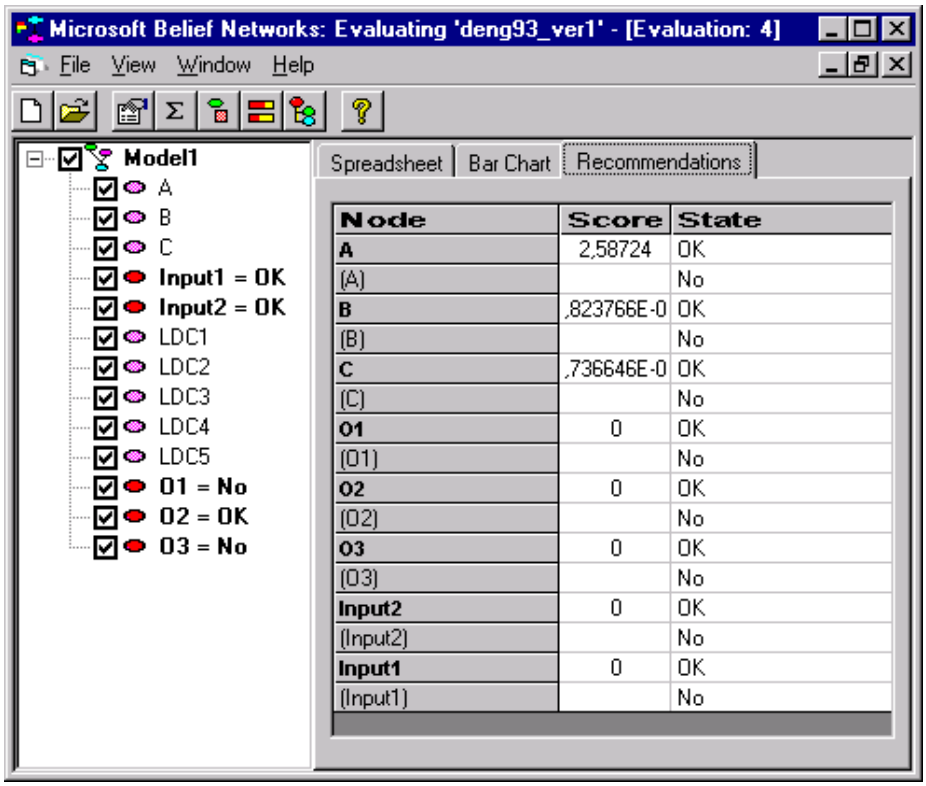
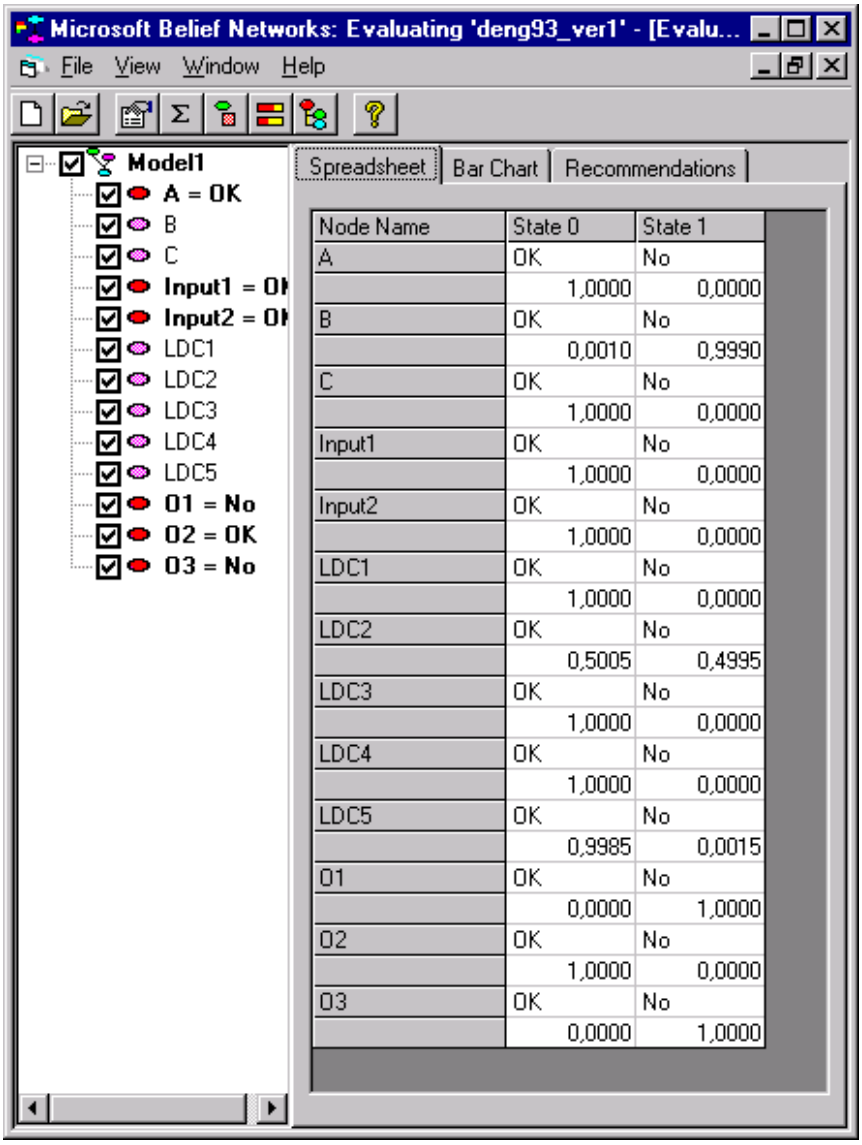


Figure 3.6.6-4 Value-of-Information based recommendations given the initial evidence.



Node Name	State 0	State 1
A	OK	No
	1,0000	0,0000
B	OK	No
	0,0010	0,9990
C	OK	No
	1,0000	0,0000
Input1	OK	No
	1,0000	0,0000
Input2	OK	No
	1,0000	0,0000
LDC1	OK	No
	1,0000	0,0000
LDC2	OK	No
	0,5005	0,4995
LDC3	OK	No
	1,0000	0,0000
LDC4	OK	No
	1,0000	0,0000
LDC5	OK	No
	0,9985	0,0015
O1	OK	No
	0,0000	1,0000
O2	OK	No
	1,0000	0,0000
O3	OK	No
	0,0000	1,0000

Figure 3.6.6-5 Marginal probabilities after entering the observation that A is normal.

3.7 How to use decision-theoretic troubleshooting to guide the repair activities to correct trouble-ticketed problems

In Figure 3.7-1, we have sketched one way of using decision-theoretic troubleshooting in fault management. In this scenario the traditional network monitoring system works as usual interpreting the gathered alarms and ordinary trouble tickets are generated. The maintenance personel is provided a maintenance support terminal (could be a handheld or mobile terminal) which provides customized help for at least some of the hardest problem types. Our scenario is that the maintainer is assigned the next problem from the trouble ticket database and asks for help from the troubleshooter that loads the appropriate decision-theoretic model from a model library and retrieves the relevant alarm evidence from the alarm database and based on this information computes the component fault probabilities and provides a prioritized list of action recommendations (repairs or observations). The user makes the desired observations and may attempt repairs or configuration changes and then enters the new observations to the terminal and gets an updated list of recommendations and component fault probabilities. The user could also have access to a digital service manual describing the recommended repair operations in detail. All the recommendations and selections could be stored into a troubleshooting session log. Also the success of the repair activities should be recorded into a maintenance log. This makes it possible to measure the success of the recommendations and identify problems in the knowledge base.

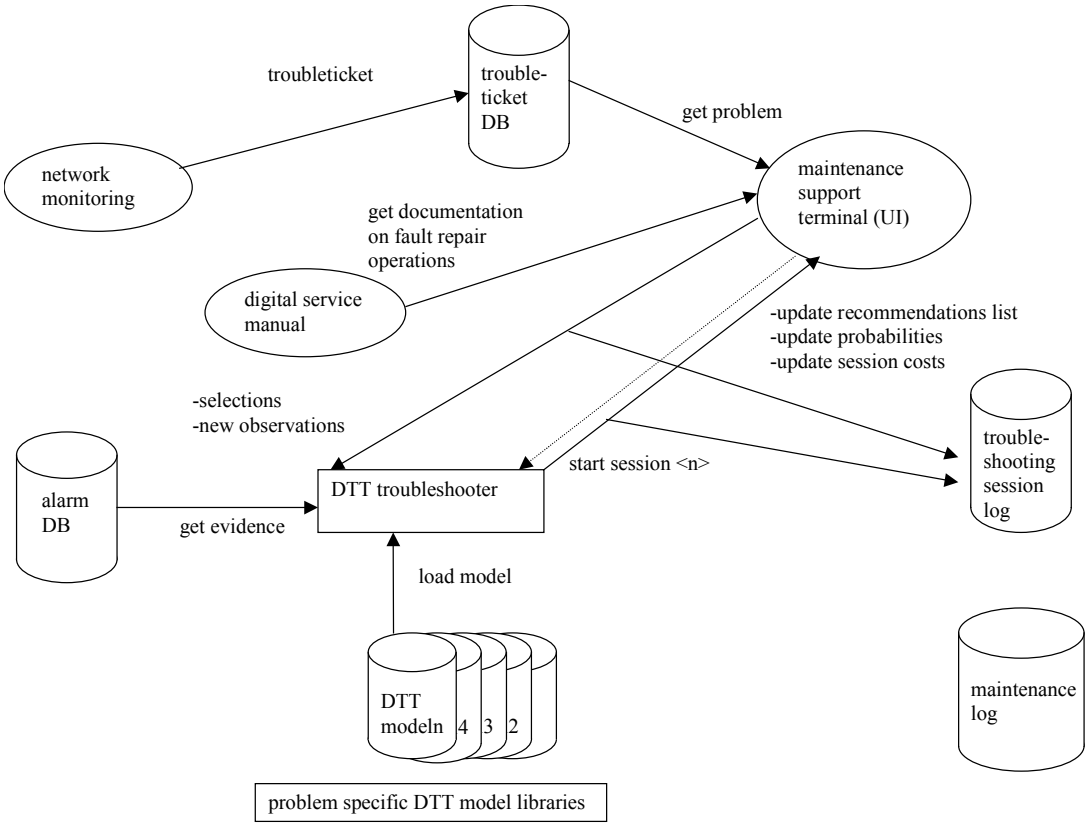


Figure 3.7-1 How to use DTT to guide maintenance persons in repair activity

3.8 Results and discussion

The troubleshooting algorithm only uses the given cost information and the marginal fault probabilities to make recommendations. It is the task of the modeler to identify all the relevant repairable components and repair operations for them. If a digital service manual exists at least the core information should be readily available. Once the repair operations are known one has to estimate a number for repair costs quantifying the different related costs (time, money, delays etc.). Component fault probabilities, repair costs and observation costs need to be known to carry out the troubleshooting activity.

Bayesian networks provide an efficient model for representing probabilistic dependencies between selected random variables. Fast algorithms exist for calculating marginal probabilities of the variables conditioned on the available evidence. This has made decision-theoretic troubleshooting feasible even for large problems.

Although the decision-theoretic troubleshooting engine only deals with the abstracted cost and probability numbers, a troubleshooting guidance application may be built which makes recommendations and has also connections to a digital service manual describing the appropriate recovery operations to perform to fix a suspected fault.

Bayesian network modeling is well suited to model dependencies between related variables when the relations are at least partly probabilistic (have some uncertainty attached). One clear advantage of Bayesian network modeling is that the modeler is able to outline these probabilistic dependencies in a qualitative level using the directed acyclic dependency graph which allows him to identify direct causal independencies which make belief propagation feasible. Uncertainty can then be modeled by specifying the conditional probability matrices.

In well defined application areas the topology of the underlying communications network can be used to automatically adjust the Bayesian network model [Deng93]. In special problems Bayesian network modeling may be used as a local diagnostic model allowing an intelligent agent to perform selected recovery actions automatically without human guidance [Haj00b]. As indicated in our first example one possible application is to model the local dependences of various, partly redundant MIB measurements.

With the previous examples we have tried to show how decision-theoretic troubleshooting can be used to guide repair operations also in the fault management area. However, it is open how well the Bayesian network modeling could be applied to model huge communication networks. Real world modeling in a promising subproblem would be needed to determine this. In a metropolitan network there can be at least 100000 alarms per day and many network elements are involved. Some of the elements contain hundreds or even thousands of types of alarm messages. Also the provided information tends to be unstructured and heterogenous between systems. Clearly building any single model to diagnose such a situation would be a large effort. The best approach is to identify the most useful subproblems and to build real-world troubleshooting methods for them. A monitoring system could then take care of activating the required diagnostic models for the different cases. The previous chapter identified one scenario.

4 Summary

In this report we have reviewed the main published application areas of intelligent methods in the domain of communications network management. We have found that various intelligent methods have been applied to solve the alarm correlation task where the main target is to filter redundant data from the overwhelming alarm data stream in order to make the information understandable. Some work has also addressed repair management of faults combining fault diagnosis and repair scheduling. Other application areas cover proactive fault detection and the use of intelligent network management agents.

As better integration of alarm correlation, fault diagnosis and repair activity management still seems to be a problem, the applicability of decision-theoretic troubleshooting and Bayesian probabilistic reasoning has been studied more closely in this report. At first we have described the repair management needs. Then decision-theoretic troubleshooting theory has been reviewed. Previous work using these methods in alarm management has also been reviewed. We have modeled three small problems using Bayesian networks and described example troubleshooting sessions with these models to demonstrate the applicability of the models. A maintenance support terminal was sketched to help the maintenance personel with decision-theoretic recommendations.

When applying decision-theoretic troubleshooting we need both failure probabilities of the repairable components conditioned on the gathered evidence and numeric cost information for making observations and for carrying out repair operations. Bayesian networks provide an efficient formalism to represent probabilistic dependencies between fault hypothesis nodes and informational evidence nodes. They also provide fast algorithms to compute the marginal probabilities of nodes conditioned on instanced evidence nodes. Therefore Bayesian networks are often used for knowledge representation in decision-theoretic troubleshooting models and typically much of the effort needs to be spent on modeling the domain to create the Bayesian network model.

When using decision-theoretic troubleshooting one needs to estimate observation and repair costs for the repairable components and observation costs for informational nodes. Also a repair cost needs to be estimated for replacing the whole system (a service call cost). The troubleshooting engine does not understand the actual repair procedures – it makes recommendations based only on the given cost estimates. All costs have to be summarized in a single number per component.

While constructing Bayesian network models the terminology needs to be defined carefully as random variables and meaningful discrete states need to be defined for these variables. Dependencies are first defined qualitatively by drawing a directed acyclic graph with the random variables as nodes. The dependencies are then quantified by editing the prior and conditional probability matrices of the nodes. The models may be manually defined or they may be learned at least partially from available measurement data.

In monitoring Bayesian networks are easiest to use for local data quantifying the uncertain dependencies between the fault hypothesis and the evidence. In some special tasks the network topology may be utilised to dynamically adjust the Bayesian network model. Bayesian networks have also been used as the modelling language of the

knowledge base of intelligent network management agents. This allows the possibility to distribute reasoning among multiple co-operating intelligent agents.

We did not have the chance to test the suitability of decision-theoretic troubleshooting and Bayesian network modeling to large network management problems as it would have required a real industrial test case with domain expertise. In medical domains Bayesian network models have been built with large diagnostic networks containing hundreds of nodes. Also considering the extension of our first example network of chapter 3.6.3 to cover the whole LAN troubleshooting would create a fairly challenging problem [Haj00a]. In order to determine the suitability of these methods in large network management problems the methods should be evaluated with a test case having good payback for better solutions and which is understood well enough.

References

[Bree96]
Breese, J.S., Heckerman, D., "Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment", Microsoft Research Techical Report MSR-TR-96-06, 1996.

[Deng93]
Deng, R.H., Lazar, A.A., Wang, W., A probabilistic approach to fault diagnosis in linear lightwave networks, IEEE Journal on Selected Areas in Communications, vol. 11, pp. 1438-1448, Dec. 1993.

[Fras92]
Fraser, A.G., Kalmanek, C.R., Kaplan, A.E., Restrick, R.C., Xunet2: a nationwide testbed in high-speed networking, Proc. of IEEE Infocom., 1992.

[Frie92]
Friedrich, G., Gottlob, G., Nejdl, W., "Formalizing the Repair Process", Proc. of the European Conference on Artificial Intelligence, Vienna, August 1992.

[Hajj98]
Hajji, H., Saniepour, S., Soueina, S.O., Hashimoto, H., Far, B.H., "Decision-Theoretic Diagnosis-Repair Planning Using Bayesian Networks", Proceedings of the 12th Annual Conference on Japanese Society of Artificial Intelligence. pp. 256-257, Tokyo, Japan, June, 1998.

[Haj00a]
Hajji, H., Far, B.H., "Distributed Software Agents for Network Fault Management", In Special issue on Knowledge-based Software Engineering in Transactions of IEICE, Vol.E83-D No.4, pp.735-746, April, 2000.

[Haj00b]
Hajji, H., Far, B.H., " Intelligent Agents For Detection of Network Performance Problems", Workshop on Software Agent and its Applications (SAA2000), pp. 269-278, Ibusuki, Japan, 9-10 November, 2000.

[Heck95]
Heckerman, D., Breese, J.S., Rommelse, K., "Decision-Theoretic Troubleshooting", Communications of the ACM, Vol.38-23, pp.49-57, 1995.

[Huar96]
Huard, J., Lazar, A., "Fault isolation based on decision-theoretic troubleshooting", Tech. Rep. CU/CTR TR 442--96--08, Center for Telecommunications Research, Columbia University, New York, NY, 1996.

[Jens01]
Jensen, F.V., Skaanning, C., Kjærulff, U., "The SACSO System for Troubleshooting of Printing Systems", Proc. Seventh Scandinavian Conference on Artificial Intelligence (SCAI '01), 2001.

[Ho99]
Ho, Lawrence L., Macey, Cristopher, J., Hiller, Ronald G., "Real-time performance monitoring and anomaly detection in the internet: An adaptive, objective-driven, mix-and-match approach", Bell Labs Technical Journal, October-December, 1999, pp.23-40.

[Hood97]
Hood, C. and Ji, C., "Proactive Network Fault Detection", IEEE Trans. Reliability, vol. 46, No.3, 333-341, Sept. 1997.

[Kala90]
Kalagnanam, J., Henrion, M., "A comparison of decision analysis and expert rules for sequential analysis", in P. Besnard & S. Hanks (eds), Uncertainty in Artificial Intelligence 4, North-Holland, New York, pp. 271-281, 1990.

[Shay00]
Shayman, M.A., Fernandez-Gaucherand, E., Fault Management in Communication Networks: Test Scheduling with a Risk-Sensitive Criterion and Precedence Constraints, in Proceedings of the IEEE Conference on Decision and Control, Sidney, Australia, December 2000.

[Wiet97]
Wietgrefe, H., Tuchs, K., Jobmann, K., Carls, G., Fröhlich, P., Nejdl, W., Steinfeld, S., Using neural networks for alarm correlation in cellular phone networks, International Workshop on Applications of Neural Networks in Telecommunications, 1997.

[Xian96]
Xiang, Y., A probabilistic framework for multi-agent distributed interpretation and optimization of communication, Artificial Intelligence, Col.87, No.1-2, pp. 295-342, 1996.

Appendix A: MSBNx tool features

In our demonstrations we have used MSBNx, a research tool made available by Microsoft Research (see http://research.microsoft.com/adapt/msbnx/). It provides a user interface and a programming library for Bayesian network reasoning with discrete random variables and also supports Value-of-Information computations and cost-based decision-theoretic troubleshooting.

Mutual information based recommendations tell what new evidence would most effectively lead to a clear diagnosis in the current evidence setting. The model needs some additional definitions to be compatible with this tool:

- At least one node needs to be assigned the role of a hypothesis. This is a primary fault node and the target is to find out its belief (a probability distribution).
- At least two nodes need to be made informational. They are candidates for the recommendation list.

Mutual information score determines the amount of weight or "lift" that evidence about the state of each information node would bring to each hypothesis variable.

Decision-theoretic troubleshooting is supported for specially crafted Bayesian network models. The idea is to choose a problem type and then iterate between requesting recommendations for next actions and making observations or repairs. While making recommendations the algorithm considers alternative actions and arranges the actions according to their efficiency (likelihood to succeed divided by its cost).

Single-fault hypothesis is assumed - only one fixable component is supposed to be broken during one session.

The network nodes need to be assigned certain roles and costs (fix/observation):

- At least one problem-defining node needs to be defined. It is the primary symptom of a failure and the target of the diagnosis. The first state has to define the normal operation.
- Evidence is defined as informational.
- Primary faults are defined as fixable. They have to be root nodes.
- All fixable nodes have to be root nodes in order for the troubleshooting to work.
- A problem–defining node cannot be a root node. It has to be a child of some fixable nodes in order to run the troubleshooting session.

One initiates the session by setting the desired problem-defining node to an abnormal state and then gets a list of recommendation of actions to follow (both observations and fix attempts). Only one problem-defining node is considered during a certain troubleshooting session.

The usability of the prototyping tool, MSBNx, was found fairly good. However, it would have been useful if the tool had provided support for logging the selected actions and had calculated the total repair cost for the whole session. It would also have been useful to have a facility to emphasize the largest differences in the beliefs from their previous state. These changes can be made in a customized user interface using the programming API.

Appendix B: Descriptions for the MIB variables used in the examples

MIB stands for Management Information Base and it is a local storage place for managed element specific data values. By monitoring these values, it is possible to maintain an up-to-date status of the (ethernet-like) interfaces of the monitored host.

ifSpeed
: The current operational speed of the interface in bits per second.

ifInOctets
: The number of octets in valid MAC frames received on this interface.

ifOutOctets
: The number of octets in valid MAC frames transmitted from this interface.

ifInErrors
: The sum of errors for this interface related to incoming traffic

ifOutErrors
: The sum of errors for this interface related to outgoing traffic

ifInNUcastPkts
: The number of non-unicast (i.e., subnetwork- broadcast or subnetwork-multicast) packets delivered to a higher-layer protocol.

ifOutQLen
: The length of the output packet queue (in packets).

ifOutDiscards
: The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.

dot3StatsFrameTooLong
: Number of received frames that were bigger than the maximum size permitted.

ipInDiscards
: The number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g., for lack of buffer space). Note that this counter does not include any datagrams discarded while waiting re-assembly.

tcpCurrEstab
: The number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.

tcpInDupSegs
: The number of TCP input partial duplicate segments.

tcpOutRsts
: The number of TCP segments sent containing the RST flag.

Appendix C: The Bayesian network description for the TCP problem: the troubleshooting case

```
network "bndefault"
{
     version is 1;
     creator is "";
     format is "DSC";
}

properties
{
     type DTASDG_Notes = array of string,
          "Notes on the diagram";
     type DTAS_ColorType = string;
     type MS_Addins = array of string;
     type MS_Asym = string,
          "Asymmetric assessment information stored as a string";
     type MS_cost_fix = real,
          "Cost To Fix";
     type MS_cost_observe = real,
          "Cost To observe";
     type MS_label = choice of
          [other,informational,problem,fixunobs,fixobs],
          "Troubleshooting Network Node Types";
     DTAS_ColorType = "MS_label";
     MS_cost_fix = 1000;
}

node BufferSize_Too_Small
{
     name = "BufferSize_Too_Small";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (7330, 3810);
     MS_cost_fix = 70;
     MS_label = fixunobs;
}

node CPU_overloaded
{
     name = "Is the CPU overloaded?";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (847, 9498);
     MS_cost_fix = 100;
```

```
     MS_label = fixunobs;
}

node Delivers_datagram_rate
{
     name = "delivers_datagram_rate";
     type = discrete[3]
     {
          "Low",
          "Normal",
          "High"
     };

     position = (12754, 7938);
     MS_cost_observe = 7;
     MS_label = informational;
}

node HostConnectionError
{
     name = "HostConnectionError";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (7197, -2990);
     MS_label = problem;
}

node IfInOctets
{
     name = "ifInOctets: The number of octets in valid MAC frames
received on this interface";
     type = discrete[3]
     {
          "low",
          "normal",
          "high"
     };

     position = (20055, 7091);
     MS_cost_observe = 5;
     MS_label = informational;
}

node IfLoopBackTest
{
     name = "Indicates whether the interface has problems using a
loopback test.";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (20797, 3174);
```

```
     MS_cost_observe = 25;
     MS_label = informational;
}

node Input_frame_rate
{
     name = "input_frame_rate";
     type = discrete[3]
     {
          "Low",
          "Normal",
          "High"
     };

     position = (14156, 4180);
     MS_Asym = "(Line_UP ((\"No\") ())((\"Yes\")
(Interface_Problem ((\"No\") ())((\"Yes\") ())))))";
     MS_cost_observe = 7;
     MS_label = informational;
}

node Interface_Problem
{
     name = "Indicates whether Interface has hardware or software
problems.";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (17569, 397);
     MS_cost_fix = 200;
     MS_label = fixunobs;
}

node IpInDiscards
{
     name = "IpInDiscards: The number of input IP datagrams for
which no problems were encountere"
          "d to prevent their continued processing but were
discarded";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (6297, 7275);
     MS_cost_observe = 5;
     MS_label = informational;
}

node Line_UP
{
     name = "Indicates whether the line is up.";
     type = discrete[2]
     {
```

```
          "Yes",
          "No"
     };

     position = (10424, 609);
     MS_cost_fix = 30;
     MS_cost_observe = 15;
     MS_label = fixobs;
}

node TcpCurrEstab
{
     name = "tcpCurrEstab: The number of TCP connections for which
the current state is either ES"
          "TABLISHED or CLOSE-WAIT.";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (873, 14314);
     MS_cost_observe = 5;
     MS_label = informational;
}

node TcpInDupSegs
{
     name = "TcpInDupSegs: The number of TCP input partial
duplicate segments";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (17674, 12171);
     MS_cost_observe = 5;
     MS_label = informational;
}

node TcpOutResets
{
     name = "tcpOutRsts: The number of TCP segments sent
containing the RST flag";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (11536, 14790);
     MS_cost_observe = 5;
     MS_label = informational;
}

probability(BufferSize_Too_Small)
{
```

```
     0.99, 0.01;
}

probability(CPU_overloaded)
{
     0.996, 0.004;
}

probability(Delivers_datagram_rate|BufferSize_Too_Small,Input_fram
e_rate)
{
     default = 0.333333, 0.333333, 0.333333;
     (0, 0) = 0.8, 0.17, 0.03;
     (0, 1) = 0.1, 0.8, 0.1;
     (0, 2) = 0.01, 0.09, 0.9;
     (1, 0) = 0.999, 0.0001, 0;
     (1, 1) = 0.99, 0.009, 0.001;
     (1, 2) = 0.9, 0.09, 0.01;
}

probability(HostConnectionError|Line_UP,Interface_Problem,BufferSi
ze_Too_Small,CPU_overloaded)
{
     function = max;
     (0, 0, 0, 0) = 0.999, 0.001;
     (1, 0, 0, 0) = 1, 0;
     (0, 1, 0, 0) = 1, 0;
     (0, 0, 1, 0) = 1, 0;
     (0, 0, 0, 1) = 1, 0;
}

probability(IfInOctets|Input_frame_rate)
{
     default = 0.333333, 0.333333, 0.333333;
     (0) = 0.95, 0.045, 0.005;
     (1) = 0.025, 0.95, 0.025;
     (2) = 0.0001, 0.0049, 0.95;
}

probability(IfLoopBackTest|Interface_Problem)
{
     (0) = 0.99, 0.01;
     (1) = 0.001, 0.999;
}

probability(Input_frame_rate|Line_UP,Interface_Problem)
{
     default = 0.333333, 0.333333, 0.333333;
     (0, 0) = 0.001, 0.90901, 0.08999;
     (0, 1) = 0.969903, 0.029997, 9.999e-005;
     (1, 0) = 0.990001, 0.00989901, 0.0001;
     (1, 1) = 0.990001, 0.00989901, 0.0001;
}

probability(Interface_Problem)
{
     0.999, 0.001;
}
```

```
probability(IpInDiscards|BufferSize_Too_Small)
{
     (0) = 0.99, 0.01;
     (1) = 0.001, 0.999;
}

probability(Line_UP)
{
     0.98, 0.02;
}

probability(TcpCurrEstab|CPU_overloaded)
{
     (0) = 0.98, 0.02;
     (1) = 0.1, 0.9;
}

probability(TcpInDupSegs|Delivers_datagram_rate)
{
     (0) = 0.9, 0.1;
     (1) = 0.6, 0.4;
     (2) = 0.15, 0.85;
}

probability(TcpOutResets|Delivers_datagram_rate,CPU_overloaded)
{
     (0, 0) = 0.95, 0.05;
     (0, 1) = 0.3, 0.7;
     (1, 0) = 0.9, 0.1;
     (1, 1) = 0.1, 0.9;
     (2, 0) = 0.7, 0.3;
     (2, 1) = 0.05, 0.95;
}
```

Appendix D: The Bayesian network description for the TCP problem: the diagnostic case

```
network "bndefault"
{
     version is 1;
     creator is "";
     format is "DSC";
}

properties
{
     type DTASDG_Notes = array of string,
          "Notes on the diagram";
     type DTAS_ColorType = string;
     type MS_Addins = array of string;
     type MS_Asym = string,
          "Asymmetric assessment information stored as a string";
     type MS_label = choice of

     [other,hypothesis,informational,problem,fixobs,fixunobs,unfix
able,configuration],
          "Diagnostic Network Node Types";
     DTAS_ColorType = "MS_label";
}

node BufferSize_Too_Small
{
     name = "BufferSize_Too_Small";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (7330, 3810);
     MS_label = hypothesis;
}

node CPU_overloaded
{
     name = "Is the CPU overloaded?";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (847, 9498);
     MS_label = hypothesis;
}

node Delivers_datagram_rate
```

```
{
     name = "delivers_datagram_rate";
     type = discrete[3]
     {
          "Low",
          "Normal",
          "High"
     };

     position = (12754, 7938);
     MS_label = informational;
}

node IfInOctets
{
     name = "ifInOctets: The number of octets in valid MAC frames
received on this interface";
     type = discrete[3]
     {
          "low",
          "normal",
          "high"
     };

     position = (20055, 7091);
     MS_label = informational;
}

node IfLoopBackTest
{
     name = "Indicates whether the interface has problems using a
loopback test.";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (20797, 3174);
     MS_label = informational;
}

node Input_frame_rate
{
     name = "input_frame_rate";
     type = discrete[3]
     {
          "Low",
          "Normal",
          "High"
     };

     position = (14156, 4180);
     MS_Asym = "(Line_UP ((((\"No\") ()))((\"Yes\")
(Interface_Problem ((((\"No\") ()))((\"Yes\") ()))))))";
     MS_label = informational;
}
```

```
node Interface_Problem
{
     name = "Indicates whether Interface has hardware or software
problems.";
     type = discrete[2]
     {
          "No",
          "Yes"
     };

     position = (17569, 397);
     MS_label = hypothesis;
}

node IpInDiscards
{
     name = "IpInDiscards: The number of input IP datagrams for
which no problems were encountere"
          "d to prevent their continued processing but were
discarded";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (6297, 7275);
     MS_label = informational;
}

node Line_UP
{
     name = "Indicates whether the line is up.";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (10424, 609);
     MS_label = hypothesis;
}

node TcpCurrEstab
{
     name = "tcpCurrEstab: The number of TCP connections for which
the current state is either ES"
          "TABLISHED or CLOSE-WAIT.";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (873, 14314);
     MS_label = informational;
}
```

```
node TcpInDupSegs
{
     name = "TcpInDupSegs: The number of TCP input partial
duplicate segments";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (17674, 12171);
     MS_label = informational;
}

node TcpOutResets
{
     name = "tcpOutRsts: The number of TCP segments sent
containing the RST flag";
     type = discrete[2]
     {
          "Normal",
          "High"
     };

     position = (11536, 14790);
     MS_label = informational;
}

probability(BufferSize_Too_Small)
{
     0.99, 0.01;
}

probability(CPU_overloaded)
{
     0.996, 0.004;
}

probability(Delivers_datagram_rate|BufferSize_Too_Small,Input_fram
e_rate)
{
     default = 0.333333, 0.333333, 0.333333;
     (0, 0) = 0.8, 0.17, 0.03;
     (0, 1) = 0.1, 0.8, 0.1;
     (0, 2) = 0.01, 0.09, 0.9;
     (1, 0) = 0.999, 0.0001, 0;
     (1, 1) = 0.99, 0.009, 0.001;
     (1, 2) = 0.9, 0.09, 0.01;
}

probability(IfInOctets|Input_frame_rate)
{
     default = 0.333333, 0.333333, 0.333333;
     (0) = 0.95, 0.045, 0.005;
     (1) = 0.025, 0.95, 0.025;
     (2) = 0.0001, 0.0049, 0.95;
}
```

```
probability(IfLoopBackTest|Interface_Problem)
{
     (0) = 0.99, 0.01;
     (1) = 0.001, 0.999;
}

probability(Input_frame_rate|Line_UP,Interface_Problem)
{
     default = 0.333333, 0.333333, 0.333333;
     (0, 0) = 0.001, 0.90901, 0.08999;
     (0, 1) = 0.969903, 0.029997, 9.999e-005;
     (1, 0) = 0.990001, 0.00989901, 0.0001;
     (1, 1) = 0.990001, 0.00989901, 0.0001;
}

probability(Interface_Problem)
{
     0.999, 0.001;
}

probability(IpInDiscards|BufferSize_Too_Small)
{
     (0) = 0.99, 0.01;
     (1) = 0.001, 0.999;
}

probability(Line_UP)
{
     0.98, 0.02;
}

probability(TcpCurrEstab|CPU_overloaded)
{
     (0) = 0.98, 0.02;
     (1) = 0.1, 0.9;
}

probability(TcpInDupSegs|Delivers_datagram_rate)
{
     (0) = 0.9, 0.1;
     (1) = 0.6, 0.4;
     (2) = 0.15, 0.85;
}

probability(TcpOutResets|Delivers_datagram_rate,CPU_overloaded)
{
     (0, 0) = 0.95, 0.05;
     (0, 1) = 0.3, 0.7;
     (1, 0) = 0.9, 0.1;
     (1, 1) = 0.1, 0.9;
     (2, 0) = 0.7, 0.3;
     (2, 1) = 0.05, 0.95;
}
```

Appendix E: The Bayesian network description for the microwave link problem in a GSM access network

```
network
{
}

properties
{
     type DTASDG_Notes = array of string,
          "Notes on the diagram";
     type MS_Addins = array of string;
     type MS_label = choice of

     [other,hypothesis,informational,problem,fixobs,fixunobs,unfix
able,configuration],
          "Diagnostic Network Node Types";
}

node BS1_alarm
{
     name = "BS1_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (19236, 2955);
     MS_label = informational;
}

node BS2_alarm
{
     name = "BS2_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (19271, 5345);
     MS_label = informational;
}

node BS3_alarm
{
     name = "BS3_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };
```

```
     position = (19386, 7735);
     MS_label = informational;
}

node BS4_alarm
{
     name = "BS4_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (19393, 10019);
     MS_label = informational;
}

node BS5_alarm
{
     name = "BS5_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (19507, 12119);
     MS_label = informational;
}

node BSC_conn_alarm
{
     name = "BSC_conn_alarm";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (19236, 476);
     MS_label = informational;
}

node MWLink1_OK
{
     name = "MWLink1_OK";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (633, 925);
     MS_label = hypothesis;
}

node MWLink2_OK
{
```

```
     name = "MWLink2_OK";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (694, 3978);
     MS_label = hypothesis;
}

node MWLink3_OK
{
     name = "MWLink3_OK";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (615, 6200);
     MS_label = hypothesis;
}

node MWLink4_OK
{
     name = "MWLink4_OK";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (598, 8749);
     MS_label = hypothesis;
}

node MWLink5_OK
{
     name = "MWLink5_OK";
     type = discrete[2]
     {
          "Yes",
          "No"
     };

     position = (554, 11219);
     MS_label = hypothesis;
}

probability(BS1_alarm|MWLink1_OK)
{
     (0) = 0.0001, 0.9999;
     (1) = 0.999, 0.001;
}

probability(BS2_alarm|MWLink1_OK,MWLink2_OK)
{
```

```
     default = 0.999, 0.001;
     (0, 0) = 0.001, 0.999;
}

probability(BS3_alarm|MWLink1_OK,MWLink3_OK)
{
     default = 0.999, 0.001;
     (0, 0) = 0.0001, 0.9999;
}

probability(BS4_alarm|MWLink1_OK,MWLink4_OK,MWLink3_OK)
{
     default = 0.999, 0.001;
     (0, 0, 0) = 0.0001, 0.9999;
}

probability(BS5_alarm|MWLink1_OK,MWLink5_OK,MWLink4_OK,MWLink3_OK)
{
     default = 0.999, 0.001;
     (0, 0, 0, 0) = 0.0001, 0.9999;
}

probability(BSC_conn_alarm|MWLink1_OK)
{
     (0) = 0.0002, 0.9998;
     (1) = 0.999, 0.001;
}

probability(MWLink1_OK)
{
     0.99, 0.01;
}

probability(MWLink2_OK)
{
     0.99, 0.01;
}

probability(MWLink3_OK)
{
     0.99, 0.01;
}

probability(MWLink4_OK)
{
     0.99, 0.01;
}

probability(MWLink5_OK)
{
     0.99, 0.01;
}
```

Appendix F: The Bayesian network description for the Linear Lightwave Network (LLN) case

```
network
{
}

properties
{
     type DTASDG_Notes = array of string,
          "Notes on the diagram";
     type MS_Addins = array of string;
     type MS_label = choice of

     [other,hypothesis,informational,problem,fixobs,fixunobs,unfix
able,configuration],
          "Diagnostic Network Node Types";
}

node A
{
     name = "A";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (7117, 5398);
     MS_label = informational;
}

node B
{
     name = "B";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (7039, 9656);
     MS_label = informational;
}

node C
{
     name = "C";
     type = discrete[2]
     {
          "OK",
          "No"
     };
```

```
     position = (16035, 8017);
     MS_label = informational;
}

node Input1
{
     name = "Input1";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (6721, 1481);
     MS_label = informational;
}

node Input2
{
     name = "Input2";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (16404, 1747);
     MS_label = informational;
}

node LDC1
{
     name = "LDC1";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (1112, 2938);
     MS_label = hypothesis;
}

node LDC2
{
     name = "LDC2";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (1165, 6958);
     MS_label = hypothesis;
}

node LDC3
```

```
{
     name = "LDC3";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (22489, 3413);
     MS_label = hypothesis;
}

node LDC4
{
     name = "LDC4";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (1447, 11642);
     MS_label = hypothesis;
}

node LDC5
{
     name = "LDC5";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (22861, 9578);
     MS_label = hypothesis;
}

node O1
{
     name = "O1";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (4103, 15345);
     MS_label = informational;
}

node O2
{
     name = "O2";
     type = discrete[2]
     {
          "OK",
          "No"
```

```
     };

     position = (12463, 15345);
     MS_label = informational;
}

node O3
{
     name = "O3";
     type = discrete[2]
     {
          "OK",
          "No"
     };

     position = (20745, 15319);
     MS_label = informational;
}

probability(A|Input1,LDC1)
{
     function = max;
     (0, 0) = 0.999, 0.001;
     (1, 0) = 0.001, 0.999;
     (0, 1) = 0.001, 0.999;
}

probability(B|A,LDC2)
{
     function = max;
     (0, 0) = 0.999, 0.001;
     (1, 0) = 0.001, 0.999;
     (0, 1) = 0.001, 0.999;
}

probability(C|Input2,LDC3)
{
     function = max;
     (0, 0) = 0.999, 0.001;
     (1, 0) = 0.001, 0.999;
     (0, 1) = 0.001, 0.999;
}

probability(Input1)
{
     0.9999, 0.0001;
}

probability(Input2)
{
     0.9999, 0.0001;
}

probability(LDC1)
{
     0.999, 0.001;
}
```

```
probability(LDC2)
{
     0.999, 0.001;
}

probability(LDC3)
{
     0.999, 0.001;
}

probability(LDC4)
{
     0.999, 0.001;
}

probability(LDC5)
{
     0.999, 0.001;
}

probability(O1|LDC4,B)
{
     function = max;
     (0, 0) = 0.999, 0.001;
     (1, 0) = 0.001, 0.999;
     (0, 1) = 0.001, 0.999;
}

probability(O2|C,LDC4)
{
     function = max;
     (0, 0) = 0.999, 0.001;
     (1, 0) = 0.001, 0.999;
     (0, 1) = 0.001, 0.999;
}

probability(O3|C,LDC5,B)
{
     function = max;
     (0, 0, 0) = 0.999, 0.001;
     (1, 0, 0) = 0.001, 0.999;
     (0, 1, 0) = 0.001, 0.999;
     (0, 0, 1) = 0.001, 0.999;
}
```