IMPLEMENTATION OF MULTIPLY SECTIONED BAYESIAN NETWORK MANAGEMENT SOFTWARE

DEPARTMENT OF COMPUTER SCIENCE

SUBMITTED BY:
THOMAS H. BRIGGS, VI
(thb@ship.edu)
SUNDAY, APRIL 01, 2001

ADVISOR:
DR. CAROL WELLINGTON
(cawell@ship.edu)

MOTIVATION AND BACKGROUND

Fault detection and analysis is an important problem domain. Many computational models exist to successfully address the general needs of this problem domain. Most of these traditional approaches rely on heuristic information provided by experts. Typically, these systems are designed with the assumption that the heuristic provided by the expert is correct, and information presented by the problem environment will be consistent. [3],[10], [11].

Common deficiencies in these systems include reconfigureability and uncertainty. Uncertainty refers to the potential for incorrect or incomplete information being presented by the environment. Dropped messages or conflicting fault symptoms provide challenges to many traditional fault detection systems. Reconfigureability describes the need for fault detection systems to be readily reconfigured as the environment in which they operate changes. Reconfiguration events happen in the physical configuration of the monitored devices change. These events also happen as a result of incompatibilities developing between different devices. [10], [11]

Public Telephone Network

The public telephone network (PTN) is a problem domain that presents these challenges to fault detection systems. The PTN is huge collection of interconnected devices over heterogeneous transmission media. Reconfiguration events happen frequently throughout the network as devices added or removed and connection paths are opened or closed. Inconsistent information about the state of the network is common as a result of the huge number of devices with potential interoperable incompatibilities. [10],[11].

This project considered fault detection in a small subset of the PTN domain. Unidirectional OC-3 rings are one of the basic building blocks of the PTN. The rings are synchronized by a single clock, and can re-route communications if a fault is detected within 50ms without dropping a call. This creates one additional constraint on a fault detection system in this problem domain. The analyzer must be able to detect, plan, and respond to a fault within 50ms to avoid dropping calls.

The PTN is also a dynamic environment. Service engineers frequently make changes to the physical network. Device configurations change automatically as a result of fault detection protocols in their firmware. Fault detection systems must be able to reconfigure themselves as a result of these reconfiguration events. Because the domain is fluid, the time requirements of reconfiguration must be minimized.

Finally, the PTN presents a unique challenge as a result of its composition. The devices that are physically connected to the network may be of different vendors, models, configuration, software revision, and age. Proper diagnosis will require a sophisticated learning and probability model to identify factors that are the root cause of a problem. For example, if two revisions of firmware are incompatible, the symptom may appear to be a physical problem. In this example, the fault analysis system must be able to learn that firmware is not compatible and is the true cause of the failure.

Bayesian Belief Network

Bayesian Belief Networks are used in many fault detection systems in other problem domains. In [3], Heckerman describes a system of fault analysis using relatively simple BBN's in the small problem domains such as printer fault diagnosis.

Bayesian Belief Networks are based on probabilistic reasoning . BBNs are often built to reason under uncertainty. However, reconfiguration and computational run-times are two significant challenges to the traditional BBN model.

BBNs are extremely computationally expensive, and require significant run-times. Ignoring the reconfiguration requirements, simply propagating belief through the network is NP-Hard [2]. As the size of the BBN grows, the time required to propagate information through the network increases exponentially.

Multiply Sectioned Bayesian Belief Networks

Dr. Yang Xiang [12] demonstrated that large BBNs tend to exhibit a locality property. That is, that evidence entered into the network tends to ripple to nodes in a given locale, determined by the configuration of the network. Small changes in the state of the network tend to not ripple throughout the entire network.

Xiang uses this locality property of large BBNs to create Multiply Sectioned Bayesian Networks (MSBN) that are composed of one or more sections. Each section of the MSBN contains those nodes that are "local" to each other with respect to their inference propagation.

In previous research done at Shippensburg University, by Janet Pitkin [5], a protocol was developed to construct sound sections from a set of individual BBNs with a set of overlapping nodes. At the completion of this algorithm, each of the original sub-networks contains the nodes that participate in a clique that crosses a network boundary. This meets the criteria from [12] for a "sound sectioning." By ensuring a sound sectioning, evidence can propagate through the network in a statistically sound manner.

These algorithms rely on "cliques" of nodes. A clique is a maximal totally connected subgraph, where the clique contains all, and only, those nodes which

are directly connected to all other nodes in the clique. Cliques' role in these algorithm are to identify those subgraphs that are semantically related. Evidence entered into a node and the corresponding alteration of its probability is likely to have a resultant alteration of the probability of the other nodes in the clique.

Propagating information through the MSBN is expected to be more efficient than a comparable "un-sectioned" network. When new evidence is entered into a traditional network, the effects of that update must be propagated to all other nodes in the network and the results must propagate back to the original node.

This process is similar for each section of the MSBN. In the best case, evidence entered into one section propagates to nodes in the same local section [see *locality property* pp. 30]. In the case that new evidence must be propagated through other sections, the MSBN method will limit the propagation to only those sections that need to be updated based on the locality of subsequent localized changes. In the degenerate case, belief entered into one node must be propagated throughout the entire network, resulting in slightly worse performance than if the network were not sectioned at all. The degradation results from evaluating the over lapping nodes to determine whether to propagate changes or not.

The degree of speed-up achieved from this method will be relative to the structure of the network. The ratio of boundary nodes (those nodes that interconnect sections) to non-boundary nodes will be of utmost importance. The lower the ratio (i.e. relatively low number of boundary nodes to a high number of non-boundary nodes) will yield higher speed-ups. This derives from the propagation algorithm that must evaluate the status of every boundary node whenever new evidence is entered into the network. The more boundary nodes there are, the more time is required in comparing their states.

For the Public Telephone Network (PTN) problem, the locality property is expected to create a large number of relatively small sections of networks with a high degree of locality. This expectation is derived from the physical structure of the PTN network. Telephones are connected to local switches, local switches are connected to regional switches, which are in turn connected to larger back bone connections. Intuitively, it is obvious that a failure of one telephone is not likely to be caused by a failure of the national communications backbone. The cause of a single phone's failure is likely to be localized to that telephone and the wires and devices that connect it to the local or regional switches.

The PTN is designed to have a high degree of localization and fault isolation. As an example of the extreme opposite, consider the shared bus Ethernets that were prevalent in the 1980's. One defective network card could consume all available bandwidth on the shared media, and create a distributed failure that was not localized to its connection to the network. Certainly, the PTN does not fall into this category, as a failure of a single homeowner's telephone is not likely to cause the entire global telecommunications infrastructure to crash. The MSBN approach is thus an appropriate candidate for the PTN problem domain. Failures detected by the network are expected to exhibit highly localized behavior [10], [11].

Scope of this Project

The scope of this project is to implement an MSBN management package. The package will be implemented over a commercial Bayesian Belief Network. The goal of this package is to demonstrate the feasibility of implementing MSBNs, and to allow further research into the applicability of MSBNs to the PTN fault-detection environment.

IMPLEMENTATION

Implementation of the MSBN manager involved developing an API that extended a commercial belief network package. The extensions provide specialized functionality for managing multiple MSBNs, and many sections within each MSBN. There are two primary functions the API provides. First, the API provides the mechanisms to create sound sections of an MSBN from one or more BBNs. Second, the API provides the mechanism to manage the propagation of evidence throughout the sound sections of the network.

Java was chosen as the development language. Java offers rapid prototyping, developing, and documenting. It is a portable language and can be moved from platform to platform, and potentially embedded in hardware devices. Furthermore, Java's implementation of inheritance and polymorphism allowed development of very elaborate data structures without the overhead of instantiating generic template classes as is commonly done in C++.

Java's advanced memory management also makes it an ideal choice. Memory is allocated and freed automatically by the environment. Buffer overflows are reduced, as Java nearly prevents access to specific pointers and accidental clobbering of data structures. Objects are finalized when they are no longer referenced, and the memory is reclaimed by Java's garbage collector. This is a tremendous improvement over traditional languages such as C or C++. This increases program stability and reduces debugging time during development by removing the most common developmental bugs.

Additionally, the future needs of the project were considered. Java's interoperability with other systems, databases, and object brokers is well known.

This suggests that Java will be an ideal language for continued development of the project in the future, allowing it to scale up to larger systems and interoperate with advanced database systems.

Runtime is a common concern using Java. Java's runtime environment is built on the byte-code interpreter, unlike other complied languages that are executed directly by the system's processor. Java programs typically do run slower in a side-by-side comparison with other "traditional" languages. Some of the slower performance of Java is offset by the expressibility of the language. The advanced data structures that are provided by the language often lead to more elegant solutions that carry more efficient runtimes. Traditional languages often lack these structures, and programmers tend to shy away from developing them, instead favoring more complex or slower algorithms. Finally, the bulk of the run-time occurs in the Netica libraries, which are ultimately written in C language, and executed as platform "native" code [8]. This mitigates some of the performance degradation caused by Java.

Norsys Netica Application Programmer Interface

Norsys' Netica application interface was chosen to manage the Bayesian Belief networks. Netica's development environment consists of libraries for a number of target platforms, and a graphical user interface for Windows and Macintosh, and a set of object wrappers, Netica-J, for Java [7], [8]. Netica-J was released to Shippensburg University pre-release 2.09, an alpha version, in October 2001 [9]. Subsequently, Netica-J was officially released January 2002 [6]. The final release contained improvements that yield incompatibilities with applications written using the pre-release, including the MSBN manager.

Netica's graphical user interface will allow experts to construct networks manually or to view the networks created by the software application. The Netica API

contains statements that allow belief network construction, and reconfiguration. The API also includes statements that facilitate entering findings into the network and querying the network for results.

Two drawbacks to using the Netica API became apparent as the project progressed. First, Netica is a commercial package[7], and secondly, the MSBN manager must shadow Netica's internal network construction.

Netica is a commercial package, and deployment of a production program would require a license to use Netica. This is not a significant drawback, since the current licensing terms are generally reasonable, and special discounts can often be negotiated depending on circumstances.

Shadowing the Netica structures was also easily overcome. Shadowing refers to the technique of augmenting a lower-level API and having to replicate the services provided by that API. Problems arose from Netica's requirement that `Node` objects are tied to `Net` objects; and from the specification of directed links. Many of the algorithms used by the MSBN manager assume a existential property for nodes, and freely copy nodes between networks, eg. The Reduction by Peeling Algorithm[1]. Netica instantiates nodes within a specific network and the API does not provide methods to treat a given node universally, such as equality tests[7]. Also, Netica is designed to use Directed Acyclic Graphs (DAGs), and there are no provisions to support the Hypergraph model, nor the moral links used in various algorithms ([1] and). The MSBN provides these functions through the shadowed API which can exchange the names of the nodes between networks and maintain the special adjacency structures.

Overall, Netica proved to be a stable platform that provided excellent development tools and documentation. Development time was reduced as a

result of using the product. Consistent Belief Networks were created using both the GUI tool and through the Netica API. Resulting networks were stored as files and reviewed using the GUI tool. Finally, being able to interchange networks between the API and the GUI made it convenient to double check the probabilities generated by the networks.

The MSBN Package

The MSBN package was written in Java, using the Standard Development Kit v1.3, under the Forte for Java editor. The package was developed to use Netica-J, which currently is available only for the Windows operating system. The MSBN package performs several tasks:

- Instantiate linkages between networks
- Import Netica sub-networks into the MSBN manager
- Construct an MSBN from the Netica sub-nets
- Apply Pitkin's algorithm to identify overlapping cliques
- Copy nodes between sub-nets to fulfill Xiang's "sound-sectioning" principle
- Return modified sub-nets to original application
- Manage inference propagation between the individual sub-nets

The MSBN manager was designed to be instantiated separately from the Netica libraries. A program can call upon the manager's functions on an "as-needed" basis. All other times, the program can interact directly with the Netica API.

Instantiate links between networks

The MSBN manager provides the ADT `netLinks` to manage the storage of network links. The class stores the name of a node that is shared between two networks, the names of the networks sharing the node, and the maximum amount of difference that can exist between the states of the nodes in each of their networks. The MSBN manager stores a linked list of these `netLinks`. As the structure of the network is reconfigured and nodes are copied between sub-nets, the list of links must be updated with the new links between networks.

This method of representing the links between networks is one way in which this could have been accomplished. The way that was not chosen was to establish the node and network of each end of a directed link between networks. In this model, instead of sharing the node, the networks would share a link between nodes. This method makes it more difficult to represent the probability tables between the networks (since the nodes would not share each other's inputs). It would also be more difficult to copy the nodes around between sub-nets as a result of other operations of the manager.

Import Netica sub-networks into the MSBN manager

Individual Netica sub-networks must be imported into the MSBN manager. There may be many MSBN managers, and many sub-networks co-existing in an application's space. Therefore, no assumptions may be made about the networks that are involved in a given MSBN.

Using the Netica network as the model for the MSBN sub-network allows network libraries to be created and stored on disk. These libraries can be created by an expert, or through pasts run of the MSBN manager. Networks can be read and written using the Netica input and output library routines. They can also be

created at run-time using the Netica API. However the Netica network is created, it can be treated the same by the MSBN manager.

The MSBN manager internally records the Netica networks that are imported. The manager compiles auxiliary data structures about each network to make retrieval of that networks' information faster or to represent information that is not present in the original Netica networks. For example, if a moral link is added between two nodes, that link cannot be represented in Netica, but it is added to the adjacency list that is stored in the manager.

Construct an MSBN from the Netica sub-nets

An MSBN is created from the imported sub-nets. The MSBN is created as a Netica network that contains all of the nodes and linkages of the sub-networks. This intermediate network is necessary to support later operations on the entire network. This intermediate network can potentially be quite large and require significant memory for storage. Therefore, it is only used temporarily, and discarded after the reconfiguration operations are complete. Limitations on the size of this intermediate representation are solely from the Netica API.

Apply Pitkin's algorithm to identify overlapping cliques

In [5], Janet Pitkin describes the steps necessary to construct an MSBN from a series of sub-networks (sub-DAGs). These steps essentially ensure that the MSBN forms a sound network from the sub-networks. The algorithm uses techniques such as Triangulation, Moralization, Cycle-Detection, and Junction Forest Creation.

The Triangulation algorithm used by this method is the standard algorithm described in many textbooks and websites. Essentially, cycles of length > 3 are found, and links are added between nodes to ensure that no cycle is of length > 3.

The Moralization algorithm searches the network for each node. For each node, its parents are retrieved, and using the auxiliary structures created when the network is imported, the "owning" network; or network that contributed the node, is retrieved to determine whether the parents of a node come from different networks. If they do, a "Moral Link" is added between those nodes. This link is added only to the auxiliary structures in the MSBN manager. One change to Pitkin's algorithm is that if only the parents are connected between networks, orphaned nodes will result. Therefore, any cliques involving the parents to be copied must also be copied between networks.

The Cycle Detection algorithm uses a common depth-first three color paint algorithm. First, all of the nodes of the MSBN are "painted" white. Then, as each node is visited, it is "painted" gray. If any child of a node is not white, then a cycle has been found. Each child is recursively visited. Finally, when all children are visited, the node is "painted" black, and the function returns up a level. When cycles are discovered, their paths are added to a linked list for later processing.

Finally, Junction Forests are created using Almond's "Reduce by Peeling" [1] algorithm. The resulting tree of cliques is used to determine nodes missing from the D-Sepset. That list of nodes is then used to identify nodes that must be exchanged between networks to fulfill this requirement.

Copy nodes between sub-nets to fulfill Xiang's "sound-sectioning" principle

The nodes identified by the prior processes are copied between sub-networks. The nodes that are identified essentially comprise the linkage hosts that ensure sound-sectioning between sub-nets. After completion of this, sub-networks may be evaluated independently of each other.

Manage inference propagation between the individual sub-nets

As evidence is entered into the individual sub-networks, localized changes in the state of the network are expected to remain local to that network. However, some changes will not be localized, and must be propagated through the entire network. A propagation manager was developed to provide this functionality.

The propagation manager is initialized with the individual networks that were adjusted through the previous algorithms; and the resulting network links. As evidence is entered into the network, the user's application can trigger a propagation request.

Propagation begins by iterating through the list of network links, and comparing each state of the nodes in each of their component networks. If the difference of any one state of either node differs more than the acceptable limit, the findings are propagated through the network. The propagation manager will recursively propagate changes throughout the network until all nodes have settled into new states.

RESULTS

The MSBN Manager was implemented and several test suites were created. The first set of tests demonstrated its accuracy in creating networks. The second set of tests demonstrated the performance characteristics. The results of these tests are presented. A summary of the conclusions reached from these results is presented after.

Sound Sectioning

One important goal of the MSBN Manager is to ensure sound sectioning according to Xiang's rules. Consider the following sub-networks:

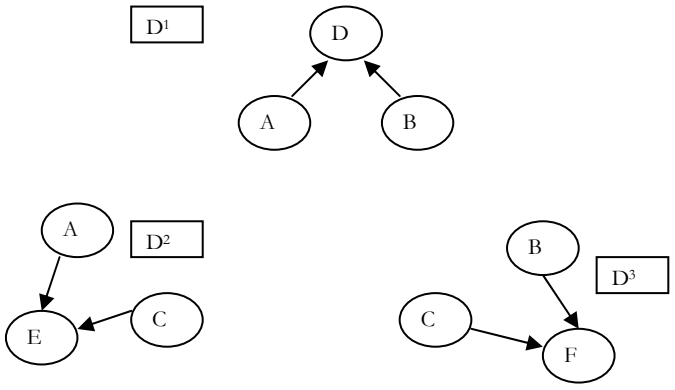


Figure 1 – First test network

Each of the three sub-nets (sub-DAGs) above need to be compiled to share the overlapping cliques. The shared nodes are: A, B, and C. In this case, there are cliques formed by `{ADB}`, `{AEC}`, and `{CFB}`. Each of these linkage cliques need to be copied into the other networks. Upon completion of Pitkin's

algorithm, each of the sub-DAGs contains all of the nodes of the other networks and the network linkages now contain references to each of the copied nodes.

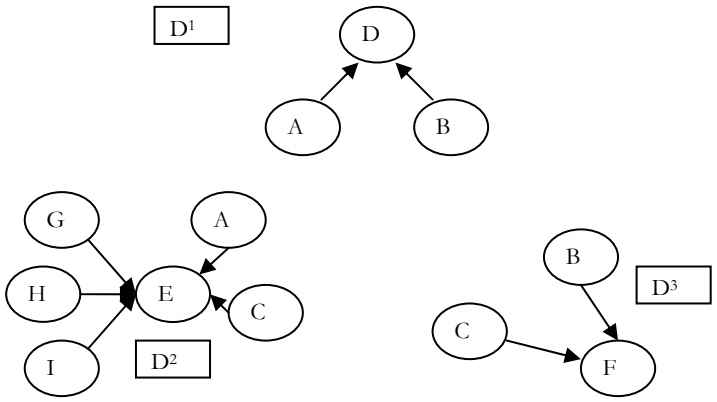


Figure 2 – Second test network

In the second test network, several children were added to node E. In this case, the same copying occurred with the nodes A, B, C, D, E, and F. The children of E were not part of any linkage cliques with any other nodes in other networks, so they remain only in network D^2.

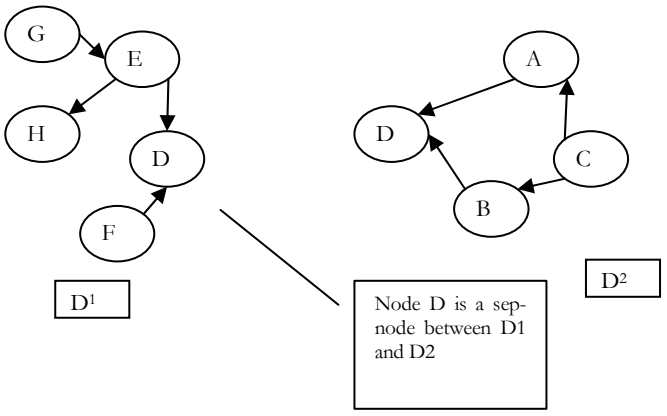


Figure 3 – Third test network

In the third test network, two networks were joined on what would be a sep-node in the unsectioned BBN. The join algorithm resulted in no nodes being copied, since the node was already a valid sep-node.

Performance

Performance for entering inference and propagating probabilities between sound sections of the networks is an important result given the time constraints described earlier. Several different models were created to determine the performance of the MSBN manager. Software was developed to create networks that were "whole" and a series of networks that were sectioned. Additional software was created to join the sections into an MSBN, and evaluate the run-times.

The methodology used to construct these tests was similar. Two driver programs were written. In each, a Netica environment was initialized, and networks were loaded into memory, and compiled [7]. In the case of sub-networks, the networks were passed through Pitkin's algorithm(see *Apply Pitkin's algorithm to identify overlapping cliques*

on pp. 11, and the resulting sound-sections were used in the performance tests. After initialization, an outer loop was used to run the test *m* number of times. Each iteration of this loop served to provide an "average" of run time. Every iteration of the outer loop retrieved a timestamp, in milliseconds, from the system's clock, and then entered into an inner loop. The inner loop ran for *n* times. Each iteration entered a finding into a set node, retrieved the belief of another node, and retracted all findings from the network. Another time stamp was read at the completion of the loops, and the difference was reported.

Three different network configurations were used to do performance testing: a *Simple Overlapping 3-DAG Network, a Simple Fault Analysis Network, and a Deep NAND-Binary Tree.*

Simple Overlapping 3-DAG Network

The network described in figure one, consisting of nodes A, B, C, D, E, and F was combined into a single network. The resulting network was run through a series of performance tests that entered new findings into the network and evaluated the state of a particular node after the probabilities where updated.

In this example, all of the nodes were copied into each of the sub-DAGs. The linkage list included all of the nodes of the networks. There was a very high cost to evaluate the linkage boundaries between the networks in comparison to evaluating the overall network.

This configuration demonstrates a network configuration that yields no localization of probability propagation. Because there is no localization in this model, it represents a base-line for comparison between other models.

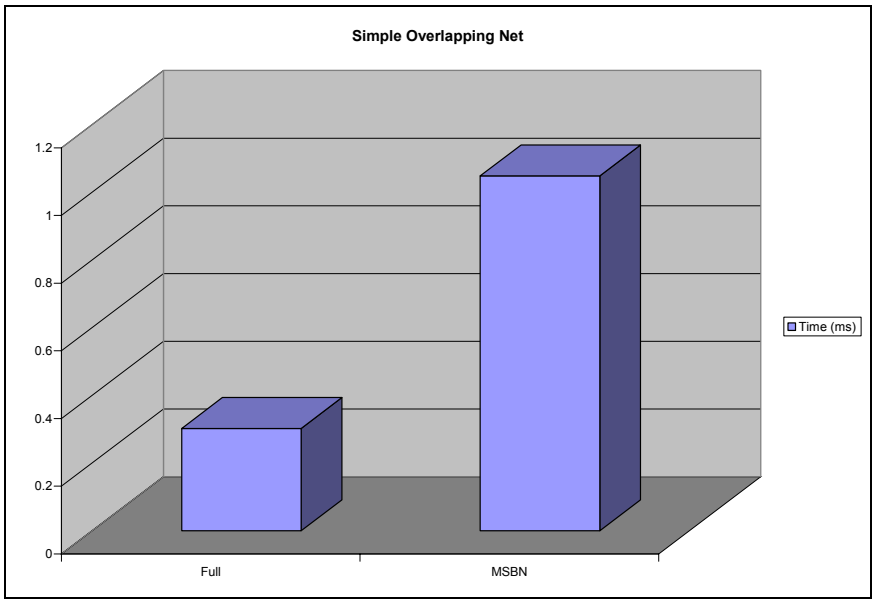


Figure 4 - Runtime Comparison

On average, it took 0.4 ms to enter findings, propagate beliefs, and retrieve results from this simple 6 node network. When the MSBN version was used, propagation had to occur 3 times, and required a little more than 3 times the computation time. The poor performance is attributable to the lack of localization in this trivial model.

Simple Fault Analysis Network

A simple fault analysis model was constructed to demonstrate a network with more localized changes. This network, depicted in Figure 5 - Computer diagnosis network, represents a trivial fault diagnostic network for a computer system. Three "interesting" nodes are: `Tones`, `System`, and `SystemReady`.

`Tones` represent the "beep codes" used by most computer BIOS chips to indicate problems on the system board. One beep indicates the system is functioning, other "beep codes" indicate a failure.

`System` represents the state of the hardware, true for OK, false for failure. In this network, the node "System" was shared between two sub-nets. This was the only node shared between sub-nets. In this case, this node is a sep-node, and no other nodes had to be copied between networks.

`SystemReady` indicates the combined state of the Hardware, Monitor, Disk, and Operating System. This node was the node that was tested in the performance study. As belief about certain conditions in the system was entered, this node's findings changed accordingly.

Following is a pictorial representation of the Network used in testing. The subnets would be divided through the `System` node. Nodes used in the performance testing were rendered showing the state of the conditional probability tables. The remaining nodes are the same type of nodes, but have been rendered to save space.

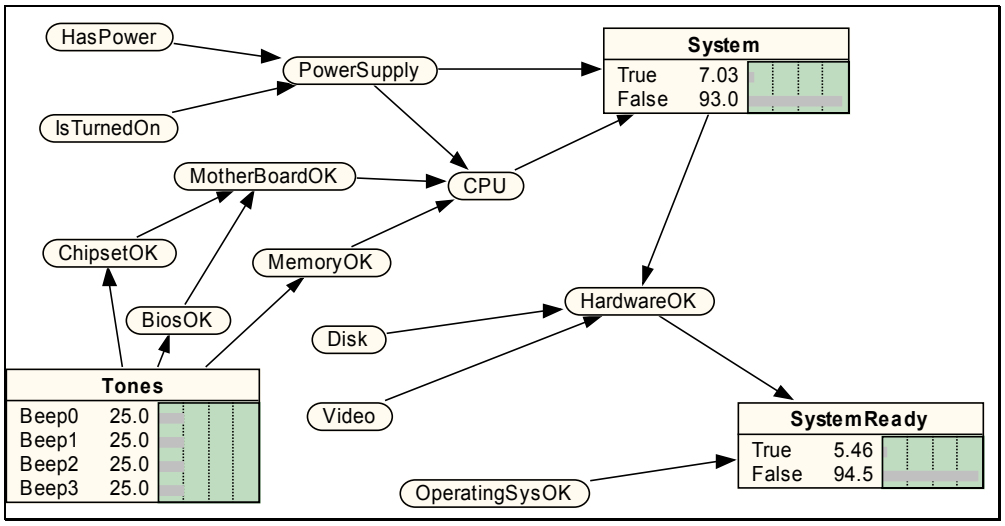


Figure 5 - Computer diagnosis network

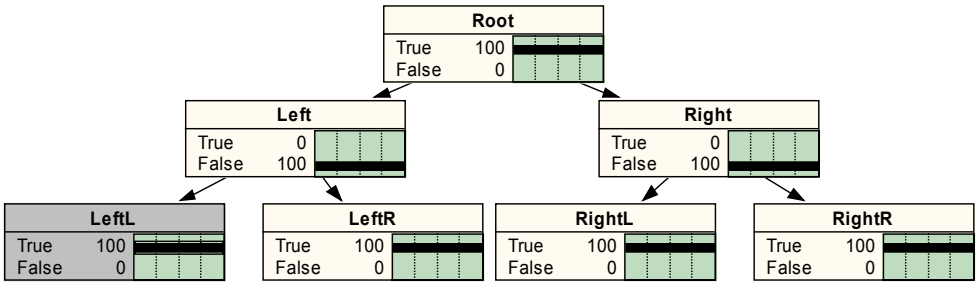
Identical performance testing methodology was used to benchmark the performance of this network, both as a complete network and as an MSBN.

Results of performance testing of this network produced nearly identical results for both utilities. There was no definable performance loss or gain by using either the MSBN Manager or not for this model. Analyzing the results shows that even subtle changes in the belief of the "Tones" node creates large changes in the belief "System", which must then be propagated through to the other network. This network demonstrates another type of localization problem. That is, when the probabilities of the sep-node are highly dependent on either of the models that that node separates.

This test also establishes that the average run-time of the MSBN manager will not be significantly worse than not using it under the following conditions: 1) when the number of nodes on either side of the separator nodes is proportionately higher than the number of nodes that make up the boundary between the networks, and 2) the run-time is proportionate to the amount of time required to evaluate the two networks plus the time to compare the Cartesian product of the number of separator nodes and their parent states.

Deep NAND-Binary Tree

This model took the shape of a binary tree. Parents and children alternate probability, as a series of NAND gates. Following is a sample network that was used, of height 3.



This network was ideal for generating a large number of nodes separated by a proportionately small number of sep-nodes. In this example, the root node was treated as the one and only sep-node. Thus, for each run, there was a left section and a right section of the MSBN.

The network was constructed with the number of nodes ranging from 2^3 (8) through 2^{10} (1024). Performance data is presented below:

Table 1 – Collected performance statistics

2^10 (1024) Size Count	Nodes in tree Run count	Time Ms	Time Ms	Average Full	Average MSBN
100	1	3235	291	32.35	2.91
100	2	3114	290	31.14	2.9
100	3	3165	270	31.65	2.7
100	4	4065	281	40.65	2.81
100	5	3185	280	31.85	2.8
100	6	3135	301	31.35	3.01
100	7	3174	280	31.74	2.8
100	8	3245	260	32.45	2.6
100	9	2954	291	29.54	2.91
100	10	2884	260	28.84	2.6
		3215.6	280.4	32.156	2.804

Performance between the two different implementations was surprising. The complete network required 32ms for each evaluation with 2^{10} nodes, while the MSBN Managed network required only 2.8ms for each evaluation. The speed-up of this process was 11.42 times.

Following are two graphs of the average time required to perform an enter-finding, update belief, and retrieve belief of another node cycle. In Figure 6 - Complete vs. MSBN performance, the performance difference appears as the number of nodes in the network increases. At the worse case, the MSBN performance becomes asymptotic with the performance of the complete network.

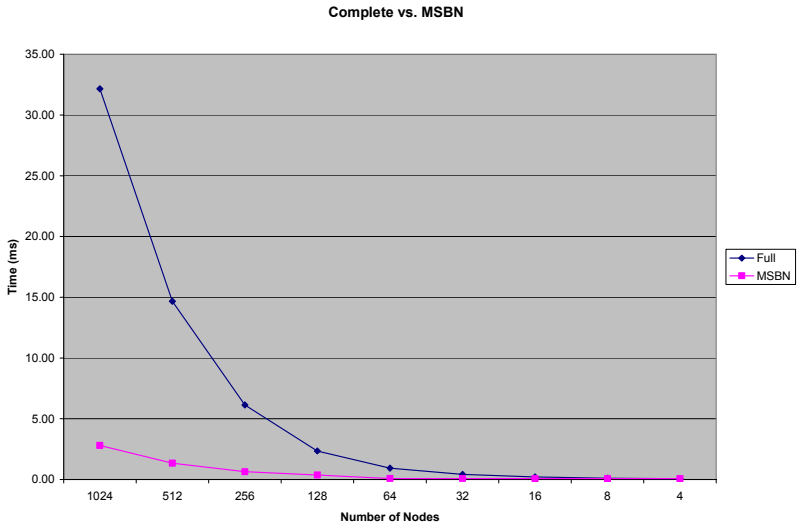


Figure 6 - Complete vs. MSBN performance

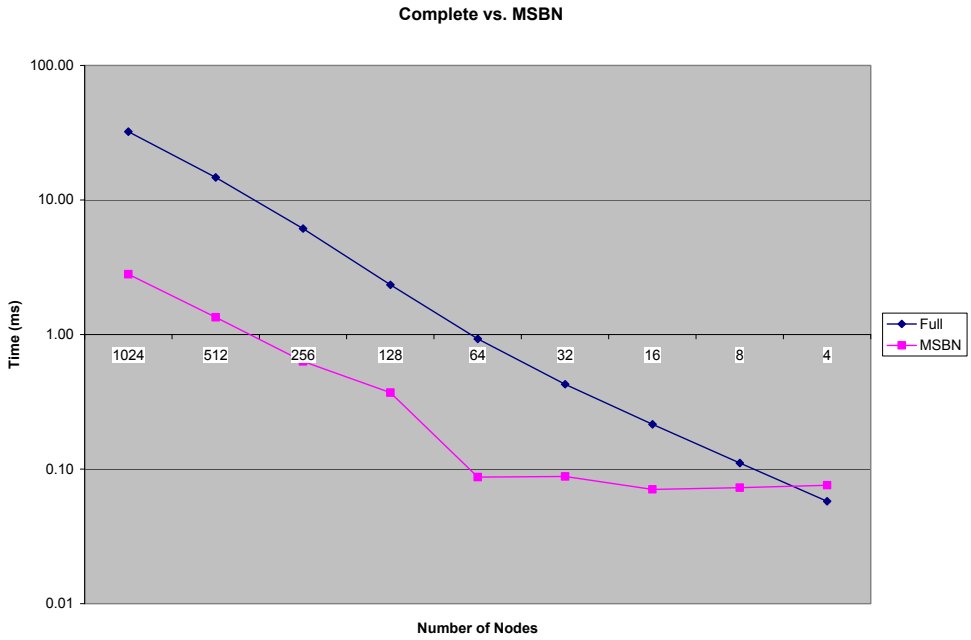


Figure 7 - Logarithmic graph of performance

In Figure 7 - Logarithmic graph of performance, the graph plots the two curves on a logarithmic plot. Notice that the time required to evaluate the full network is log-linear with the number of nodes in the graph. The time required to evaluate the MSBN managed graph is log-linear until the graph reaches 64 nodes, and then the graph flattens out. Presumably, the linear cost of evaluating the sep-nodes begins to out-weigh the gains of not having to propagate the probabilities each time.

Sample Code

Following is sample code that instantiates networks, processes them for sound sectioning, and begins to enter probabilities that may propagate between the nodes of the network.

```
/*
 *NeticaTestFour.java
 *
 * Created on November 10, 2001, 12:14 AM
 */

/**
 *
 * @author  Tom Briggs
 * @version
 */

import MSBN.*;
import norsys.netica.*;

import java.util.*;

public class NeticaTestFive {

    /** Creates new NeticaTestFour */
    public NeticaTestFive() {
    }

    /**
    * @param args the command line arguments
    */

    public static void main (String args[]) {
        LinkedList linkages = new LinkedList( );
        try {
            Environ e = new Environ("");

            Streamer S1 = new Streamer("d:\\rd1.dne");
            Streamer S2 = new Streamer("d:\\rd2.dne");
            Streamer S3 = new Streamer("d:\\rd3.dne");

            norsys.netica.Net D1 = norsys.netica.Net.read(S1, true);
            norsys.netica.Net D2 = norsys.netica.Net.read(S2, true);
            norsys.netica.Net D3 = norsys.netica.Net.read(S3, true);

            linkages.add( new netLinks( "D1", "D2", "A",0.2));
            linkages.add( new netLinks( "D2", "D3", "C",0.2));
            linkages.add( new netLinks( "D1", "D3", "B",0.2));

            MSBNMgr M = new MSBNMgr( "msbn4");

            M.importNetwork(D1);
            M.importNetwork(D2);
            M.importNetwork(D3);

            M.importLinkages( linkages );
            M.processIsolatedDAGs( );


            linkages = M.getLinkages( );

            D1.compile( );
            D2.compile( );
            D3.compile( );

            PropagationWorker p = new PropagationWorker(  );
            p.addNetwork(D1);
            p.addNetwork(D2);
            p.addNetwork(D3);
            p.setInterNetLinks(linkages);

            norsys.netica.Node D1A = D1.getNodeNamed("A");
            p.enterFinding(D1, D1A,"FALSE");
            p.updateMSBN( );

            norsys.netica.Node D2Z = D2.getNodeNamed("Z");
            System.out.println("Z: " + D2Z.getBelief("False"));

            System.out.println("Finished");

        }
        catch (norsys.netica.NeticaException e)
        {
            e.printStackTrace( );
        }

    }

}
```

Figure 8 - Example MSBN Java Code

The code in Figure 8 - Example MSBN Java Code reads three Netica sub-nets from the disk, loads them into the MSBN manager. The statement "processIsolatedDAGs" begins Pitkin's algorithm. This causes the side effects of the sub-networks being modified to be sound sections with each other. The linkages that were added are retrieved from the MSBN manager. The set of objects are then passed to the propagation manager that intercepts Netica's "EnterFindings" routines to manage the propagation of values.

CONCLUSIONS

The scope of the project was to demonstrate that an MSBN management package could be created that would meet the demands of a fault analysis system. Specifically, the model would be required to be able to update the probabilities of large networks in 50ms or less.

A management package was developed using the Java programming language and the Netica-J package. The MSBN Manager accepted as input one or more Netica networks and the linkages between them. The networks were reconfigured to support a sound-sectioning to ensure that probabilities propagated through the sections in the same manner as it would if the networks were not segmented.

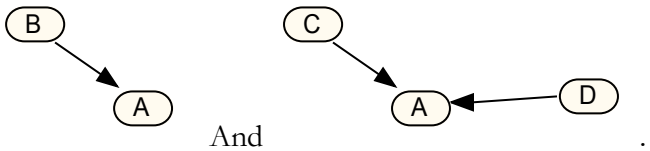
This implementation did address some of the time constraints. In particular, the amount of time required to process new findings was proportionate to the nodes in the networks and the number of nodes that separated them. The processing time was also proven to be relative to the configuration of the network and the amount of localization within the networks.

New networks could be added to the MSBN Manager, but there were no provisions to handle network deletion. Additional research must be conducted to determine a sound method for removing duplicate nodes from sub-networks and the linkage lists while maintaining sound sectioning of the network.

Another area where there needs to be additional research is involved in adding new networks to an existing MSBN. Currently, the entire pool of networks must be re-evaluated. This will add considerable time to the overall reconfiguration times. One suggestion is to identify those nodes, and the networks that contain them, that form the cliques of the joined networks, and only look at those

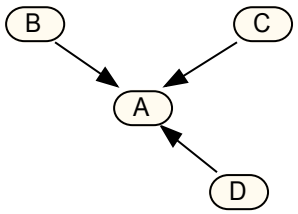
networks during a reconfiguration. The goal of this operation is to make reconfiguration a localized event to reduce the reconfiguration time.

Finally, one other area of additional research needs to be conducted into merging the probability tables of nodes that are copied between networks. For example, consider the following two sub-nets:



And .

Node B would be copied into the second DAG, and C and D would be copied into the first DAG. The resulting node would appear as:



Node A's conditioned probability table would need to be merged to include the three parents of A. This is currently not being done effectively.

Overall, the project was successful at meeting its stated goals. Although this project did not create an environment suitable for diagnosing faults, it did demonstrate that, as a concept, the process has merit. Further, it identified other issues that must be considered for future work.

BIBLIOGRAPHY

[1] Almond, Russell G. *Graphical Belief Modeling*. New York: Chapman & Hall USA, 1995.

[2] Cooper, G.F. "Probabilistic Inference Using Belief Networks is NP-hard." Stanford University Knowledge Systems Laboratory Memo KSL-82-27, May 1987, quoted in Xiang, Poole, Beddoes 1993.

[3] Heckerman, Breese, Rommelse, "Troubleshooting Under Uncertainty" Technical Report MSR-TR-94-07, Microsoft Research Advanced Technology Division, Microsoft Corp., Redmond, January 1994. Database on-line. Available from CiteSeer, http://citeseer.nj.nec.com/ heckerman94troubleshooting.html, (July 27, 2002).

[4] Pearl, Judea. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco: Morgan Kaufmann Publishers, Inc. 1988.

[5] Pitkin, Janet. "Adapting Bayesian Belief Networks for Fault Analysis in Real-Time Systems." Master's thesis, Shippensburg University, August 1998.

[6] Norsys, Inc., Announcement of Official Release of Netica-J, http://www.norsys.com/index.html?news_set=1 (July 27, 2002).

[7] Norys, Inc. "Netica C API Manual" http://www.norsys.com/download_api.html (July 27, 2002).

[8] Norsys, Inc. "Netica-J API Reference Manual", http://www.norsys.com/netica-j/docs/NeticaJ_Man.pdf (July 27, 2002)

[9] Romanycia, Marc, "Netica-J for Windows release," 2 October 2001, personal email, forwarded by Wellington C, 3 October 2001 (July 27, 2002).

[10] Wellington, C. and Burnett, R. "Prototype for Diagnosing Network Failures Using Bayesian Belief Networks." Aug. 1996 (n.p.).

[11] Wellington, Carol. "Feasibility Study for Diagnosing Network Failures Using Bayesian Belief Networks.," Research proposal, Shippensburg University, January 1998.

[12] Xiang, Poole, Beddoes. "Multiply Sectioned Bayesian Networks and Junction Forests for Large Knowledge-based Systems." *Computational Intelligence* 9, no. 2 (1993): 171-220.