APPROXIMATE INFERENCE OF BAYESIAN NETWORKS THROUGH EDGE DELETION

by

JULIE ANN THORNTON

B.S., Kansas State University, 2003

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2005

Approved by:

Major Professor
William Hsu

ABSTRACT

Bayesian networks are graphical models whose nodes represent random variables and whose edges represent conditional dependence between variables. Each node in a Bayesian network is equipped with a conditional probability function that expresses the likelihood that the node will take on different values given the values of its parents. A common task for a Bayesian network is to perform *inference* by computing the marginal probabilities of each possible value for each node. In this thesis, I introduce three new algorithms for approximate inference of Bayesian networks that use edge deletion techniques.

The first reduces a network to its maximal weight spanning tree using the Kullback-Leibler information divergence as edge weights, and then runs Pearl's algorithm on the resulting tree. Because Pearl's algorithm can perform inference on a tree in linear time, as opposed to the exponential running time of all general exact inference algorithms, this reduction results in a tremendous speedup in inference.

The second algorithm applies triangulation pre-processing rules that are guaranteed to be optimal if the original graph has a treewidth of four or less, and then deletes edges from the network and continues applying rules so that the resulting triangulated graph will have a maximum clique size of no more than five. The junction tree exact inference algorithm can then be run on the reduced triangulated graph. While the junction tree algorithm has an exponential worst-case running time in the size of the maximum clique in the triangulated graph, placing a bound on the clique size effectively places a polynomial time bound on the inference procedure.

The third algorithm deletes edges from a triangulation of the original network until the maximum clique size in the triangulated graph is below a desired bound. Again, the junction tree algorithm can then be run on the resulting triangulated graph, and the bound on the maximum clique size will also polynomially bound the inference time.

When tested for efficiency and accuracy on common Bayesian networks, these three algorithms perform up to 10,000 times faster than current exact and approximate techniques while achieving error values close to those of sampling techniques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Approximate Inference of Bayesian Networks through Edge Deletion

CHAPTER ONE: Introduction

Bayesian belief networks in artificial intelligence are powerful tools to simulate and approximate real situations. For example, the Pathfinder Bayesian network is used in a commercial product called Intellipath to diagnose lymph-node pathology. Bayesian networks are directed acyclic graphs whose nodes represent a certain state or event in the situation being modeled, such as the symptoms of a patient. Dependencies between nodes, or states, are represented by edges, and each node has a conditional probability function that represents the probability that the node will take a certain value (for example, true or false for a binary node representing whether an event happens) given the values of the nodes on which it is dependent. Bayesian networks can help predict outcomes in the situations they model when the values of certain nodes are given, for example, if a node in Pathfinder corresponds to HIV-status and it is known that the patient is HIV-positive. Such nodes that can be measured are called the evidence for the

network. Many exact and approximation algorithms exist to infer the states of the remaining nodes, hence simulating the real situation.

Unfortunately, both exact inference [Cooper90] of Bayesian networks and approximate inference [DagumLuby93] within a given error bound are *NP*-hard for general networks, so finding fast approximate inference algorithms that perform well in practice is very important. While there are several exact inference algorithms that perform fairly quickly on some networks, these algorithms have an exponential worst-case running times.

While all existing exact inference algorithms for general Bayesian networks do have exponential running times, inference can be performed in linear time on a polytree using Pearl's algorithm [Pearl88]. Thus if a network could be approximated by a polytree, inference could be performed very quickly. Furthermore, while the junction tree algorithm is one of the most popular algorithms for exact inference of Bayesian networks, its running time is exponential in the size of the biggest clique in the triangulated graph. However, if the cliques in the triangulated graph could be guaranteed to be within a certain bound, then the running time of the junction tree algorithm would be polynomially bounded.

I propose three new algorithms for approximate inference of Bayesian networks that use edge deletion techniques to reduce the complexity of the network and enforce structural constraints on the reduced network that guarantee a polynomial time bound on the inference step. While the technique of edge deletion to reduce the complexity of a Bayesian network before performing inference is not a new one, no research has been

directed towards using edge deletion to constrain the structure of a network in order to enforce a polynomial time bound on inference.

The first of these algorithms, called “reduction to polytree”, computes the maximum weight spanning tree of a Bayesian network using Kullback-Leibler (KL) information divergence as a weight for each edge. Because KL divergence computes the distance between two probability distributions (in this case, between the original distribution and the approximate distribution that results from removing an edge), the resulting spanning tree is the polytree that is a good approximation of the original network. After reducing a network to a polytree, Pearl’s algorithm can be used to perform inference in linear time.

The second algorithm, called “bounding clique sizes with pre-processing”, does not reduce a network all the way down to a polytree structure, but instead deletes edges to guarantee that the maximum clique size in the triangulated graph is less than or equal to five. This algorithm first employs a set of pre-processing rules for triangulating Bayesian networks [Bodlaender01] that remove vertices (and sometimes add edges) in order to determine a perfect elimination scheme for the network. If the treewidth of the original graph is no more than four, these rules are guaranteed to reduce the original graph to the empty graph – and, consequently, learn the complete perfect elimination scheme for the network. After applying these rules, I delete edges from the original network (and the corresponding edges in the reduced graph) in order to continue applying the pre-processing rules. Eventually, the original graph will be reduced to the empty graph and a perfect elimination scheme for the entire network will be learned. Consequently, I can run the junction tree algorithm on the fill-in triangulation of the perfect elimination

scheme and be guaranteed a polynomial time bound on the inference, since the maximum clique size of the triangulation will be bounded by five.

The final new algorithm, called “general bounding clique sizes”, also deletes edges in order to place a bound on the maximum clique size of the triangulated graph. This algorithm is given an initial perfect elimination scheme for the network as well as a desired bound on the maximum clique. It then removes edges from the original network until the fill-in triangulation of the given perfect elimination scheme is guaranteed to not have a clique that is bigger than the desired bound. Just as for “bounding clique sizes with pre-processing”, the junction tree algorithm can then be run on the resulting triangulated graph, and again the bound on the clique sizes will force a polynomial time bound on the inference stage.

In later chapters, I will provide implementation details for these three new approximate inference algorithms, as well as results for how their accuracy and efficiency compare to existing exact and approximate inference techniques. In this chapter, however, I will describe the necessary elements of graph theory and information theory, as well as a formal description of Bayesian networks and common inference algorithms, that are needed as background information.

1.1 Bayesian Networks

In this section, I will first give a general background on Bayesian networks. Specifically, I will describe what they model and when they are used. Next, I will give a more formal definition of a Bayesian network and also introduce other common definitions and theorems associated with Bayesian networks. I will close this section

with an example, which will explain why the model is useful and what sorts of things we can learn from it.

1.1.1 General Description and Uses

As was mentioned above, a Bayesian network is essentially a directed acyclic graph where each node represents a random variable, together with a set of conditional probability functions. If two nodes *u* and *v* are connected by a directed edge, then *v* is conditionally dependent on *u*. If two other nodes are not connected by a directed edge, then they are said to be conditionally independent.

Each node also has an associated conditional probability function that describes the probability that the node will take on its different values (for example, true or false for a binary random variable) given the values of its parents. Bayesian networks can, in general, have nodes that represent continuous random variables. However, any specific example of a Bayesian network in this document will be assumed to have nodes that represent discrete random variables. In this case, we refer to the associated conditional probability function as a conditional probability table.

Bayesian networks are used in a variety of settings, and can be useful in any situation that requires making predictions based on a known model of related variables. Specifically, Bayesian networks are commonly used in medical diagnostic software, spam filtering, and in targeted marketing on the Internet.

1.1.2 Definitions and Theorems

In this section, I give several definitions and theorems pertaining to Bayesian networks, including a formal definition of a Bayesian network itself. The terms and theorem below

will be referenced throughout the remainder of the paper. Many of these definitions are taken from [Neapolitan04].

Theorem 1.1. (*Bayes' Theorem).* Given two events E and F such that $P(E) \neq 0$ and $P(F) \neq 0$, we have that

$$P(E|F) = \frac{P(F|E)P(E)}{P(F)}.$$

Furthermore, given n mutually exclusive and exhaustive events $E_1, E_2, \ldots, E_n$ such that $P(E_i) \neq 0$ for all i, we have for $1 \leq i \leq n$,

$$P(E_i|F) = \frac{P(F|E_i)P(E_i)}{P(F|E_1)P(E_1)+P(F|E_2)P(E_2)+\ldots+P(F|E_n)P(E_n)}.$$

Definition. Let X and Y be random variables with possible values x and y, respectively. Then the *joint probability distribution* of X and Y is $P(X = x, Y = y)$.

Definition. Let X and Y be discrete random variables with possible values x and y. Suppose we have a joint probability distribution $P(X = x, Y = y)$. Then

$$P(X = x) = \sum_y P(X = x, Y = y),$$

and $P(X = x)$ is called the *marginal probability distribution* of X.

Definition. Let A, B, and C be sets of random variables defined on the same probability space. Then sets A and B are said to be *conditionally independent* given the set C if, for all values of the variables in the sets a, b, and c, whenever $P(c) \neq 0$, the events $A = a$ and

$B = b$ are conditionally independent given the event $C = c$. That is, either $P(a|c) = 0$, or $P(b|c) = 0$, or $P(a|b,c) = P(a|c)$.

Definition. Suppose we have a joint probability distribution P of the random variables in some set V and a directed acyclic graph $G = (V, E)$. We say that (G, P) satisfies the *Markov condition* if for each variable $X \in V$, $\{X\}$ is conditionally independent of the set of all its nondescendents given the set of all its parents.

Theorem 1.2. Let a directed acyclic graph G be given in which each node is a random variable, and let a discrete conditional probability distribution of each node given values of the parents in G be specified. Then the product of these conditional distributions yields a joint probability distribution P of the variables, and (G, P) satisfies the Markov condition.

Definition. Let P be a joint probability distribution of the random variables in some set V, and let $G = (V, E)$ be a directed acyclic graph. We call (G, P) a *Bayesian network* if (G, P) satisfies the Markov condition. Furthermore, if we specify a directed acyclic graph G and any discrete conditional distributions, we obtain a Bayesian network. (This is the common way for defining a Bayesian network.)

Definition. Let $B = (G, P)$ be a Bayesian network. We call B a *polytree* if and only if G has no undirected cycles.

Definition. Let (G, P) be a Bayesian network, and let $X \in V$. We define $Pa(X) = \{Y | (Y, X) \in A\}$ to be the parent set of X.

Theorem 1.3. Let (G, P) be a Bayesian network, and suppose $V = \{X_1, X_2, \ldots, X_n\}$. Then we have that the *joint probability* of V is

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(x_i | Pa(x_i)).$$

This property is called the *chain rule* for random variables.

Definition. Let $B=(G=(V, E), P)$ be a Bayesian network, and let $\mathbf{e} \subseteq V$ be a set of evidence nodes. Then the *importance function* for B given **e** is a new probability distribution P' such that for each node $X \in V$ with possible value x_i, $P(X=x_i|\mathbf{e}) = P'(X=x_i)$.

1.1.3 Example

Figure 1 contains an example of a simple Bayesian network. This network has four nodes – Cloudy, Sprinkler, Rain, and WetGrass – each of which are binary random variables. Furthermore, it can be seen by the network that the value of Sprinkler is conditionally dependent on the value of Cloudy, that the value of Rain is conditionally dependent on the value of Cloudy, and that the value of WetGrass is conditionally dependent on the values of Sprinkler and Rain. All other variables are conditionally independent. Finally, the tables of probabilities next to each node in the network represent the conditional distributions of each node. (Since each variable is discrete, the functions can be represented as tables.)

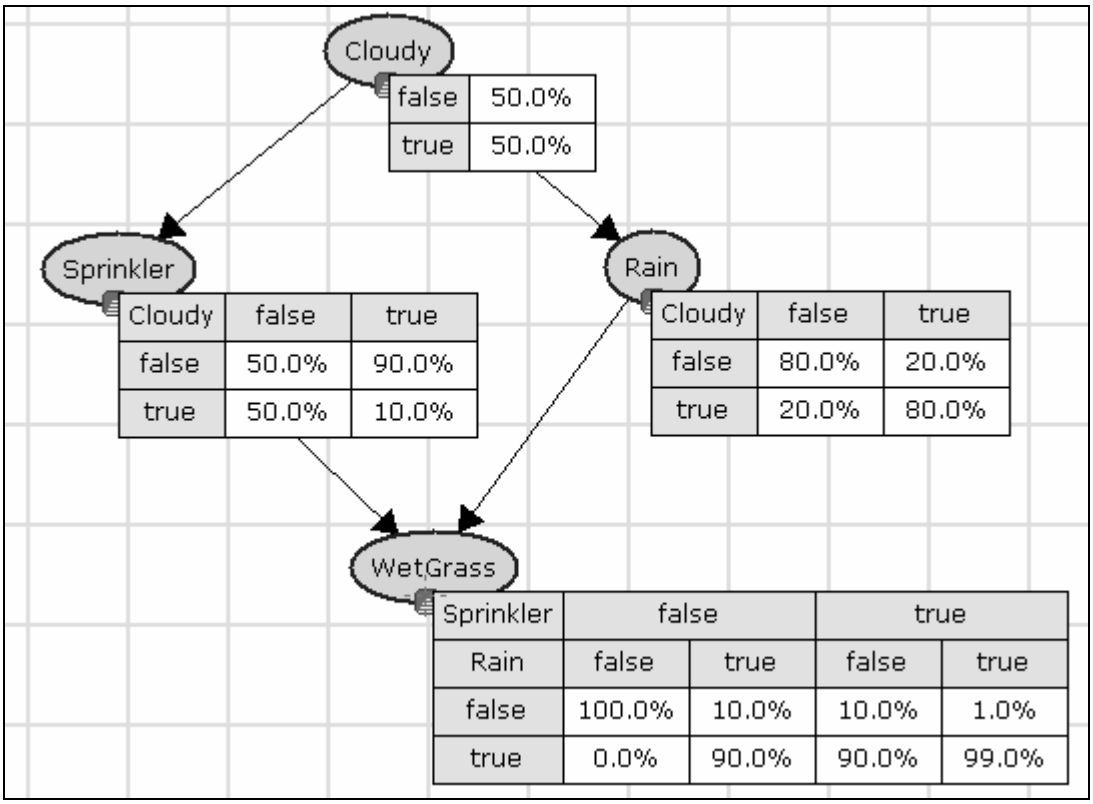

Figure 1. The Sprinkler-Rain Bayesian network

Suppose we wanted to calculate the probability that it is cloudy, the sprinkler is off, it is raining, and the grass is dry. We would need to apply the chain rule for random variables, and would make the calculation:

$$P(\text{Cloudy} = \text{T}, \text{Sprinkler} = \text{F}, \text{Rain} = \text{T}, \text{WetGrass} = \text{F})$$

$$= P(\text{Cloudy} = \text{T}) * P(\text{Sprinkler} = \text{F} | \text{Cloudy} = \text{T}) * P(\text{Rain} = \text{T} | \text{Cloudy} = \text{T})$$

$$* P(\text{WetGrass} = \text{F} | \text{Rain} = \text{T}, \text{Sprinkler} = \text{F})$$

We can find these probabilities by consulting the conditional probability tables for each node in the network. When we do this, we get that the probability of the above situation is 0.5*0.9*0.8*0.1 = 0.036, or 3.6%.

1.2 Bayesian Inference

One of the most common things to do with a Bayesian network is to perform *inference*, which computes the marginal probability $P(V = v)$ for each node V and each possible instantiation v. What inference does is give us an idea of how likely cases for a specific random variable are, using the information in the Bayesian network. Inference can also be done on a Bayesian network when we know the values of some nodes (as *evidence*) and wish to compute the likelihood of values of other nodes. This is called computing *posterior probabilities*, because we are trying to find $P(V = v|\mathbf{e})$ for each node V and each possible instantiation v, given the evidence value **e**.

There are two types of inference on Bayesian networks: exact and approximate. As the name suggests, exact inference algorithms compute the exact values of each marginal or posterior probability, while approximate inference algorithms sacrifice some accuracy of the probabilities to report results quickly. The following two sections will discuss common exact and approximate inference algorithms in more detail.

1.2.1 Exact Inference

The goal of an exact inference algorithm is to report the exact values for either the marginal ($P(V = v)$) or posterior probabilities ($P(V = v|\mathbf{e})$) for each instantiation v of each node V, possible given some evidence **e** of other node values. Below, I will discuss two common exact inference algorithms – Pearl's algorithm and the Lauritzen-Spiegelhalter algorithm. A walkthrough of both algorithms appears in the appendix. I will also provide some insight into the complexity of exact inference on general Bayesian networks.

Pearl's Algorithm.

Pearl's algorithm is a linear-time algorithm that computes the posterior probabilities of each node given evidence of singly-connected networks. The algorithm itself is fairly complicated, but the general idea is to separate the evidence for each node into the evidence "above" the node and the evidence "below" the node. This is only possible in a singly-connected network. Pearl introduced the notation $\lambda(X = x)$ for the *diagnostic support* of a node X with value x, which is the probability of evidence below X given that $X = x$. He also used the notation $\pi(X = x)$ for the *causal support* of a node X with value x, which is the probability that $X = x$ given the evidence above X. Using these values, we have that $P(X = x|\mathbf{e}) = \alpha^* \lambda(X = x)^* \pi(X = x)$, where α is a normalizing constant to ensure that the necessary probabilities add to one [Pearl88].

To calculate these λ and π values, Pearl introduced λ and π messages. The λ messages are passed from child to parent and give the probability of the evidence in the child's subtree given that the parent X has value x. π messages are passed from parent to child and give the probability that the parent X has value x given the evidence in the parent's subtree [Pearl88].

The λ and π messages and λ and π values are calculated as follows:

<u>λ message:</u> If B is a child of A, B has k possible values, A has m possible values, and B has other parents $D_1, D_2 \ldots, D_r$, each with n_i possible values $(d_{i1}, \ldots, d_{ini})$, then for $1 \leq j \leq m$, the λ message from B to A is

$$\lambda_B(A=a_j)=\sum_{p_1=1}^{n_1}\sum_{p_2=1}^{n_2}\ldots\sum_{p_r=1}^{n_r}\pi_B(D_1=d_{1p_1})\times\pi_B(D_2=d_{2p_2})\times\ldots\times\pi_B(D_r=d_{rp_r})$$
$$\times(\sum_{i=1}^{k}P(B=b_i|A=a_j,D_1=d_{1p_1},\ldots,D_r=d_{rp_r})\times\lambda(B=b_i))$$

π message: If B is a child of A and A has m possible values, then for $1 \leq j \leq m$, the π message from A to B is given by

$$\pi_B(A=a_j)=\begin{cases}1 & \text{if } A \text{ is instantiated to } a_j,\\ 0 & \text{if } A \text{ is instantiated but not to } a_j,\\ \frac{P'(A=a_j)}{\lambda_B(A=a_j)} & \text{if } A \text{ is not instantiated.}\end{cases}$$

π value: If B has k possible values and parents A_1, A_2, …, A_m, each with n_i possible values, then for $1 \leq j \leq k$, the π value of B is given by

$$\pi(B=b_j)=\sum_{i_1=1}^{n_1}\sum_{i_2=1}^{n_2}\ldots\sum_{i_m=1}^{n_m}P(B=b_j|A_1=a_{i_1},\ldots,A_m=a_{i_m})\pi_B(A_1=a_{i_1})$$
$$\times\ldots\times\pi_B(A_m=a_{i_m})$$

λ value: If B has k possible values and $S(B)$ is the set of B's children, then for $1 \leq i \leq k$, the λ value of B is

$$\lambda(B=b_i)=\begin{cases}\prod_{c\in S(B)}\lambda_c(B=b_i) & \text{if } B \text{ is not instantiated,}\\ 1 & \text{if } B \text{ is instantiated to } b_i,\\ 0 & \text{if } B \text{ is instantiated but not to } b_i.\end{cases}$$

Posterior probability: If B has k possible values, then for $1 \leq i \leq k$, $P'(b_i)$, the conditional probability of b_i based on the variables thus far instantiated (the evidence), is given by

$$P'(B = b_i) = \alpha^* \lambda(B = b_i)^* \pi(B = b_i)$$

Using these calculations, Pearl then described the following algorithm to compute the posterior probabilities of each node. (The algorithm below is taken from [Neapolitan04].)

Initialization:

1) Set all λ values, λ messages, and π messages to 1

2) For all roots A, if A has m possible values, then for $1 \leq j \leq m$, set

$$\pi(A = a_j) = P(a_j)$$

3) For all roots A and for all children B of A, do

Post a new π message to B (this will start a propagation flow)

Updating Rules:

1) If a variable B is instantiated to b_j, then

- Set $P'(B = b_j) = 1$, and for $i \neq j$, set $P'(B = b_i) = 0$

- Compute $\lambda(B)$

- Post λ messages to B's parents

- Post π messages to B's children

2) If B receives a λ message from a child A and is not instantiated

- Compute $\lambda(B)$

- Compute $P'(B)$

- Post λ messages to B's parents

- Post π messages to B's other children

3) If B receives a π message from a parent A

a) If B is not instantiated

- Compute $\pi(B)$

- Compute $P'(B)$

- Post π messages to B's children

b) If $\lambda(B) \neq (1, 1, \ldots, 1)$

- Post λ messages to B's other parents

Exact Inference Complexity.

While Pearl's algorithm performs exact inference in linear time on Bayesian networks that are polytrees, the same running time is not possible for Bayesian networks in the general case. Exact inference has been shown to be *NP*-hard for Bayesian networks, using a reduction from *3-SAT* [Cooper90]. Because of this, all exact inference algorithms that work on general Bayesian networks must have an exponential running time in the worst-case, unless $P = NP$. However, there are still several general exact inference algorithms that perform quite well in practice.

Lauritzen-Spiegelhalter (LS) Algorithm.

The Lauritzen-Spiegelhalter (LS) algorithm [LauritzenSpiegelhalter88] is an exact inference algorithm for Bayesian networks that works on all models. A summary of the algorithm is given below:

1) Moralize the network by adding edges between common parents and removing the directionality of the edges. The purpose of the moral edges is to ensure that the resulting undirected graph does not contain any conditional independence relations that are not expressed in the original network.
2) Triangulate the graph by adding edges so that there are no induced subgraphs that are simple cycles of length four or greater
3) Construct a tree of cliques from the triangulated graph
4) Initialize clique potentials using the conditional probabilities in each clique's nodes
5) Propagate diagnostic and causal support messages through the clique-tree using as in Pearl's algorithm. These messages will update the clique potentials.
6) Marginalize the final clique potentials to determine the posterior probabilities for each node

Section 1.3.3 contains a detailed description of how to construct a tree of cliques from the triangulated graph. An example of building the clique tree for a Bayesian network also appears in the appendix.

The LS algorithm performs very well on average, but its running time is exponential in the size of the biggest clique in the clique tree. Thus if the selected triangulation algorithm adds many more edges than were necessary, LS will be very slow. However, the problem of finding a triangulation for a graph that minimizes the maximum clique size in the clique tree is also *NP*-hard [Arnborg87].

1.2.2 Approximate Inference

Because all exact inference algorithms for Bayesian networks must have an exponential worst-case running time unless *P = NP*, there has been much research devoted to finding efficient approximate inference algorithms. By design, approximate inference algorithms sacrifice accuracy in order to deliver estimated posterior probabilities sooner – in some cases, these inference algorithms can deliver anytime results. Below, I will discuss two of the most common types of approximate inference algorithms – sampling-based and edge deletion techniques. I will also discuss the complexity of using an approximate inference algorithm to achieve a specified error bound for the posterior probabilities.

Sampling.

Sampling algorithms are the simplest and most widely used approximate inference algorithms. While there are many sampling variants, the underlying idea is to randomly instantiate each node in the Bayesian network in topological order to produce a single "sample". Samples are gathered until a specified number has been reached or a certain amount of time has passed, and then posterior probabilities for each node are computed by looking at the frequency of each possible instantiation of each node in all of the samples. Below, I give a brief summary of the two most common sampling

algorithms – forward sampling and likelihood weighting. I also describe the adaptive importance sampling algorithm, which is the most accurate sampling algorithm to date.

Forward Sampling:

Forward sampling is one of the first and simplest approaches to sampling. In this algorithm, we start with a topological ordering of the nodes, values for desired evidence variables, and a maximum number of samples *n*. (This algorithm can easily be modified to use a maximum time bound instead of a maximum number of samples.) Then, the following steps are repeated *n* times:

1) Randomly instantiate each non-evidence variable based on the values of its parents and the conditional probability function for that variable (the order of instantiation is determined by the topological ordering of the nodes)
2) Instantiate each evidence node to the desired value
3) Store the current instantiation of all nodes as a new sample

Once *n* samples have been gathered, the posterior probability $P(X = x)$ is computed for each node X and each possible value x by summing the number of samples in which X is instantiated to x and dividing by *n*. [Henrion88]

Likelihood Weighting:

Likelihood weighting is similar to forward sampling, except each sample is also given a weight. Suppose a Bayesian network has evidence nodes $E_1, E_2, \ldots, E_k$, which are instantiated to the values $e_1, e_2, \ldots, e_k$, respectively. Then the weight of a sample *s* would be $P(E_1 = e_1 \mid Pa(e_1)) * P(E_2 = e_2 \mid Pa(e_2)) * \ldots * P(E_k = e_k \mid Pa(e_k))$. In other words, the weight would be the product of the probability that each evidence node has its desired value given the value of its parents in the sample *s*. Once *n* samples have been

gathered and weighted, the posterior probability $P(X = x)$ is computed for each node X and each possible value x by summing the weight of the samples in which X is instantiated to x and dividing by the total weight of all n samples. [10, 17]

For example, consider again the Sprinkler-Rain Bayesian network in Figure 1. Suppose we have as evidence that WetGrass is true and that Cloudy is false. Suppose further that we randomly select the value of Sprinkler to be true and the value of Rainy to be false. Then the weight for the sample {Cloudy = false, Rainy = false, Sprinkler = true, WetGrass = true} is P(Cloudy = false) * P(WetGrass = true | Rainy = false, Sprinkler = true}, which is 0.5*0.9 = 0.45.

Adaptive Importance Sampling:

Adaptive importance sampling (AIS) is not as widely used as forward sampling or likelihood weighting, but it is considered by many to be the most accurate sampling algorithm – particularly for Bayesian networks with unlikely evidence values. AIS uses initialization heuristics that help unlikely values be picked more and that resets the conditional probability of the parents of unlikely evidence nodes to a uniform distribution. AIS also introduces the notion of an *importance function*, which is the ideal conditional probability table of each node given the evidence values. Because the importance function for some nodes in a network with unlikely evidence nodes is often very different than the original conditional probabilities, the initialization heuristics help start a search for the importance function away from a local optimum.

AIS further updates the importance function several times throughout the sampling process, by considering the most recent batch of samples, the old importance

function, and the current rate that the importance function is being learned. Further, the weight for a sample *s* is given by

$$\omega^k \times \frac{Pr(s)}{Pr^k(\mathbf{e}|s)}$$

where $Pr(s)$ is the joint probability of *s* using the original conditional probability functions, $Pr^k(\mathbf{e} \mid s)$ is the likelihood weight of *s* using the current importance functions, and ω^k is a monotonically increasing weight function. Although it is more complicated, AIS still follows the same steps as likelihood weighting -- a batch of samples are generated and weighted, and then the posterior probabilities of each node are calculated using the frequency of each node value in the sample and the weight of the samples. [ChengDruzdzel00]

Edge Deletion.

While sampling is the most common form of approximate inference, many other methods have been developed. One technique that has been used is to reduce the complexity of the Bayesian network by deleting some of its edges, and then applying some exact inference algorithm to the resulting network. Unlike with sampling, an error bound on the resulting approximate posterior probabilities can often be given based on which edges are deleted. There are two edge deletion techniques that have been created for approximate inference: one by Kjaerulff in [Kjaerulff93], and the second by van Engelen in [vanEngelen96]. Kjaerulff chooses edges for deletion from the triangulated graph of a network, while van Engelen chooses edges from the original network. Both these techniques will be discussed in more detail in Chapter 2.

Approximate Inference Complexity.

While approximate inference algorithms are an excellent way to obtain fairly accurate estimates of posterior probabilities very quickly, they have their limitations. For several years after the exact inference problem was proved to be *NP*-hard, many researchers thought that approximate inference could be done in polynomial time. However, it was soon proved that the problem of approximating posterior probabilities to within a specified relative error is also *NP*-hard [DagumLuby93]. However, there is still much room for research on approximate inference algorithms that produce fairly accurate results quickly in most Bayesian networks.

1.3 Graph Theory

In Section 1.2, I dismissed the process of triangulation as simply adding chords to cycles until there was no induced subgraph that was a simple cycle of length four or greater. This section contains a much more detailed background on triangulation, and discusses two techniques for triangulating a graph – maximum cardinality search and using pre-processing rules. Finally, I will also provide more insight on how to construct a clique-tree from a triangulated graph in an exact inference algorithm and discuss the complexity of finding an optimal triangulation.

1.3.1 Definitions and Theorems

Below, I give several definitions and a theorem pertaining to graph triangulation. The terms and theorem below will be referenced throughout the remainder of the paper. Also, note that the definition $G = (N, A)$ refers to a directed graph, while the definition $G = (V, E)$ refers to an undirected graph. Furthermore, the edge notation $\{u, v\}$ refers to an

undirected edge, while the arc notation (u, v) refers to a directed edge with source u and sink v.

Definition. Let $G = (V, E)$ be an undirected graph. A set of nodes $S \subseteq E$ is a *clique* if the following conditions hold.

1) S is a fully connected subgraph: for each pair of nodes $u, v \in S$, $\{u, v\} \in E$

2) S is maximal: for all $x \in V$, $x \notin S$, $\exists\, y \in S$ such that $\{x, y\} \notin E$

Definition. Let $G = (N, A)$ be a directed graph. Then $G' = (V, E)$ is the *moralized graph* of G if $V = N$ and $E = \{\{x, y\} : (x, y) \in A)\} \cup \{\{u, v\} : u, v, w \in N \wedge (u, w) \in A \wedge (v, w) \in A\}$.

Definition. Let $G = (V, E)$ be an undirected graph, and let f: $V \leftrightarrow \{1, 2, \ldots, |V|\}$ be a one-to-one correspondence. Then f is a *perfect elimination scheme* for G if for each $v \in V$, for all x and y such that $\{v, x\} \in E$ and $\{v, y\} \in E$, if $f(y) > f(v)$ and $f(x) > f(v)$, then $\{x, y\} \in E$. Thus for each vertex, its higher-ordered neighbors form a clique.

Definition. Let $G = (V, E)$ be an undirected graph. G is *triangulated* if it does not contain an induced subgraph that is a simple cycle of length greater than or equal to four.

Theorem 1.4. A graph is triangulated if and only if it has a perfect elimination scheme.

Definition. Let $G = (V, E)$ be an undirected graph, and let f: $V \leftrightarrow \{1, 2, \ldots, |V|\}$ be a one-to-one correspondence. We can construct the *fill-in* triangulation of G given f by turning the set of higher-ordered neighbors of each node into a clique.

Definition. Let $G = (V, E)$ be an undirected graph, and let f be a perfect elimination scheme for G. We say that the fill-in triangulation of G given f is an *optimal triangulation* if the size of the biggest clique in the fill-in triangulation is less than or equal to the size of the biggest clique in any other fill-in triangulation.

Definition. Let $G = (V, E)$ be an undirected graph. The *treewidth* of G is the size of the biggest clique in the optimal triangulation of G minus one.

Definition. Let $G = (V, E)$ be an undirected graph. A vertex $v \in V$ is *simplicial* if all vertices adjacent to v form a clique.

1.3.2 Common Triangulation Techniques

In this section, I will discuss two common triangulation algorithms – maximum cardinality search and triangulation by pre-processing rules. I will also provide a walkthrough on the Asia Bayesian network for each algorithm. When triangulating a Bayesian network $B = (G = (N, A), P)$, G is first moralized to yield an undirected graph G'. The triangulation of the Bayesian network B is the triangulation of G'.

Maximum Cardinality Search (MCS).

Maximum cardinality search is a linear-time algorithm that takes as input a moralized, undirected graph G and outputs a perfect elimination scheme f for G [TarjanYannakakis84]. After the perfect elimination scheme has been constructed, the fill-in of G given f can be constructed, yielding the triangulation of G. MCS computes a weight $w(v)$ for each vertex $v \in V$ to help calculate the perfect elimination scheme. Below is a description of the MCS algorithm. A walkthrough of MCS on the Asia Bayesian network appears in the appendix.

MCS Algorithm:

MaximumCardinalitySearch($G = (V, E)$)

for all vertices $v \in V$, $w(v) \leftarrow 0$

for $i = |V|$ downto 1

choose $u \in V$ such that $w(u) \geq w(x)\ \forall\ x \in V$

$f(u) \leftarrow i$

for all $y \in V$ s.t. {y, u}$\in E$ and $f(y)$ is undefined

$w(y) \leftarrow w(y) + 1$

Essentially, the MCS algorithm picks the last vertex for the perfect elimination scheme, and then increases the weight of all adjacent vertices. Then, until all vertices have been "numbered" in *f*, MCS picks the vertex with the greatest weight, places it in the next-highest spot in *f*, and increases the weight of all adjacent vertices. The motivation for this algorithm is that vertices with many neighbors will tend to have a high ordering in *f*, and vertices with few neighbors will tend to have a low ordering. Thus, when the fill-in of *G* given *f* is constructed, and an edge must be added between all higher-ordered neighbors of each vertex, the vertices with high degree will have few higher-ordered neighbors, so fewer edges will have to be added than would for some arbitrary construction of *f*.

Pre-Processing Rules.

Another triangulation technique is to apply a set of pre-processing rules to a graph, thereby coming up with a partial elimination scheme for the graph. If the pre-processing rules provide an order for each vertex, then the resulting fill-in triangulation is

guaranteed to be optimal. Otherwise, another triangulation algorithm must be applied to the induced subgraph containing the unordered vertices to complete the perfect elimination scheme. This technique, proposed by Bodlaender in [Bodlaender01], is guaranteed to produce an optimal triangulation for any graph with treewidth less than or equal to four. The technique, along with the specific pre-processing rules, will be discussed in more detail in Chapter 2.

1.3.3 Constructing a Clique-Tree from a Triangulated Graph

Triangulation algorithms are often used in conjunction with Bayesian inference. For example, in the LS exact inference algorithm, the moralized graph is triangulated, and then a clique-tree is constructed from the triangulated graph. Message-passing is then performed on the clique-tree to determine the posterior probabilities for each node. This section will focus on the details in constructing a clique-tree for exact inference.

The first step in constructing a clique-tree is to identify the cliques in the triangulated graph. Let G be the fill-in triangulation of some Bayesian network given a perfect elimination scheme f. The algorithm *findCliques* below computes the set of cliques in G. (Note that this algorithm only computes maximal cliques.)

findCliques($G=(V, E)$, f)

$cliqueSet \leftarrow \varnothing$

for each $v \in V$

$clique \leftarrow \{u \mid \{u, v\} \in E \ \wedge f(u) > f(v)\} \cup \{v\}$

if $\forall\ S \in cliqueSet, clique \not\subset S$

$$cliqueSet \leftarrow cliqueSet \cup clique$$

When the *findCliques* algorithm terminates, each element in *cliqueSet* will be a set containing the nodes in one of the cliques in the triangulated graph *G*. Furthermore, every maximal clique in *G* will be contained in some element in *cliqueSet*.

Once the cliques in the triangulated graph have been identified, construction of the clique-tree is fairly straightforward:

1) Make each clique a node in a graph

2) Connect cliques that have nodes in common (*separator nodes*) with an edge

3) Weight each edge with the number of separator nodes between the two cliques

4) Find the maximal spanning tree of the clique graph – the clique-tree

Now, message-passing can be performed on the clique-tree to complete the LS algorithm.

1.3.4 Complexity of Triangulation

Like the problems of exact inference and approximate inference for Bayesian networks, finding the triangulation for a Bayesian network that minimizes the size of the maximum clique is *NP*-hard [Wen90]. Because of this, any currently known triangulation algorithm will perform very poorly on certain networks – producing maximum cliques that are nearly the size of the original graph. In these cases, the LS algorithm for exact inference (which triangulates the graph before propagating belief values with message-passing) will take exponential time. Due to the hardness of

triangulation, algorithms that can bound the maximum clique size or that produce optimal triangulations for certain types of graphs are very useful.

1.4 Information Theory

In this section, I will discuss one of the fundamental concepts in information theory – Kullback-Leibler (KL) information divergence, or relative entropy. This measurement provides a distance between two probability distributions, and can be helpful in determining how well one probability distribution can be approximated by another (probably simpler) distribution. In Bayesian networks, KL divergence can help determine the distance between a probability distribution containing all information and another distribution with one of the edges (conditional dependencies) left out. This way, the importance of the edges in the network can be ranked according to the KL divergence obtained by deleting each edge. Both van Engelen [vanEngelen96] and Kjaerulff [Kjaerulff93] used KL divergence as a way to choose edges to delete in their approximate inference techniques.

1.4.1 Kullback-Leibler Divergence

KL divergence is measured between two probability distributions, *P* and *P'*. Traditionally, *P'* is an approximation of *P*, such as the probability distribution obtained by deleting an edge in a Bayesian network. While KL divergence is often seen as a "distance" between two probability distributions, it is not a distance metric because it is not symmetric. The formal definition for KL divergence is as follows:

Definition. Let *X* be a set of random variables, and let *P* and *P'* be probability distributions on *X*. Let the set *inst* denote all possible instantiations of the variables in *X*. Then the *Kullback-Leibler information divergence* between *P* and *P'* is

$$I(P;P') = \sum_{cur \in inst} P(cur) \times log \frac{P(cur)}{P'(cur)}.$$

1.4.2 Optimized Kullback-Leibler Divergence

When computing the KL divergence between the original probability distribution of a Bayesian network and the probability distribution obtained by deleting an edge, one must step through each possible instantiation of the nodes in the Bayesian network. If a network has *n* binary nodes, then this is 2^n possible instantiations. Thus, using this computation as a part of selecting edges to delete in an approximate inference algorithm is undesirable. As part of his paper on an approximate inference technique for Bayesian networks using edge deletion, van Engelen (in [vanEngelen96]) proposed an optimized computation for KL divergence that only needs to look at the parents of the sink node in the edge being deleted, rather than every node in the network. Below is the formula for the optimized calculation of KL divergence. An example computation of the optimized KL divergence for an edge in a Bayesian network appears in the appendix.

Definition. Let $B = (G = (N, A), P)$ be a Bayesian network. Let $U, V \in N$, for some $(U, V) \in A$. Let *P'* be the probability distribution associated with deleting the edge (U, V) from *B*. Finally, let *inst* be the set of all possible instantiations of $Pa(V) \cup \{V\}$. Then the *optimized KL divergence between P and P'* is

$$I(P;P') = \sum_{\{V=v,Pa(V)=par\}\in inst} P(v|par) \times P(par) \times log\frac{P(v|par)}{P'(v|par)}.$$

CHAPTER TWO: Related Research

In the previous chapter, I briefly introduced two existing approximate inference algorithms that use edge deletion techniques, as well as a triangulation algorithm that uses pre-processing rules to ensure an optimal triangulation for certain types of Bayesian networks. This chapter will discuss each of these algorithms in greater detail, as they form the basis for my own edge deletion algorithms.

2.1 Kjaerulff's Edge Deletion Techniques

The first edge deletion technique was developed by Kjaerulff in [Kjaerulff93]. He proposes selecting edges for removal from the triangulated graph of a Bayesian network. Any edge in the triangulated graph can legally be removed if the graph is still triangulated after its removal. Each "legal" edge will necessarily belong to only one clique. To choose edges for removal, Kjaerulff computes the KL divergence for each edge. This measures the amount of mutual information between two probability distributions (the original distribution and the distribution associated with removing a particular edge), and so it is preferable to remove edges with a low amount of mutual information. (KL information divergence as well as general information theory concepts are discussed much more in Section 1.4 .) Kjaerulff's technique also considers that the removal of a single edge from a triangulated graph can cause many other fill-in edges to become unnecessary, so he removes these newly obsolete edges as well. Finally, he provides the error bound on the posterior probabilities that is introduced by removing a particular edge.

2.2 van Engelen's Edge Deletion Techniques

A second technique for removing edges to reduce network complexity was developed by van Engelen in [vanEngelen96]. Instead of deleting edges from the triangulated graph (which forces the use of a junction tree algorithm for exact inference), van Engelen suggests deleting edges from the original network. After a set of edges have been removed, any exact inference algorithm can be run on the modified network to obtain approximate posterior probabilities. Like Kjaerulff, van Engelen uses KL divergence as a weight for selecting which edges to delete. Because computing the KL divergence takes exponential time and must iterate through all possible values of every node in the network, van Engelen also devised a computation of the divergence that only requires local information. (The details of this local computation are given in Section 1.4.) Again like Kjaerulff, van Engelen provides an upper-bound on the absolute error of the posterior probabilities introduced by deleting an edge.

To help place an upper bound on the posterior probabilities, van Engelen proved that the error for a single posterior probability introduced by deleting edges was bounded by a function of the KL divergence between the original probability distribution and the new probability distribution after a group of edges have been deleted. Let $I(P; P')$ be the KL divergence between the original probability distribution P for a Bayesian network and an approximate distribution P' obtained by deleting a group of edges from the network. van Engelen proved that the absolute bound on the error of the posterior probability for some node V with possible value v is given by

$$|P(V = v) - P'(V = v)| \leq \sqrt{\tfrac{1}{2} I(P; P')}.$$

2.3 Bodlaender's Triangulation Pre-Processing Techniques

The third algorithm directly related to my research is Bodlaender's triangulation algorithm with pre-processing techniques [Bodlaender01]. Bodlaender developed a set of pre-processing rules for triangulation that ensure the triangulation for any graph with treewidth no more than four will be optimal, thereby having a maximum clique of size less than or equal to five. These pre-processing rules work by deleting vertices and possibly adding edges to the original graph. If any vertices remain after the rules have been applied, the new, smaller graph is triangulated with another algorithm. A perfect elimination scheme for the original graph is then constructed by combining the order that the vertices were removed with the perfect elimination scheme for the triangulation of the smaller graph.

Throughout this process, a stack *S* of eliminated vertices is maintained from which the perfect elimination scheme can be constructed. A variable *low* is also updated to contain a lower bound for the treewidth of the original graph. In each update rule, we modify the current graph, possibly update *low*, and possibly add a new vertex to *S*. Below, I give a description of each pre-processing rule and the corresponding triangulation algorithm. A walkthrough of this algorithm on the Asia Bayesian network appears in the appendix.

Simplicial Vertex Rule. Let *v* be a simplicial vertex of degree $d \leq low$. Remove *v* and set *low* to max(*low*, *d*).

Twig Rule. Let *v* be a vertex of degree one. Remove *v*.

Islet Rule. Let *v* be a vertex of degree zero. Remove *v*.

Series Rule. Let v be a vertex of degree two. If $low \geq 2$, then add an edge between the neighbors of v and remove v.

Triangle Rule. Let v be a vertex of degree three such that at least two of its neighbors are adjacent. If $low \geq 3$, then add an edge between every pair of non-adjacent neighbors of v and remove v.

Buddy Rule. Let v, w be vertices of degree three with the same set of neighbors. If $low \geq 3$, then add an edge between every pair of non-adjacent neighbors of v and remove v and w.

Cube Rule. Let a, b, c, d, v, w, x be vertices such that a, b, c, d have degree three. If the following four conditions hold and if $low \geq 3$, then delete a, b, c, and d, and add edges $\{v, w\}$, $\{v, x\}$, and $\{w, x\}$.

1) v is adjacent to a and b
2) x is adjacent to b and c
3) w is adjacent to a and c
4) d is adjacent to a, b, and c

Pre-Processing Algorithm:

The triangulation algorithm for employing the above pre-processing rules is as follows (taken from [Bodlaender01]):

1) G' is initialized to the result of the moralization and removed directionality of the original graph, G. Initialize values: $low \leftarrow 1$, $S \leftarrow \varnothing$.

2) If a reduction rule can be applied to *G'*, it is executed and *G'* is modified accordingly. Each removed vertex is pushed onto the stack *S*, and *low* is updated as specified by the applied rule. This step is repeated until no reduction rules can be applied.

3) If $low < 4$, then $low \leftarrow low + 1$. The reduction is continued in step 2.

4) The graph *G'* that results from the reduction rules is triangulated, yielding a perfect elimination scheme *f*.

5) The perfect elimination scheme *f* is modified by placing all vertices in *S* at the beginning of *f* as they are popped off the stack (all other vertices numbered in *f* have their numbering increased by one).

6) The fill-in of *f* is constructed to yield the triangulation of *G*.

CHAPTER THREE: Methodology

In this chapter, I present the details of my three new approximate inference algorithms for Bayesian networks, called "reduction to polytree," "bounding clique sizes with pre-processing," and "general bounding clique sizes". Each of these algorithms uses edge deletion to reduce the complexity of the network before performing inference. Below, I provide pseudocode and a detailed description for each algorithm. I also provide an upper-bound for the running time of each algorithm, as well as an example walkthrough. In the running time analyses, I use the notation *n* for the number of nodes in a network and the notation *m* for the number of edges. After each algorithm has been described in more detail, I will compare and contrast the new algorithms to previous edge deletion techniques.

I will close this chapter with a discussion of my experimental design for this research. This section will include descriptions of what evaluation criteria I feel are accurate measures of the usefulness of an inference algorithm, as well as why I chose those metrics. I will also discuss which Bayesian networks I plan to use in my tests, and which currently existing inference algorithms I will be using as a basis for comparison.

3.1 Reduction to Polytree

The idea behind the reduction to polytree algorithm is that exact inference is *NP*-hard for general Bayesian networks, but takes linear time for singly-connected networks. Thus, if we delete edges from a Bayesian network until a polytree structure remains, then inference on the resulting polytree can be performed in linear time. In the reduction to polytree algorithm, I weight each edge with the optimized KL divergence between the original distribution and the distribution resulting from deleting that edge from the

network. Thus, an edge weight is low if that edge contains little additional information, and high if the edge holds more information about the network's distribution. I then use Kruskal's maximal spanning tree algorithm, along with the calculated edge weights, to compute the polytree of the Bayesian network that contains the edges which are the most important to the network's probability distribution. Finally, I run Pearl's algorithm on the resulting polytree to get approximate posterior probabilities for each node.

Note from Section 1.4.2 that computing the optimized KL divergence requires computing $P(par)$ for some instantiation *par* of the parent nodes of the sink node for the edge being deleted. Because the exact value for $P(par)$ can only be found using an exact inference algorithm (such as LS, whose exponential running time is what the reduction to polytree algorithm is trying to simplify), I approximate the value in my implementation. To obtain an approximate value of $P(par)$, I sum over all possible joint probabilities of network instantiations that contain the node instantiations in *par*, and then divide by the number of instantiations of the set of nodes not in *par*.

3.1.1 Algorithm Description

Figure 2 contains the pseudocode for the reduction to polytree algorithm. This algorithm takes as input a Bayesian network B and outputs approximate posterior probabilities for each node in the network.

```
reductionToPolytree(B = (G = (N, A), P))
        for each (u, v) ∈ A
                P'← distribution associated with deleting (u, v) from B
                weight(u, v) ← optimized KL divergence between P, P'

        G' ← result of Kruskal's maximal spanning tree algorithm run on G using
```

the *weight* function as edge weights
$B' \leftarrow (G', P'')$, where P'' is the distribution associated with deleting all edges not in G' from B
run Pearl's algorithm on B'
report the posterior probabilities for each node

Figure 2. Reduction to polytree pseudocode

3.1.2 Running Time

In this section, I consider the running time for the reduction to polytree algorithm described in the previous section. For simplicity, I assume that the input network to the algorithm has binary nodes. The first step in the algorithm assigns a weight to each edge (u, v) that corresponds to the optimized KL divergence between the original probability distribution, P, and the probability distribution associated with deleting (u, v) from the network, P'. Recall from Section 1.4.2 that computing the optimized KL divergence between P and P' requires iterating through each possible instantiation the nodes in the set $\{v, Pa(v)\}$. Because the computation depends on the number of parents of each node in the input network, I denote the maximum number of parents for any node by C. The optimized KL divergence must be computed for each edge, so the step of finding the weights for each edge takes time $O(m*2^{C+1})$.

The next step in the algorithm is to run Kruskal's algorithm on the underlying graph of the Bayesian network, using the previously computed optimized KL divergence values as edge weights. Kruskal's algorithm runs in time $O(m*\log m)$. Next, each edge not in the maximal spanning tree must be deleted from the original Bayesian network and the probability distribution of the original network must be updated to reflect the deleted edges. Because all trees have n-1 edges, m-n+1 edges must be deleted from the network

– each deletion can be done in constant time. Updating the probability distribution for each deleted edge is slightly more difficult, as the conditional probabilities for the sink node of each edge must be averaged to reflect that the corresponding source node is no longer its parent. For each deleted edge, this requires stepping through each possible instantiation of the sink node and its parents, and averaging the probabilities for those instantiations that are the same except for the value of the source node. This takes at most $O(2^{C+1})$ time, so to complete this process for all deleted edges takes time $O((m\text{-}n)*2^{C+1})$.

Finally, Pearl's algorithm is run on the resulting singly-connected Bayesian network, and the corresponding posterior probabilities for each node are reported. Pearl's algorithm runs in linear time, so this takes $O(n + m)$ time. Overall, the running time for the algorithm is $O(m*2^{C+1})$. Even though the running time is exponential, the reduction to polytree algorithm performs very well in practice. Because most real-world Bayesian networks do not contain any nodes with more than five parents (which is this case of all networks used in this paper), the exponent C+1 is normally no more than six. Figure 3 contains a summarization of the running time analysis for the reduction to polytree algorithm.

	reductionToPolytree($B = (G = (N, A), P)$)
$O(m*2^{C+1})$	for each $(u, v) \in A$
$O(2^{C+1})$	$P' \leftarrow$ distribution associated with deleting (u, v) from B
$O(2^{C+1})$	*weight*$(u, v) \leftarrow$ optimized KL divergence between P, P'
$O(m*\log m)$	$G' \leftarrow$ result of Kruskal's maximal spanning tree algorithm run on G using the *weight* function as edge weights

$O((m\text{-}n)*2^{C+1})$	$B' \leftarrow (G', P'')$, where P'' is the distribution associated with deleting all edges not in G' from B
$O(n + m)$	run Pearl's algorithm on B'
$O(n)$	report the posterior probabilities for each node
Total running time: $O(m*2^{C+1})$	

Figure 3. Summarization of running time for reduction to polytree

3.1.3 Algorithm Walkthrough

Consider the Asia Bayesian network in Figure 4. This section will go through a walkthrough of the reduction to polytree algorithm on this network.

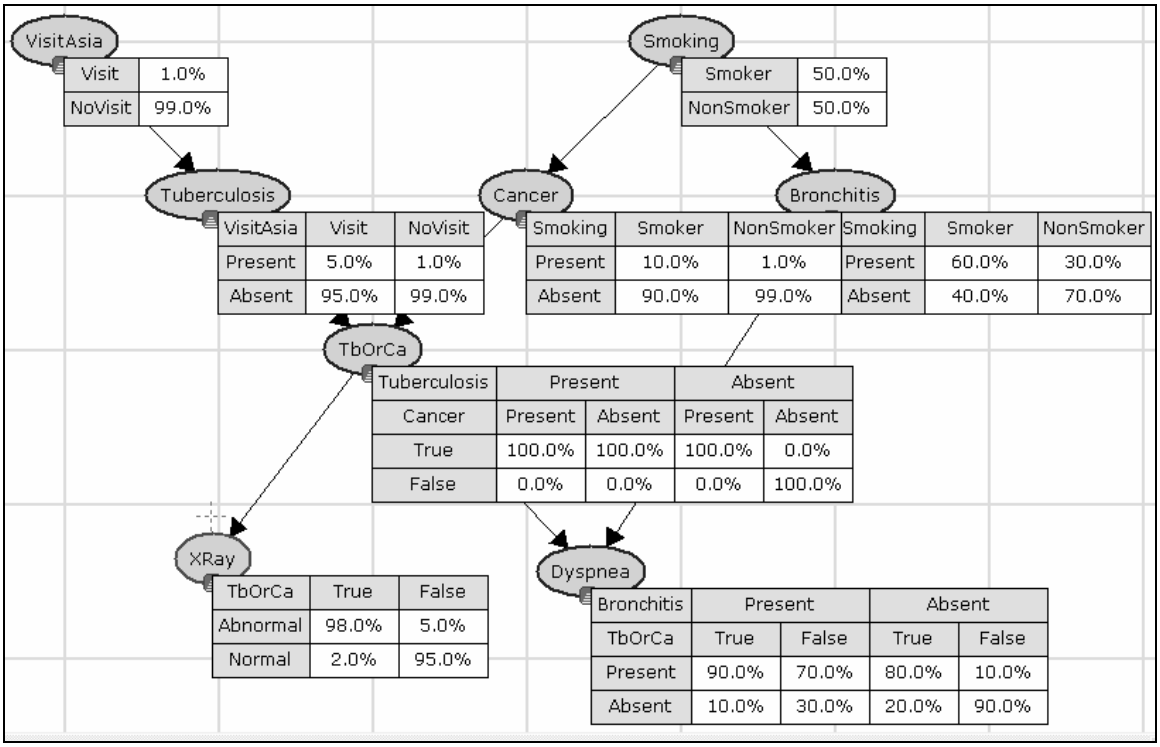

Figure 4. The Asia Bayesian network for reduction to polytree walkthrough

The first step in the reduction to polytree algorithm is to compute the optimized KL divergence for every edge in the network. We can compute this weight for every edge

using the same technique described in Section 1.4.2. (An example of computing the optimized KL divergence for a single edge appears in the appendix.) The Asia network with weight labels on each edge corresponding to the computed KL divergence appears in Figure 5.

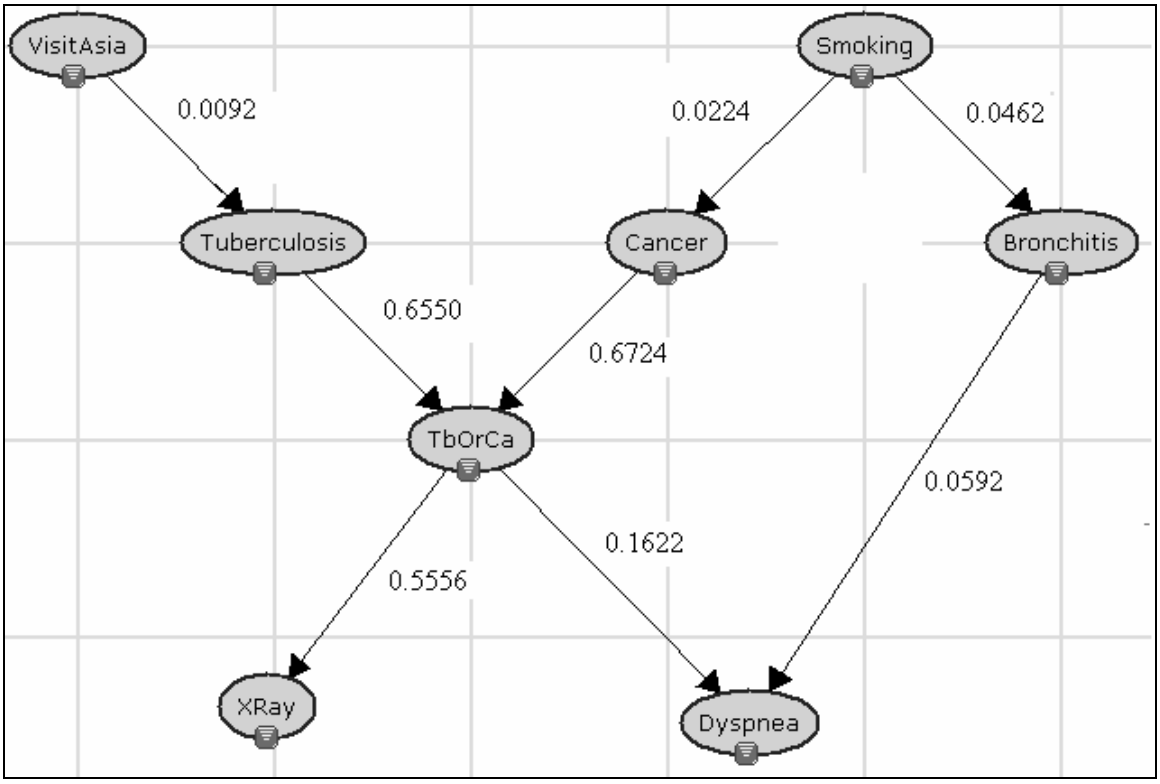

Figure 5. KL divergence values each edge in the Asia network

The next step in the reduction to polytree algorithm is to run Kruskal's maximal spanning tree algorithm, using the KL divergence values above as edge weights. We see from Figure 5 that to convert the Asia network to a polytree, we need to delete an edge from the undirected cycle {Smoking, Cancer, TbOrCa, Dyspnea, Bronchitis, Smoking}. We see that the lowest KL divergence weight for any edge on that cycle is 0.0224 for the edge (Smoking, Cancer). Figure 6 shows the maximal spanning tree that results from Kruskal's algorithm, as well as the new conditional probability tables for each node that

were modified to no longer hold information about the deleted edges. Note that the only edge not including in the spanning tree is (Smoking, Cancer).

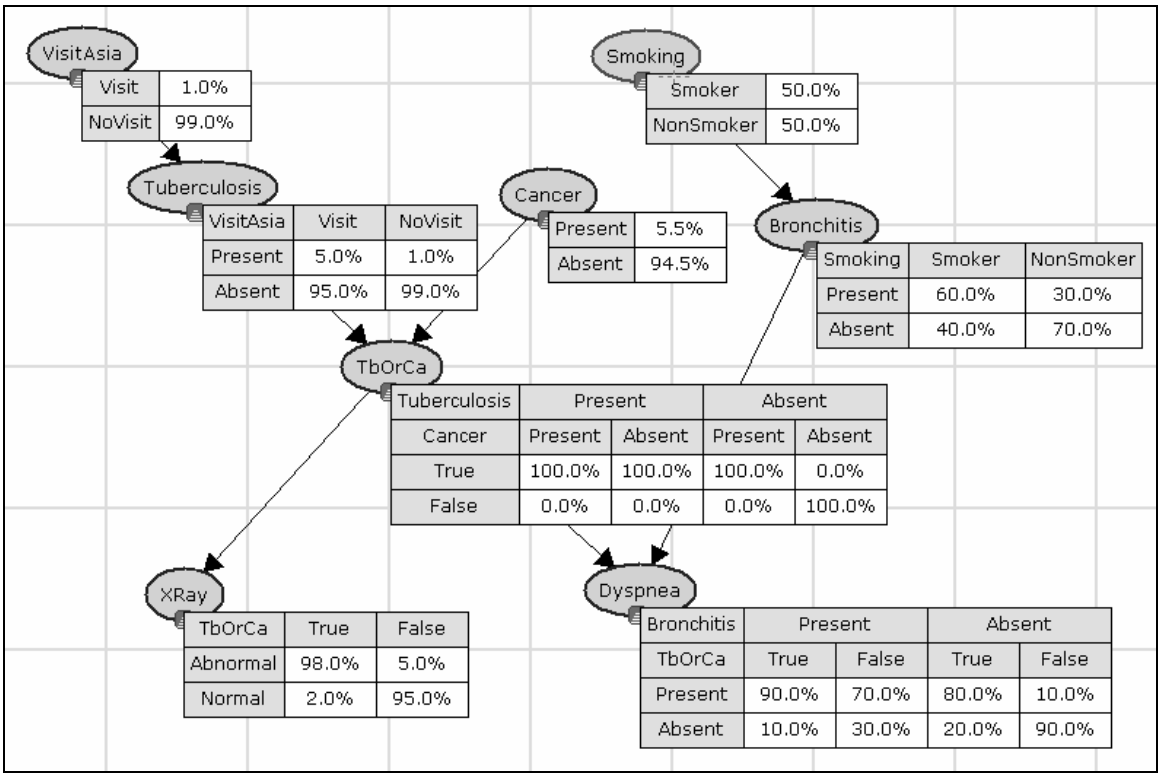

Figure 6. The maximal spanning tree for the Asia network, using KL divergence values from Figure 3 as edge weights

Finally, the last step in the reduction to polytree algorithm is to run Pearl's algorithm on the resulting polytree (in this case, the maximal spanning tree shown in Figure 6) to obtain posterior probabilities on each node. If we run Pearl's algorithm on the polytree in Figure 6 as described in Section 1.2.1, we get the posterior probabilities shown in Figure 7.

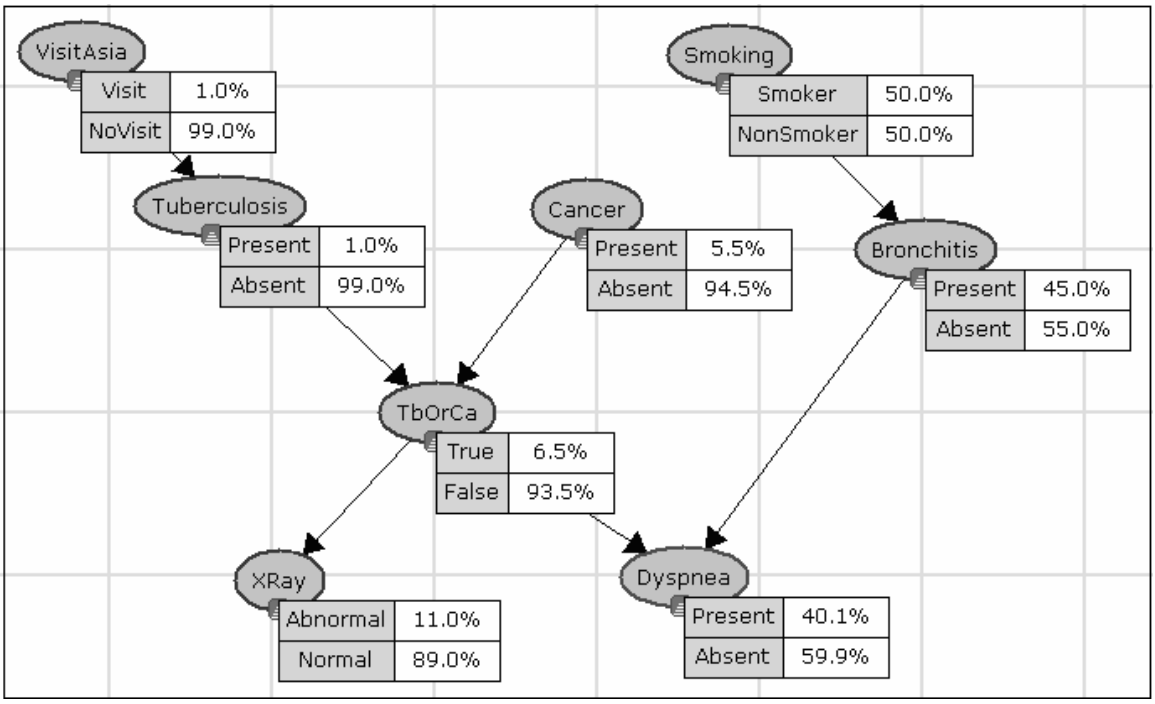

Figure 7. The posterior probabilities for Asia from reduction to polytree technique

3.2 Bounding Clique Sizes with Pre-Processing

My second new approximate inference algorithm, bounding clique sizes with pre-processing, also employs edge deletion techniques to simplify the complexity of the Bayesian network, but it does not reduce the original network all the way to a polytree. Instead, the algorithm chooses edges for deletion before triangulating the graph and running the LS exact inference algorithm to obtain posterior probabilities for each node. Edges are chosen in such a way that the triangulated graph has a maximum clique size of no more than five. Because the LS algorithm has a worst-case exponential running time in terms of the sizes of the cliques in the junction tree, placing a bound on the maximum clique size consequently places a polynomial time bound on the LS algorithm.

In this algorithm I first run Bodlaender's pre-processing rules [Bodlaender01] to get a partial perfect elimination scheme for the vertices. If the treewidth of the original

graph is less than or equal to four, then the pre-processing rules are guaranteed to reduce the original graph to the empty graph. In this case, the perfect elimination scheme will simply be the order in which the vertices were eliminated. If nodes remain, I choose edges for deletion from the original and reduced graph, as well as any moral edges that are no longer needed because of an edge's removal, and continue applying the simplicial vertex rule to eliminate vertices of degree four or less. Finally, I construct the fill-in triangulation of the original graph given the perfect elimination scheme, and run the LS exact inference algorithm on the junction tree corresponding to the triangulated graph.

In this section, I provide pseudocode for the bounding clique sizes with pre-processing algorithm. I also discuss the running time for the algorithm. I do not provide a walkthrough for this algorithm because it behaves identically to Bodlaender's algorithm until the original network has a treewidth of at least five, which usually only occurs in large real-world networks. Section 2.3 describes Bodlaender's pre-processing triangulation rules, and the appendix provides an example walkthrough of the rules on the Asia Bayesian network.

3.2.1 Algorithm Description

Figure 8 contains the pseudocode for the bounding clique sizes with pre-processing algorithm. This algorithm takes as input a Bayesian network B and outputs approximate posterior probabilities for each node in the network.

```
boundCliqueSizes(B = (G = (N, A), P))
    G' = (V, E) ← moralized graph of G
    run Bodlaender's triangulation algorithm on G'
        let S be the stack of eliminated vertices
        let G'' = (V', E') be the remaining graph
```

```
while G'' is not empty
        choose an edge {u, v} ∈ E' or (u, v) ∈ E for deletion
        let E ← E\ {u, v}, E' ← E' \ {u, v}
        remove from E, E' any unnecessary moral edges
        apply simplicial vertex rule (degree < 5), if possible, to G''
        update S, G''

for i ← |N| downto 1
        f(i) ← pop(S)

construct the fill-in triangulation of G' given f
construct the junction tree for the fill-in triangulation
run LS on the junction tree
report the posterior probabilities for each node
```

Figure 8. Bounding clique sizes with pre-processing pseudocode

Before continuing to a description of the running time for this algorithm, I will provide more details about several steps in the algorithm. Consider the step that removes "unnecessary" moral edges from E (and the corresponding edge in E', if it exists) after the selected edge (u, v) has been removed. I define a moral edge $\{u, x\}$ as unnecessary if the following four conditions hold:

1) $\{u, x\} \in E$ -- the moral edge is in the moralized graph

2) $(u, x) \notin A$, $(x, u) \notin A$ – the moral edge is not part of the original graph

3) $(u, v) \in A$ and $(x, v) \in A$ – the moral edge is necessary to join common parents u and x

4) $\forall w \in N, w \neq v$, if $(u, w) \in A$, then $\{u, w\} \notin E$, and if $(x, w) \in A$, then $\{x, w\} \notin E$ – the moral edge does not serve as a moral edge for any other common parents

Finally, consider the step in the algorithm that picks an edge (u, v) from the original edges in G, and then deletes that edge from G' and G''. This step can be performed in a variety of ways, using a different heuristic to select an edge for deletion. One must be careful to not delete an edge that was not part of the original graph G (such as a moral edge), or an edge from the original graph that behaves as a moral edge (such as if the edges (u, w) and (v, w) also exist in the original graph, and (u, v) serves as a moral edge for them). Recall from Section 1.2.1 that moral edges are necessary when using LS for exact inference because they prevent additional conditional independence relations from being added when making the graph undirected. Keeping these considerations in mind, however, any selection criterion may be used.

In my implementation of *boundCliqueSizes*, I first try to select the edge (u, v) for deletion that is in both the original graph G and the reduced graph G'' whose deletion renders the most moral edges unnecessary. If no edge in the reduced graph G'' is available for deletion that satisfies the requirements given in the previous paragraph, then any satisfactory edge in G whose deletion maximizes the number of unnecessary moral edges is chosen. In this case, I do not delete (u, v) from the reduced subgraph G'', but I do remove from G'' and G any unnecessary moral edges.

3.2.2 Running Time

In this section, I consider the running time for the bounding clique sizes with pre-processing algorithm described above. Consider the first step of moralizing the Bayesian network. We must examine each possible pair of nodes to see if a moral edge should be added between them. Thus moralization takes time $O(n^2)$.

Next, we must consider the time required to run Bodlaender's triangulation algorithm with pre-processing techniques. According to Bodlaender, attempting to apply the simplicial vertex rule to every node in the graph can be done in time $O(n^2m)$. Every other pre-processing rules can be tested on every node in the graph in $O(n)$ time [Bodlaender01]. Because a single iteration through Bodlaender's algorithm either applies a pre-processing rule or attempts every possible pre-processing rule on every node with no luck, a single iteration takes at most $O(n^2m)$ time. Furthermore, iterations through the algorithm continue until the *low* variable is greater than four, or until the original network has been reduced to the empty graph. However, we can ignore the *low* variable in the running time analysis because it represents a constant number of iterations in which no rules might be applied. Thus, we can assume that in the worst case a single rule is applied each iteration, resulting in the removal of one vertex, until the original network has been reduced to the empty graph. This yields n iterations, and an overall running time of $O(n^3m)$ for Bodlaender's triangulation algorithm.

Next, we come to the while loop in the algorithm that chooses an edge for deletion, removes that edge from the reduced and original graphs, and removes any newly unnecessary moral edges from the reduced and original graphs. In the worst case, this loop could delete a single edge from both graphs until both the original and reduced

graphs were empty – thus, the loop can iterate at most m times. The first step inside the loop is to choose an edge for removal from the original network. Each iteration, I remove the edge in the reduced graph (that is also in the original graph) that renders the most moral edges unnecessary. If no such edge exists, then I remove the edge in the original graph that makes the most moral edges unnecessary. In each case, I ensure that the chosen edge does not serve as a moral edge between two vertices.

Essentially, I must examine each edge in the graph (possibly twice, if I examine an edge once because it is in the reduced graph and again because it is in the original graph), and check that it does not serve as a moral edge between two vertices and obtain a score on the number of moral edges that would be unnecessary if it was deleted. The first check requires looking to see if both the sink and the source of the selected edge have a common child, which can be done in time $O(m)$. Computing the score for the selected edge (u, v) requires examining each other child x of u for the edge and determining if all of the four properties described in Section 3.3.1 are satisfied. The first three properties can be examined in time $O(m)$, and the fourth property can be determined in time $O(mn)$. Since these properties must be evaluated for each other child x of u, computing the score for (u, v) takes time $O(mn^2)$ in the worst case. Thus evaluating a single edge for possible removal can be performed in time $O(mn^2)$, and evaluating all edges can be done in time $O(m^2n^2)$. Determining the best score among all examined edges takes time $O(m)$, so the entire step of choosing an edge for deleting takes time $O(m^2n^2)$.

Once an edge is chosen for removal, that edge and all unnecessary moral edges must be deleted from the original graph, which can be done in $O(m)$ time. After removing the moral edges, I must attempt to apply the simplicial vertex rule to every

node in the reduced graph, which takes time $O(n^2m)$ [Kjaerulff93]. Thus, a single iteration of the loop takes time $O(m^2n^2)$, and so the entire loop takes time $O(m^3n^2)$.

After completing the loop, we construct the perfect elimination scheme for the original graph by popping each vertex off the stack S one by one, which takes $O(n)$ time. The next step in the algorithm constructs the fill-in triangulation of the original Bayesian network given the perfect elimination scheme. Computing a fill-in triangulation involves examining each pair of nodes and determining if a fill edge needs to be added between them, which can be done in time $O(n^2)$. Next, we must construct a junction tree (or tree of cliques) for the triangulated graph. Identifying the cliques in the junction tree takes time $O(m+n)$ [LauritzenSpiegelhalter88], and constructing the junction tree from the cliques can be done using Kruskal's algorithm, which requires $O(m \log m)$ time.

The last step in the algorithm is the inference portion which runs LS on the constructed junction tree. This involves message-passing, as in Pearl's algorithm, between each clique in the junction tree and then calculating the posterior probabilities of each node by marginalizing the probabilities assigned to each clique (called potentials). The message-passing and marginalization is exponential in the size of the biggest clique in the junction tree [LauritzenSpiegelhalter88]. Because all cliques in the junction tree are no bigger than five, this step in the algorithm (and hence the entire algorithm) runs in time $O(\max\{m^3n^2, p^5\})$, where p^5 is the polynomial bound for the LS algorithm and $O(m^3n^2)$ is the worst-case running time for the remainder of the algorithm. A summarization of the running time analysis for the bounding clique sizes algorithm appears in Figure 9.

	boundCliqueSizes($B = (G = (N, A), P)$)	
$O(n^2)$	$G' = (V, E) \leftarrow$ moralized graph of G	
$O(n^3m)$	run Bodlaender's triangulation algorithm on G'	
		let S be the stack of eliminated vertices
		let $G'' = (V', E')$ be the remaining graph
$O(m^3n^2)$	while G'' is not empty	
	$O(m^2n^2)$	choose an edge $\{u, v\} \in E'$ or $(u, v) \in E$ for deletion
	$O(m)$	let $E \leftarrow E \setminus \{u, v\}$, $E' \leftarrow E' \setminus \{u, v\}$
	$O(m)$	remove from E, E' any unnecessary moral edges
	$O(n^2m)$	apply simplicial vertex rule (degree < 5), if possible, to G''
		update S, G''
$O(n)$	for $i \leftarrow \|N\|$ downto 1	
		$f(i) \leftarrow pop(S)$
$O(n^2)$	construct the fill-in triangulation of G' given f	
$O(m \log m)$	construct the junction tree for the fill-in triangulation	
$O(p^5)$	run LS on the junction tree	
$O(n)$	report the posterior probabilities for each node	

Total running time: $O(\max\{m^3n^2, p^5\})$

Figure 9. Summarization of running time for bounding clique sizes with pre-processing

3.3 General Bounding Clique Sizes

My third algorithm extends the idea of bounding the maximum clique size of the triangulated network to allow the user to input a desired clique bound. The algorithm takes as input a Bayesian network, a desired maximum clique bound b for the triangulated network, and a perfect elimination scheme for the nodes in the network that

is obtained from some (exact or approximate) triangulation algorithm. The algorithm will then output the Bayesian network with a set of edges removed such that the triangulation of the modified Bayesian network will have a maximum clique size of no more than *b* when triangulated using the fill-in of the given perfect elimination scheme. Again, because LS has a worst-case exponential running time in terms of the sizes of the cliques in the junction tree, placing a bound on the maximum clique size consequently places a polynomial time bound on the LS algorithm.

In this section, I provide pseudocode for the general bounding clique sizes algorithm. I also discuss the running time for the algorithm, and provide a walkthrough of its execution on the Asia Bayesian network.

3.3.1 Algorithm Description

Figure 10 contains the pseudocode for the general bounding clique sizes algorithm. The input *B* is the initial Bayesian network, *f* is a perfect elimination scheme for *B*, and *b* is the desired bound for the maximum clique size of the fill-in triangulation of *B* given *f*.

```
generalBoundCliqueSizes(B = (G = (N, A), P), f, b)
    G' = (V, E) ← moralized graph of G
    FillIn ← the fill-in triangulation of G' given f
    maxClique ← the set of nodes in the maximum clique of FillIn

    while sizeof maxClique > b
        pick an edge (u, v) ∈ A
        G' ← (V, E\{u, v})
        remove unnecessary moral edges from G'
        FillIn ← the fill-in triangulation of G' given f
```

maxClique ← the set of nodes in the maximum clique of *FillIn*

report *FillIn* as the triangulation of *G'* with max clique no greater than *b*
run LS on the junction tree for *FillIn*
report the posterior probabilities for *G'* computed by LS

Figure 10. General bounding clique sizes pseudocode

Before discussing the running time of *generalBoundCliqueSizes*, I provide more details on several of the less straightforward steps in the algorithm. Consider first the step that determines the set of nodes in the maximum clique of *FillIn*. This can be accomplished by iterating through each node in *FillIn* and creating a set with that node and its higher-ordered neighbors, called that node's "higher-clique set". After looking at each node, the largest set contains the nodes that form the maximum clique in *FillIn*.

Next, consider the step that removes "unnecessary" moral edges from *G'* after the selected edge (*u*, *v*) has been removed. A moral edge is considered unnecessary if it satisfies the four conditions described in Section 3.2.1. Finally, consider the step in the algorithm that picks an edge (*u*, *v*) from the original edges in *G*, and then deletes that edge from *G'*. This step can be performed in a variety of ways, using a different heuristic to select an edge for deletion. Just as for the pre-processing bounding clique sizes algorithm, one must be careful not to select a moral edge for deletion – but any selection criterion that satisfies this condition is acceptable.

In my implementation of *generalBoundCliqueSizes*, I did preliminary testing on a variety of heuristics for selecting which edge to delete next. I tried selecting edges with the minimum KL divergence, edges whose deletion caused the greatest reduction in the

number of edges in the *FillIn* graph, and a combination of the two. For all three of these approaches, I tried selecting edges from the entire graph or only ones adjacent to some node in the maximum clique. However, the heuristic that performed the best in my preliminary tests was to select the edge adjacent to some node in the maximum clique whose deletion renders the most moral edges unnecessary. If no edge adjacent to some node in the maximum clique is available for deletion, then any satisfactory edge whose deletion maximizes the number of unnecessary moral edges is chosen.

3.3.2 Running Time

In this section, I consider the running time for the general bounding clique sizes algorithm described above. The first step of moralizing the Bayesian network takes time $O(n^2)$, as was described in Section 3.2.2. The next step of computing the fill-in triangulation of the moralized graph given the inputted perfect elimination scheme f takes time $O(n^2)$, which was also described in Section 3.2.2. The last step in the initialization phase is to compute the set of nodes that form the maximum clique in the fill-in triangulation, using the process described in Section 3.3.1 above. This process requires iteration through each node in the network, constructing the higher-clique set for each node, and then determining which higher-clique set contains the most nodes. Computing the higher-clique set for a single node requires examining the node's set of neighbors, and adding to the higher-clique set all neighbors with a higher ordering than the node itself – which can be done in $O(n)$ time. Repeating this process for each node and then selecting the biggest higher-clique set as the maximum clique for the triangulated graph brings the total time for determining the nodes in the maximum clique up to $O(n^2)$.

Next, we consider the time needed to execute each step inside the while loop in the algorithm. The first step inside the loop is to choose an edge for removal from the original network, and then to remove that edge. The criteria for removing edges is the same as for bounding clique sizes with pre-processing, so this step takes time $O(m^2n^2)$ (see Section 3.2.2). Once an edge is chosen for removal, that edge and all unnecessary moral edges must be deleted from the original graph, which can be done in time $O(m)$, again according to Section 3.2.2. Finally, to complete a single iteration of the while loop, we must re-compute the fill-in triangulation of the new network (given that several edges have been deleted) and determine the new maximum clique in the triangulation. As was described previously in this section, these steps both take time $O(n^2)$. Thus the time required for a single iteration of the while loop is $O(m^2n^2)$.

The while loop in the algorithm continues iterating until the size of the maximum clique is less than or equal to b. Each iteration removes at least one edge from the original graph, so this loop cannot continue for more than m iterations. Thus the entire while loop can be executed in time $O(m^3n^2)$.

Finally, we must construct a junction tree for the triangulated graph, run LS on that junction tree, and report the posterior probabilities computed by LS. As was described in Section 3.2.2, constructing a junction tree can be done in time $O(m \log m)$, and running LS is exponential in the size of the largest clique in the junction tree. Thus this step of the algorithm is polynomially bounded with degree b, and the entire algorithm runs in time $O(\max\{m^3n^2, p^b\})$, where p^b is the polynomial bound for the LS algorithm. A summarization of the running time analysis for the general bounding clique sizes algorithm appears in Figure 11.

	generalBoundCliqueSizes($B = (G = (N, A), P)$, f, b)	
$O(n^2)$	$G' = (V, E) \leftarrow$ moralized graph of G	
$O(n^2)$	*FillIn* ← the fill-in triangulation of G' given f	
$O(n^2)$	*maxClique* ← the set of nodes in the maximum clique of *FillIn*	
$O(m^3n^2)$	while sizeof *maxClique* > b	
	$O(m^2n^2)$	pick an edge $(u, v) \in A$
	$O(m)$	$G' \leftarrow (V, E \backslash \{u, v\})$
	$O(m)$	remove unnecessary moral edges from G'
	$O(n^2)$	*FillIn* ← the fill-in triangulation of G' given f
	$O(n^2)$	*maxClique* ← the set of nodes in the maximum clique of *FillIn*
$O(n^2)$	report *FillIn* as the triangulation of G' with max clique no greater than b	
$O(p^b)$	run LS on the junction tree for *FillIn*	
$O(n)$	report the posterior probabilities for G' computed by LS	
Total running time: $O(\max\{m^3n^2, p^b\})$		

Figure 11. Summarization of running time for general bounding clique sizes

3.3.3 Algorithm Walkthrough

Consider again the Asia Bayesian network in Figure 4. This section will go through a walkthrough of the general bounding clique sizes algorithm on this network. Assume that the algorithm is given as input a desired maximum clique bound of three and the perfect elimination scheme f produced by running maximum cardinality search on Asia. (For a walkthrough of maximum cardinality search on Asia, consult the appendix.) The values of f are:

$f(1)$ = VisitAsia, $f(2)$ = Tuberculosis,

$f(3)$ = Cancer, $f(4)$ = TbOrCa,

f(5) = Smoking, *f*(6) = Bronchitis,

f(7) = Dyspnea, *f*(8) = XRay.

Initialization:

First, we must moralize Asia by adding edges between co-parents and removing the directionality of the edges. This requires adding the edges {Tuberculosis, Cancer} and {TbOrCa, Bronchitis}. The moralized graph for the Asia Bayesian network appears in Figure 12 below.

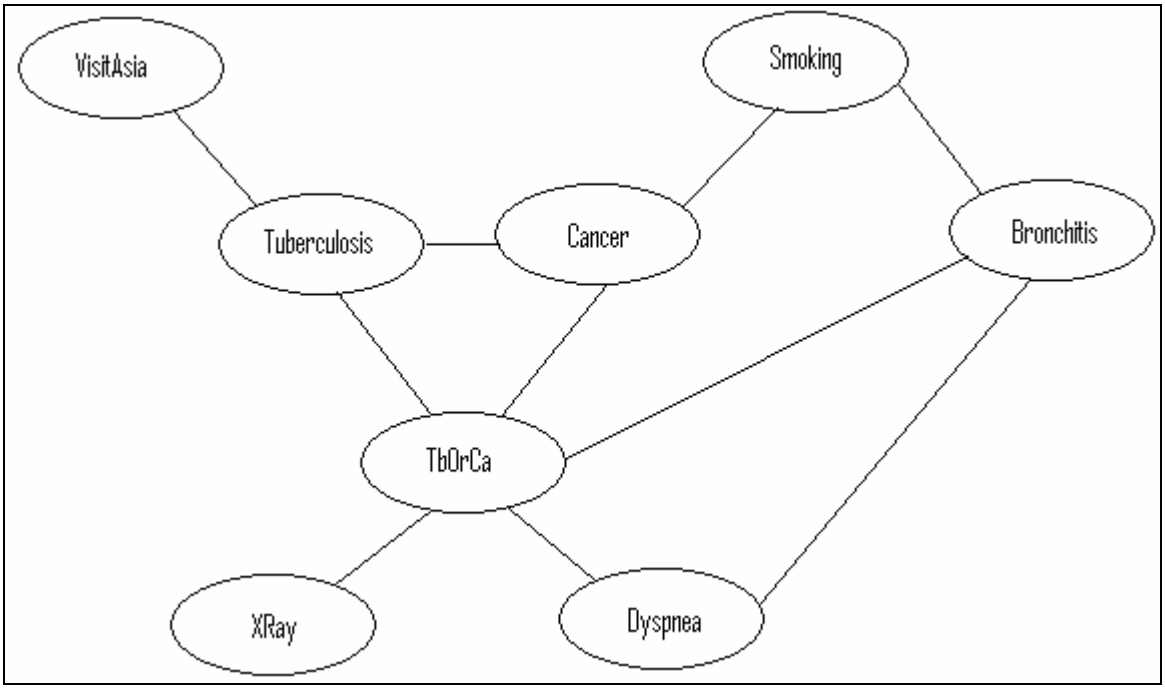

Figure 12. The moralized graph for the Asia Bayesian network

Next, we must compute the fill-in triangulation of the moralized graph given *f*. Thus for each node *v*, we must turn *v*'s higher-ordered neighbors into a clique. This process is shown in detail in the maximum cardinality search walkthrough in the

appendix, and requires adding the edges {Smoking, TbOrCa}, {XRay, Bronchitis}, {Smoking, XRay}, {Smoking, Dyspnea}, and {XRay, Dyspnea} The triangulation of Asia given f appears in Figure 13 below.

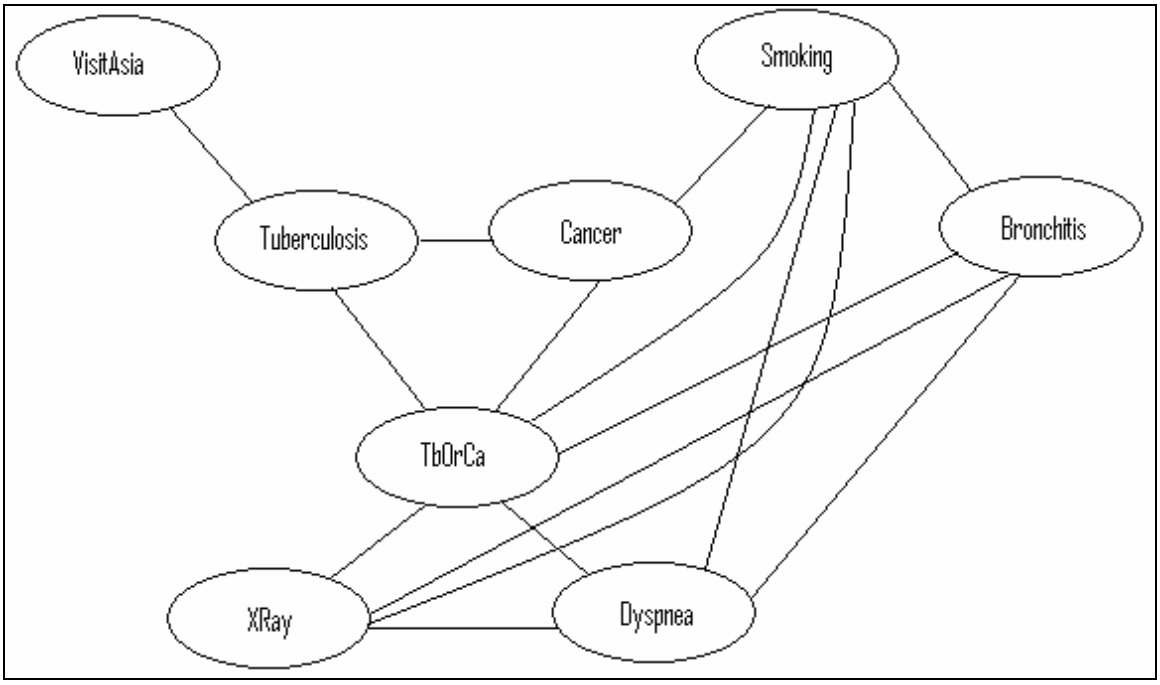

Figure 13. The triangulation of Asia using maximum cardinality search

Finally, we must determine the set of nodes in the maximum clique of the triangulated graph. This can be done by computing the higher-clique set for each node in the graph, which is the set of higher-ordered neighbors (and the node itself) of each node. A list of each node and its higher-clique set appears in Table 1.

Node Name	Higher-Clique Set
VisitAsia	{VisitAsia, Tuberculosis}
Tuberculosis	{Tuberculosis, Cancer, TbOrCa}
Cancer	{Cancer, TbOrCa, Smoking}
TbOrCa	{TbOrCa, Smoking, Bronchitis, XRay, Dyspnea}
Smoking	{Smoking, Dyspnea, Bronchitis, XRay}
Bronchitis	{Bronchitis, Dyspnea, XRay}
Dyspnea	{Dyspnea, XRay}
XRay	{XRay}

Table 1. Higher-clique sets for computing the maximum clique in Asia's triangulation

From consulting Table 1, we can see that the maximum clique in Asia's triangulation is {TbOrCa, Smoking, Dyspnea, Bronchitis, XRay} and has size five.

Main step 1, max clique size = 5:

Now we begin the main loop of the algorithm, which continually selects edges for deletion from the original Bayesian network until the desired maximum clique bound of three is reached. To select an edge for deletion, we must determine for each edge adjacent to some node in the maximum clique the number of moral edges that would become unnecessary if that edge were deleted. Table 2 below shows each edge in the original Bayesian network that is adjacent to some node in the maximum clique, and which moral edges would become unnecessary if that edge was deleted.

Candidate Edge	Unnecessary Moral Edges
(TbOrCa, Dyspnea)	{TbOrCa, Bronchitis}
(Cancer, TbOrCa)	{Tuberculosis, Cancer}
(TbOrCa, XRay)	none
(Tuberculosis, TbOrCa)	{Tuberculosis, Cancer}
(Smoking, Cancer)	none
(Smoking, Bronchitis)	none
(Bronchitis, Dyspnea)	{TbOrCa, Bronchitis}

Table 2. Unnecessary moral edges for candidate edges for deletion

By consulting Table 2, we learn that choosing four different edges for deletion would each cause one moral edge to be unnecessary. We choose the edge (TbOrCa, Dyspnea) for deletion since it is the first edge we examined that would cause a moral edge to be unnecessary. We thus remove (TbOrCa, Dyspnea) and the moral edge {TbOrCa, Bronchitis}, and reconstruct the fill-in triangulation given *f*, remembering that the moral graph of Asia now has two fewer edges. The new fill-in triangulation appears in Figure 14 below.

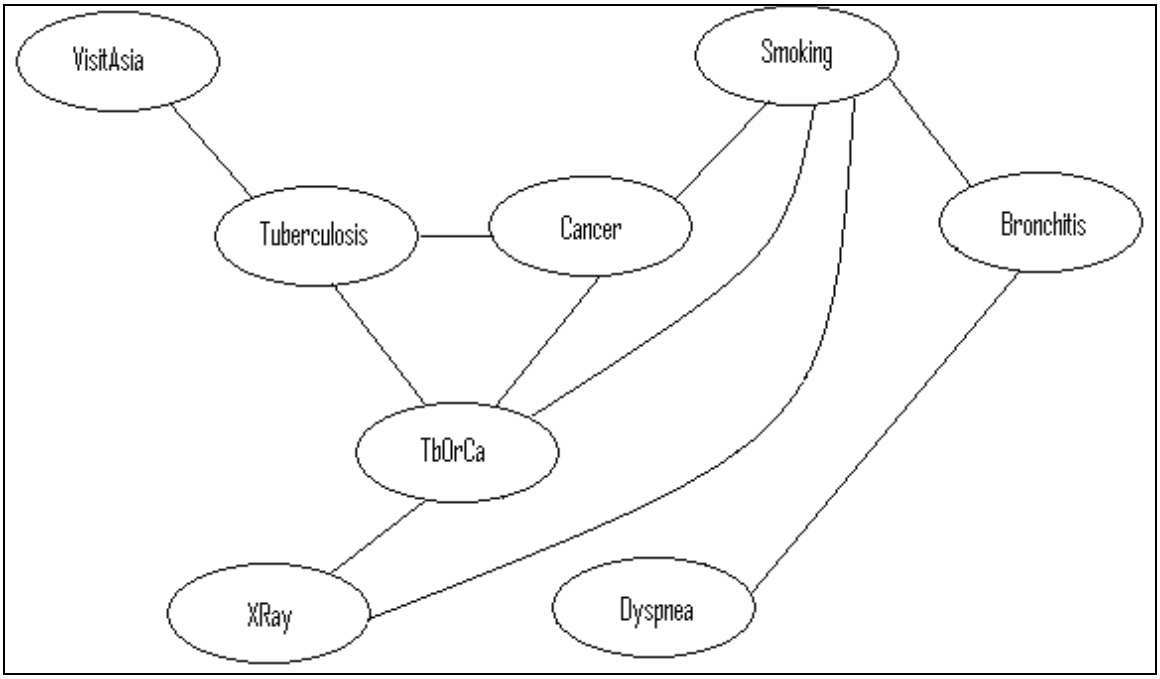

Figure 14. Triangulated graph after step 1 in general bounding clique sizes algorithm

By applying the same technique as in the initialization phase (constructing the "higher-clique sets" for each node) we learn that there are several cliques in the new triangulated graph that have size three, but that there is no clique that is bigger than three. Thus, after choosing one edge for deletion from the original graph, we have achieved our desired maximum clique bound of three.

Now, we can construct a junction tree from the triangulated graph in Figure 14 and run LS on that junction tree to obtain the posterior probabilities for each node, as described in Section 1.2.1. The resulting posterior probabilities appear in Figure 15 below. Note that the edge (TbOrCa, Dyspnea), which was present in the original network, is missing in Figure 15 below. This is because to get the clique size down to three in the fill-in triangulation graph, this edge had to be deleted.

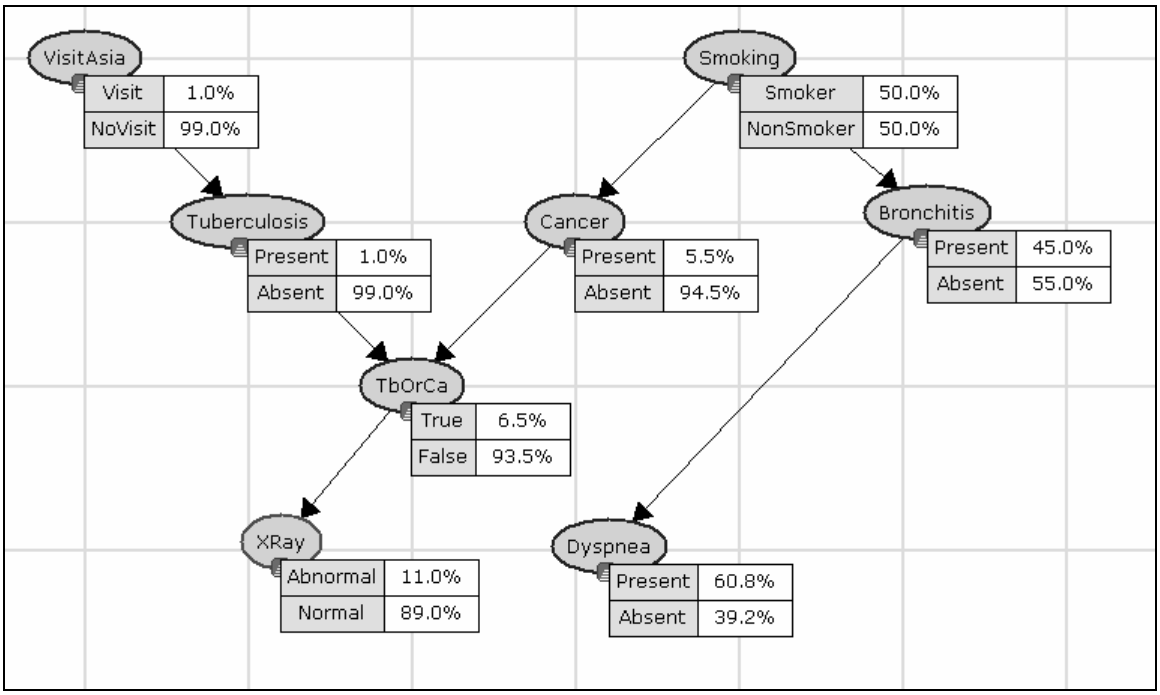

Figure 15. The posterior probabilities for Asia using general bounding clique sizes

3.4 Comparisons of New Algorithms to Past Techniques

Chapter 2 presented two previous techniques that used edge deletion – the first by Kjaerulff in [Kjaerulff93] and the second by van Engelen in [vanEngelen96]. Both Kjaerulff and van Engelen emphasized the fact that many Bayesian networks contain edges that add a lot to the complexity of the network but add little information to the probability distribution. They also both focus on deleting edges to stay within a certain

bound of the exact posterior probabilities. However, neither of these techniques addressed the speedup associated with deleting edges with low information. Additionally, neither author considered deleting edges to enforce structural constraints on the modified network that would place a polynomial time bound on inference.

The three new algorithms in this thesis also use edge deletion, and the reduction to polytree algorithm uses KL divergence to select edges for deletion, just like Kjaerulff and van Engelen. However, my techniques focus on deleting edges to either force the modified network to be a polytree or to ensure that the triangulation of the modified network has clique sizes within a certain bound. The goal of these new algorithms is to provide polynomial-time approximate inference, while the goals of Kjaerulff and van Engelen are to delete edges while remaining within a certain error bound.

3.5 Experimental Design

This section contains the criteria for evaluating the usefulness of an approximate inference algorithm for Bayesian networks. It describes which Bayesian networks will be used in testing and why they are good choices. In this section, I also discuss how to test the speed and accuracy of an approximate inference algorithm, including which currently existing algorithms are good bases for comparison.

3.5.1 Networks Used

Table 3 below describes the ten Bayesian networks selected to help test the speed and accuracy of my approximate inference algorithms: Sprinkler, Asia, Alarm-13, Insurance, Water, Alarm, Barley, CPCS-54, Hailfinder, and CPCS-179. These networks were chosen for the experiment because they represent a large range in size – from four nodes and four edges up to 179 nodes and 239 edges. Each of these networks is either a

"toy" network commonly used in examples in Bayesian network research, or is a hand-constructed real-world network that models a specific domain. Each network is also freely available in the public domain, and as such is commonly used for testing. Table 3 also includes the maximum clique size for each network when triangulated with the maximum cardinality search (MCS) algorithm. Recall that the speed of the LS exact inference algorithm is exponential in the size of the maximum clique in the triangulated graph.

Network Name	**Number of Nodes**	**Number of Edges**	**Max Clique Size (MCS)**
Sprinkler	4	4	3
Asia	8	8	4
Alarm-13	13	14	4
Insurance	27	52	10
Water	32	66	12
Alarm	37	46	5
Barley	48	84	9
CPCS-54	54	108	18
Hailfinder	56	66	6
CPCS-179	179	239	9

Table 3. Summary of Bayesian networks used in experiment

3.5.2 Algorithms Tested

I will perform speed and accuracy tests on LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes. I decided to use LS and AIS for comparison because LS is a version of the standard junction tree algorithm, and AIS is often regarded as the best approximate inference algorithm. I do not provide any comparison between the techniques of Kjaerulff [Kjaerulff93] and van Engelen [vanEngelen96] because they are both error-bounding techniques for reducing network complexity and not actual inference algorithms. Their research focuses on

proving that removing a particular set of edges would not introduce error beyond a desired bound, not on inference speedup.

Because the general bounding clique sizes algorithm can be run with any desired clique bound, I will test this algorithm multiple times with different bounds. I supply the general bounding clique sizes algorithm with the initial perfect elimination scheme from MCS, which is also the triangulation algorithm used in the implementation of LS. I will supply the general bounding clique sizes algorithm with the different clique bounds of 4, 5, 6, 8, 10, and 12. These bounds allow for a range of edges to be deleted from each network, given the initial maximum clique sizes in Table 3. However, the general bounding clique sizes will always be given a clique bound of five when it is to be directly compared to the bounding clique sizes with pre-processing algorithm. This is because the bounding clique sizes with pre-processing algorithm also forces the maximum clique size in the triangulated graph to be five or less.

The AIS algorithm, like all sampling algorithms, is designed to be given a desired number of samples to take of the network, and then to report the estimated posterior probabilities based on those samples. While allowing AIS to take a very large number of samples usually yields more accurate results than taking a smaller batch of samples, this process can make AIS run slower than LS. In order to keep the time needed to run AIS similar to the time required to run the other inference algorithms, I will limit the number of samples taken by AIS to 2,000 for each of the tests.

3.5.3 Evaluation Techniques

This subsection describes what specific evaluation techniques will be used on the new approximate inference algorithms to determine how fast and accurate they are as

compared to currently existing algorithms. The reduction to polytree technique and bounding clique sizes with pre-processing will be performed as described in Sections 3.1.1 and 3.2.1, while the general bounding clique sizes algorithm described in section 3.3.1 will be tested with several different bounds on the maximum clique size: 4, 5, 6, 8, 10, and 12. The results of running each new inference algorithm -- the first two algorithms and the general bounding clique sizes algorithm with each different maximum clique size bound -- for the Bayesian networks in Table 3 appear in the results section in Chapter 5.

Speed Evaluation:

An important factor for determining the usefulness of any approximation algorithm or heuristic is its speed. Because approximation algorithms usually produce results with at least some error, it is important that they make up for loss of information by being considerably faster than an exact technique.

In Bayesian network inference, however, the speed of the entire algorithm is less important than the speed of the inference portion of the algorithm. Each new approximate inference algorithm involves a lengthy pre-processing stage of triangulating the graph and/or determining which edges to delete. However, this pre-processing stage need only be done once -- whereas the inference portion would need to be repeated again and again with different node values as evidence. Thus I will report the inference time as well as the total time for each algorithm when a test is run.

In order to give an accurate speed comparison for how these algorithms will be used in practice, I will also determine the total time to run each algorithm 100 times –

which is significantly fewer times than an algorithm is likely to be run when used in a practical setting like a medical diagnostic system. However, the total time for running an algorithm 100 times will begin to show the trends of which algorithm will be faster over the long run. The total time for 100 runs of some algorithm will be computed as the sum of the pre-processing time for the algorithm (determining which edges to delete and/or building the junction tree) and the time required to run the inference portion of the algorithm 100 times. Note that in the case of the AIS algorithm, which has no pre-processing step, the total time for 100 runs will just be 100 times the total time for the algorithm. Again, the results for each of these tests on the networks listed in Table 3 appear in the results section in Chapter 5.

Accuracy Evaluation:

While the speed of an approximation algorithm is the most important factor for determining that algorithm's usefulness, it is certainly not the only factor worth considering. If an approximation algorithm returns results that are considerably different than an exact algorithm for the same problem, then that approximation algorithm is useless – no matter how fast it can run. To evaluate the accuracy of my approximate inference algorithms, I plan to compare the posterior probabilities for each node reported by the approximate algorithm to the posterior probabilities reported by an exact algorithm (namely, LS) by calculating the root mean-squared-error (RMSE) between the two sets of probabilities. The formula for RMSE is:

$$RMSE = \sqrt{\frac{1}{total}\sum_{i=1}^{n}\sum_{j=1}^{inst(i)}[P(\theta_{ij}) - P'(\theta_{ij})]^2}$$

where n is the number of nodes, $inst(i)$ is the number of possible values for node i, $total$ is $inst(1)+inst(2)+\ldots+inst(n)$, $P(\theta_{ij})$ is the posterior probability reported by LS for the j^{th} instantiation of node i, and $P'(\theta_{ij})$ is the posterior probability reported by an approximate inference algorithm for the j^{th} instantiation of node i.

For comparison, I will also compute the RMSE of AIS, where AIS is run with 2,000 samples of the network. In the Chapter 5, I will report the RMSE for each new approximate inference algorithm and AIS on each network in Table 3.

CHAPTER FOUR: Error Bounds and Pathological Cases

In Chapter 5, we will see the results of the three new approximate inference algorithms when run on ten different real-world Bayesian networks. While that chapter will be devoted to experimental results that show how fast and accurate the algorithms are in practice, this chapter will focus on how accurate the algorithms are in theory. Because the reduction to polytree algorithm is the only technique that chooses edges for deletion based on a measurement (KL divergence) that reflects how well the reduced network approximates the original network, it is the only algorithm for which I can provide a guaranteed bound on how different the approximate posterior probabilities will be from the exact values. However, this chapter does provide pathological cases for all three algorithms for which they will be very inaccurate. Again, just because the algorithms perform poorly on these example cases does not mean that such cases are likely to arise in practice – instead, they provide us with an idea of how bad the algorithms can be. We will see in the results section in Chapter 5 that all three algorithms produce accurate results on real-world networks.

4.1 Reduction to Polytree

Because the reduction to polytree algorithm uses the KL divergence measurement when choosing edges for deletion, we can take advantage of the same error-bounding properties on the posterior probabilities proved by van Engelen in [vanEngelen96]. Recall from Section 2.2 that if we let $I(P, P')$ be the KL divergence between the original probability distribution P for a Bayesian network and an approximate distribution P' obtained by deleting a group of edges from the network, then the absolute bound on the error of the posterior probability for some node V with possible value v is given by

$$|P(V=v)-P'(V=v)| \leq \sqrt{\tfrac{1}{2}I(P;P')}.$$

Because neither the absolute value expression on the left-hand side nor the square root on the right-hand side will ever be less than zero, we can derive that

$$[P(V=v)-P'(V=v)]^2 \leq \tfrac{1}{2}I(P;P').$$

Because this inequality holds for all nodes *V* and each possible node value *v*, we can add all possible forms of this inequality (for all nodes and node values) to get

$$\sum_{i=1}^{n}\sum_{j=1}^{inst(i)}[P(\theta_{ij})-P'(\theta_{ij})]^2 \leq \frac{total}{2}I(P;P')$$

where *n* is the number of nodes, *inst*(*i*) is the number of possible values for node *i*, *total* is *inst*(1)+*inst*(2)+…+*inst*(*n*), $P(\theta_{ij})$ is the posterior probability reported by LS for the j^{th} instantiation of node *i*, and $P'(\theta_{ij})$ is the posterior probability reported by an approximate inference algorithm for the j^{th} instantiation of node *i*.

Next, by dividing both sides by total and taking the square root, we get

$$\sqrt{\frac{1}{total}\sum_{i=1}^{n}\sum_{j=1}^{inst(i)}[P(\theta_{ij})-P'(\theta_{ij})]^2} \leq \sqrt{\tfrac{1}{2}I(P;P')}.$$

Notice now that the right-hand side of the inequality is the RMSE formula presented in Section 3.5.3. Thus the RMSE between the approximate posterior probabilities and the exact result is bounded by a function of the KL divergence between the original probability distribution and the approximate distribution that results from deleting edges:

$$RMSE \leq \sqrt{\tfrac{1}{2} I(P;P')}.$$

This result suggests that we can calculate an upper bound for the RMSE after selecting a set of edges for deletion by calculating the KL divergence between the original distribution and the distribution resulting from deleting that set of edges. Thus we can determine an upper bound on the error introduced by the reduction to polytree technique without having to run an exponential-time exact inference algorithm to compare our results.

Because the reduction to polytree technique must delete edges until the reduced network has a polytree structure, it may have to delete as many as $n^2 - (n - 1)$ edges for a complete network. Furthermore, the approximate distribution that results from deleting a single edge is computed by averaging entries in the sink node's probability table. If the entries in the table that are averaged are very different, then the approximate distribution will lose information.

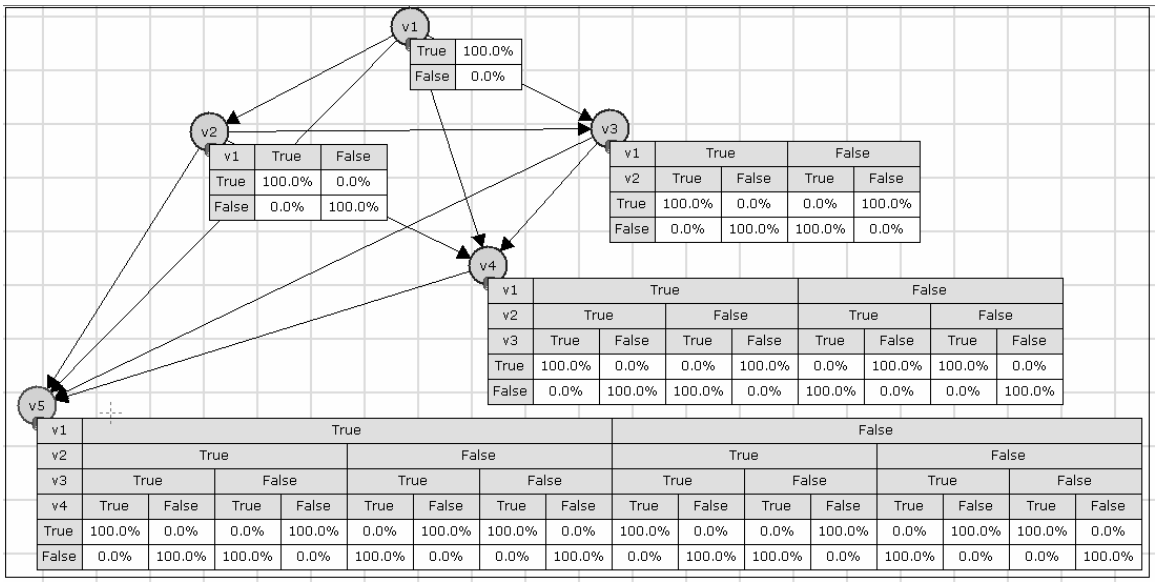

Figure 16. Pathological example for reduction to polytree

Consider the example network in Figure 16. Notice that this network is a complete graph, with edges between each pair of nodes v_i and v_j, where $i < j$. Also note that the conditional probability tables are constructed so that if any edge is deleted, the entries that would need to be averaged are either 100% or 0% -- so that the resulting average will be as close to 50% (and as far away from the original probabilities) as possible.

When the reduction to polytree algorithm is run on this network, it deletes the edges (v_2, v_3), (v_2, v_4), (v_3, v_4), (v_3, v_5), (v_4, v_5), and (v_1, v_5) when turning the network into a polytree. The KL divergence between the original probability distribution for the network and the approximate probability distribution that results from deleting that set of edges is 2.08, using the formula in Section 1.4.1. Thus according to the error bound computed above, the RMSE should be no more than 1.02. An RMSE value is generally considered acceptable if it is below 0.1, so this error bound suggests that the posterior probabilities computed for this network by reduction to polytree will not be very accurate. Indeed, the actual RMSE produced by the algorithm is 0.387 – making this particular approximation unusable in practice.

4.2 Bounding Clique Sizes Techniques

While reduction to polytree picks edges for deletion with minimal KL divergence values, both bounding clique sizes algorithms (bounding clique sizes with pre-processing and general bounding clique sizes) choose edges for deletion that maximize the number of moral edges rendered unnecessary. Because the score for eligible edges is based on structural properties instead of on information loss, no error bound can be placed on the

bounding clique sizes techniques. However, because both bounding clique sizes algorithms ignore information loss in favor of removing more moral edges, they can make the mistake of deleting a single edge that is very important to the overall probability distribution just because its deletion causes several moral edges to become unnecessary.

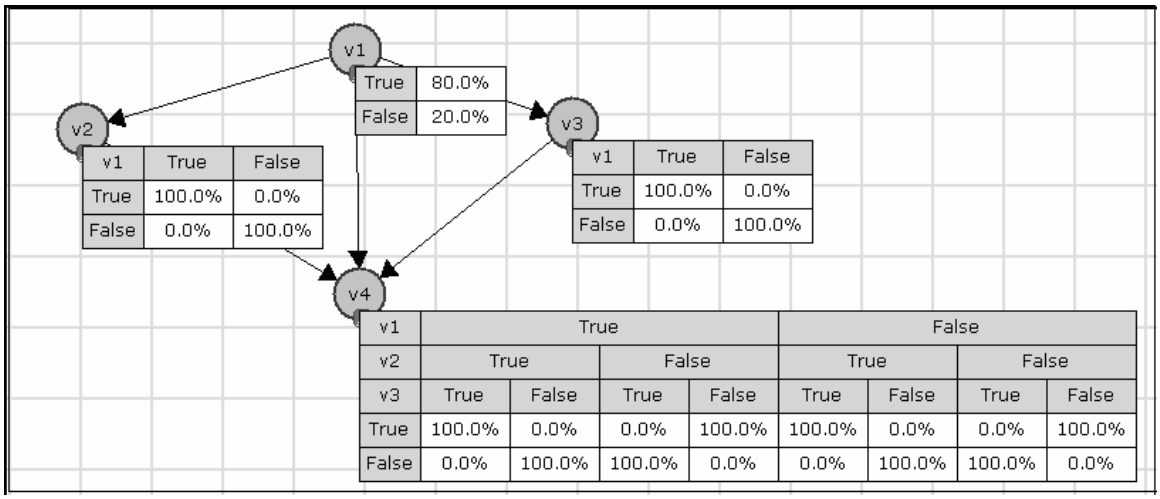

Figure 17. Pathological example for both bounding clique sizes algorithms

Consider the example network in Figure 17. Note that because nodes v_2 and v_3 are co-parents of node v_4, a moral edge would need to be added between nodes v_2 and v_3, thus making the network a complete four-node graph. First, assume that this network was given to the general bounding clique sizes algorithm with a desired clique bound of three. The algorithm would only need to delete one edge from the network to satisfy the desired clique bound. However, when examining eligible edges for deletion, the general bounding clique sizes would notice that edges (v_2, v_4) and (v_3, v_4) would both cause moral edge (v_2, v_3) to be unnecessary, while edge (v_1, v_4) would not cause any moral edges to be unnecessary. (Edges (v_1, v_2) and (v_1, v_3) would not be eligible since they serve as moral

edges.) If the algorithm chose edge (v_2, v_4) for removal, however, the resulting RMSE is a poor 0.345.

Note in Figure 17 that node v_4 is conditionally independent of its parent v_1, because $P(v_4=T| v_1, v_2, v_3) = P(v_4=T| v_2, v_3)$. Thus if we had instead deleted the edge (v_1, v_3) to reduce the maximum clique size to three, no information would have been lost and the RMSE would have been 0.0. Thus the general bounding clique size's strategy of always removing the edge that renders the most moral edges unnecessary is not always wise – especially when the network contains edges with no information.

While this example was designed for the general bounding clique sizes algorithm, the result would have been the same for the bounding clique sizes with pre-processing algorithm. If the network in Figure 17 was the reduced network after running the pre-processing rules in the bounding clique sizes with pre-processing algorithm, the algorithm would have been trying to delete edges from that reduced network so that it could continue applying the simplicial vertex rule. The bounding clique sizes with pre-processing algorithm also choosing edges for deletion based on which edge's removal will maximize the number of unnecessary moral edges, and so it would also have chosen edge (v_2, v_4) for deletion instead of the extraneous edge (v_1, v_3). Thus this algorithm would also have unnecessarily increased the inference error.

CHAPTER FIVE: Results

This chapter contains the results of running the reduction to polytree algorithm, bounding clique sizes with pre-processing algorithm, the general bounding clique sizes algorithm (with clique bounds 4, 5, 6, 8, 10, and 12), LS, and AIS (with 2,000 samples) on each of the 10 Bayesian networks described in Table 3. Comparisons of the total time, inference time, and error (using RMSE measurement) are provided for each algorithm, according to the experimental design specification provided in Section 3.5.3. Interpretation of these results appear in Chapter 6.

In these experiments, each algorithm was run five times on each network. The total times and inference times reported for each algorithm are the average of those five runs. The RMSE values for the three new approximate inference algorithms (reduction to polytree and the two different bounding clique sizes methods) are not averaged over the five runs, because these algorithms have no randomness and will execute in exactly the same way every time they are run. Consequently, they will produce identical posterior probabilities with every run. The RMSE values reported for the AIS algorithm, however, are averaged over five runs, because AIS randomly samples the network many times to learn the posterior probabilities. In the graphs and tables in this section, I use the abbreviation “RP” for reduction to polytree, “BCS-P” for bounding clique sizes with pre-processing, and “G-BCS” for general bounding clique sizes.

Each experimental run was performed on a PC with a 3.0 GHz Pentium 4 processor with 1 GB of RAM. A maximum heap size of 1 GB was allocated for the Java Virtual Machine during each experiment. The total times and inference times recorded below were measured using the Java current time method.

5.1 Total Time Comparisons

The total time of running an inference algorithm on a Bayesian network includes any pre-processing time (such as choosing edges for deletion or building the junction tree) as well as the time for inference. In the case of sampling algorithm such as AIS, the total time is the same as the inference time because there is no pre-processing step. Total time measurements are important because they measure the amount of time needed to perform the entire process. However, because the pre-processing stage needs to be run only once before running the inference portion multiple times with different evidence values, the total time measurement is not as helpful as the inference time in determining how efficient an algorithm will be in practice. This section provides average total time measurements for LS, AIS, and each of the three new inference algorithms (including all six possible clique bounds for the general bounding cliques algorithm).

Figure 18 shows a comparison of the average total times required for LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes with a bound of five on each of the ten networks described in Table 3. Note that the scale for the total time required in each instance is logarithmic. The percentage of standard deviation of the total time over each of the five runs appears in Table 4. This table reports standard deviation values for the total time of LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes with a bound of five. The percentage standard deviation is calculated by taking the standard deviation of the total time for the five runs divided by the average time, and then converting to a percent.

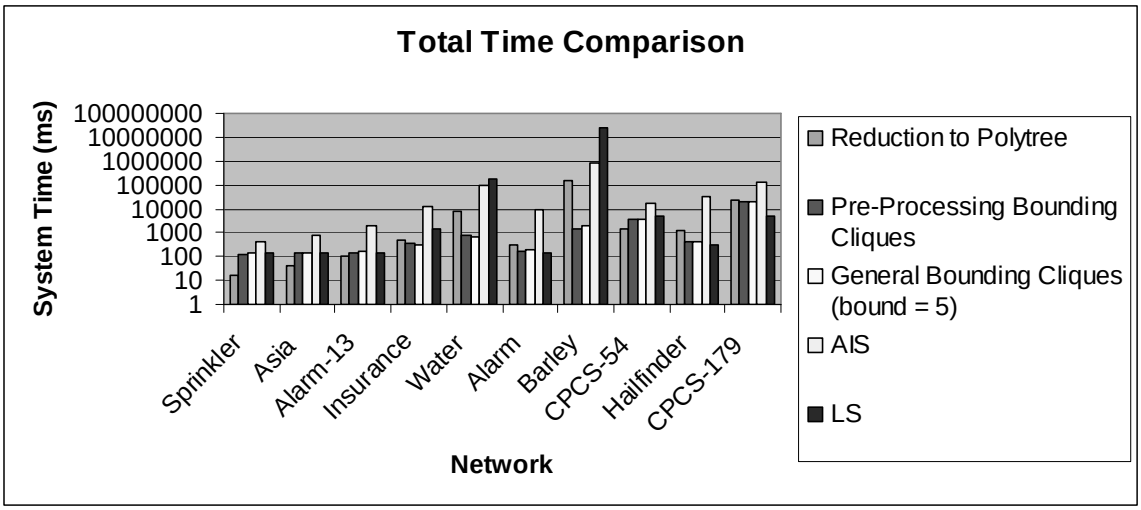

Figure 18. Total time comparison between all algorithms. Time scale is logarithmic, and lower is better.

Network	LS	RP	BCS-P	G- BCS (bound = 5)	AIS
Sprinkler	6.53%	3.51%	5.24%	4.98%	1.74%
Asia	5.06%	15.25%	6.27%	0.0%	2.84%
Alarm-13	5.19%	0.0%	5.06%	4.28%	4.88%
Insurance	1.13%	3.65%	2.40%	4.94%	4.68%
Water	0.16%	0.61%	1.79%	3.33%	5.12%
Alarm	5.77%	2.48%	4.24%	2.20%	3.46%
Barley	1.25%	0.70%	0.59%	3.29%	19.7%
CPCS-54	0.90%	0.92%	0.60%	0.83%	9.42%
Hailfinder	2.03%	1.28%	0.0%	2.97%	4.65%
CPCS-179	0.68%	0.25%	1.73%	1.29%	5.37%

Table 4. Standard deviation percentage of total times for all algorithms (stddev/avg time*100)

Figure 19 shows a comparison of the average total time required for the general bounding clique sizes algorithm using the different clique bounds of 4, 5, 6, 8, 10, and 12. Recall that the general bounding clique sizes algorithm is given an initial perfect elimination scheme for the network and then deletes edges so that the given node ordering obeys the desired clique bound. In my implementation, I used maximum cardinality search (MCS) to generate the initial perfect elimination scheme. The

maximum clique sizes for each network using MCS are given in Table 3. For some of the executions of the general bounding clique sizes algorithm, the maximum clique size of the initial triangulation is already smaller than the desired bound. In these cases, the total running time is about the same as the total running time for LS. Note that the system time scale for this graph is also logarithmic.

The percentage of standard deviation of the total time over each of the five runs appears in Table 5. This table reports standard deviation values for the total time of the general bounding clique sizes algorithm with desired clique bounds of 4, 5, 6, 8, 10, and 12. The percentage standard deviation is calculated by taking the standard deviation of the total time for the five runs divided by the average time, and then converting to a percent. In this table, the notation *b=i* refers to a run of the general bounding clique sizes algorithm with a desired clique bound of *i*.

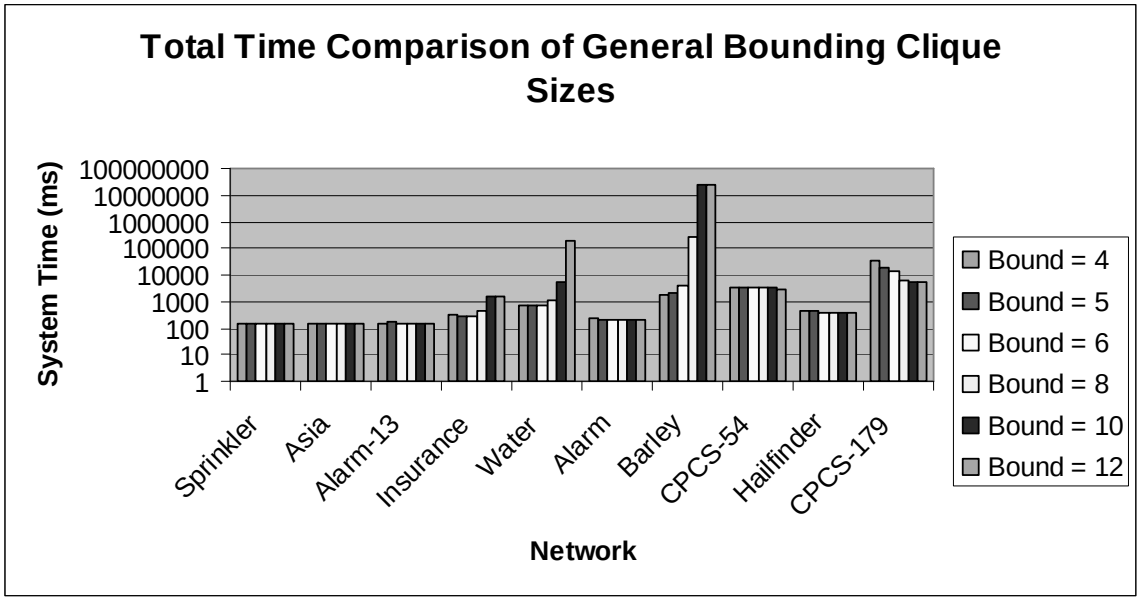

Figure 19. Total time comparison of general bounding clique sizes with different desired clique bounds. Time scale is logarithmic, and lower is better.

Network	$b = 4$	$b = 5$	$b = 6$	$b = 8$	$b = 10$	$b = 12$
Sprinkler	0.39%	4.98%	0.39%	4.98%	0.0%	5.13%
Asia	0.39%	0.0%	4.66%	4.75%	5.99%	0.32%
Alarm-13	0.35%	4.28%	4.49%	4.46%	0.29%	0.29%
Insurance	2.24%	4.94%	2.66%	1.65%	0.87%	0.75%
Water	3.40%	3.33%	1.00%	1.23%	0.26%	1.22%
Alarm	3.28%	2.20%	3.58%	4.32%	0.22%	0.0%
Barley	1.19%	3.29%	1.49%	0.87%	0.76%	0.21%
CPCS-54	1.14%	0.83%	0.93%	0.90%	0.81%	0.39%
Hailfinder	1.86%	2.97%	5.47%	1.93%	1.89%	6.14%
CPCS-179	0.09%	1.29%	1.24%	1.95%	1.80%	5.09%

Table 5. Standard deviation percentage of total times for general bounding clique sizes (stddev/avg time*100)

5.2 Inference Time Comparisons

The "inference time" for running an inference algorithm is the time required to compute the posterior probabilities for each node, excluding any pre-processing steps that can be reused for multiple executions of the inference algorithm. For example, construction of the junction tree only needs to be done once, and then the message-passing phase to compute the posterior probabilities can be repeated for multiple runs of the inference algorithm. Note that the message-passing does need to be repeated, because if the algorithm is run with different evidence values, the values of the messages will be different. Evidence values do not, however, have any effect on pre-processing steps.

This section reports comparisons between inference time for LS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes (with each of the six different clique bounds) when run on each of the ten networks in Table 3. For each algorithm, the inference time does not include choosing which edges to delete or building the junction tree. The inference time does include the message-

passing phase of Pearl's algorithm (for the reduction to polytree technique) and of LS (for each of the other algorithms). Note that AIS is not included in these comparisons because its total time is the same as its inference time, as it has no pre-processing step.

Figure 20 shows a comparison of the average inference times required for LS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes with a clique bound of five on each of the ten networks described in Table 3. Note that the scale for the total time required in each instance is logarithmic. The percentage of standard deviation of the inference time over each of the five runs appears in Table 6. This table reports standard deviation values for the inference time of LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes with a bound of five. The percentage standard deviation is calculated by taking the standard deviation of the inference time for the five runs divided by the average time, and then converting to a percent.

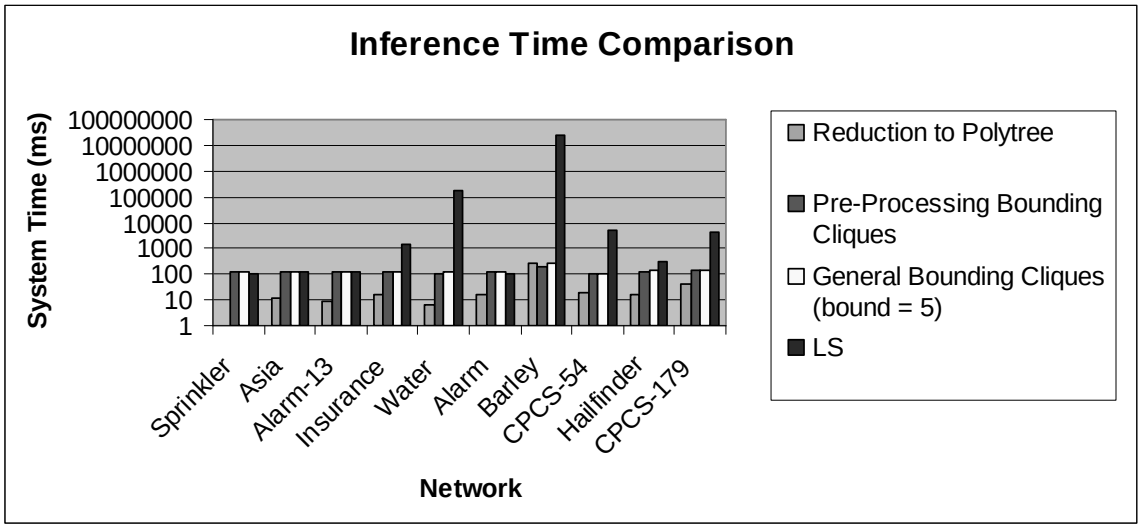

Figure 20. Inference time comparisons between all algorithms. Time scale is logarithmic, and lower is better.

Network	LS	RP	BCS-P	G- BCS (bound = 5)	AIS
Sprinkler	0.50%	0.0%	0.0%	5.87%	1.74%
Asia	5.67%	56.0%	5.87%	5.50%	2.84%
Alarm-13	11.7%	91.4%	7.15%	5.58%	4.88%
Insurance	1.19%	3.51%	6.28%	14.86%	4.68%
Water	0.16%	137%	6.38%	0.0%	5.12%
Alarm	14.4%	3.56%	0.0%	6.48%	3.46%
Barley	1.25%	2.72%	3.41%	3.99%	19.7%
CPCS-54	1.06%	38.2%	8.54%	6.38%	9.42%
Hailfinder	2.96%	3.51%	0.0%	4.64%	4.65%
CPCS-179	0.89%	16.3%	7.79%	0.26%	5.37%

Table 6. Standard deviation ratio of inference times for all algorithms (stddev/avg time*100)

Figure 21 below shows the inference time required for the general bounding clique sizes algorithm using the different clique bounds of 4, 5, 6, 8, 10, and 12. Again, remember that the general bounding clique sizes algorithm is supplied with the perfect elimination scheme from MCS, and then deletes edges from the original network so that the given node ordering will satisfy the required clique bound. The maximum clique sizes for each network using MCS are provided in Table 3 – in some cases, these values are smaller than the desired clique bound, so the resulting inference time will be similar to the inference time for LS.

The percentage of standard deviation of the inference time over each of the five runs appears in Table 7. This table reports standard deviation values for the inference time of the general bounding clique sizes algorithm with desired clique bounds of 4, 5, 6, 8, 10, and 12. The percentage standard deviation is calculated by taking the standard deviation of the inference time for the five runs divided by the average time, and then

converting to a percent. In this table, the notation *b=i* refers to a run of the general bounding clique sizes algorithm with a desired clique bound of *i*.

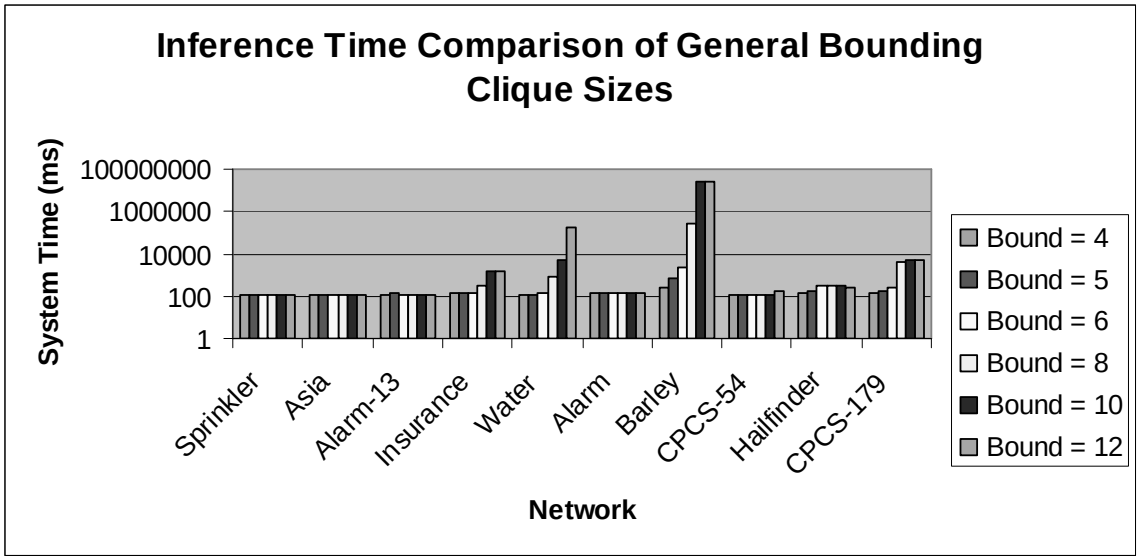

Figure 21. Inference time comparison of general bounding clique sizes with different desired clique bounds. Time scale is logarithmic, and lower is better.

Network	***b* = 4**	***b* = 5**	***b* = 6**	***b* = 8**	***b* = 10**	***b* = 12**
Sprinkler	0.0%	5.87%	5.50%	5.87%	0.0%	5.50%
Asia	5.87%	5.50%	5.87%	0.0%	0.0%	0.0%
Alarm-13	0.0%	5.58%	0.0%	0.0%	0.0%	0.0%
Insurance	5.58%	14.9%	0.39%	3.51%	0.92%	0.75%
Water	5.50%	0.0%	4.46%	1.00%	0.23%	1.23%
Alarm	5.58%	6.48%	5.13%	6.27%	5.58%	5.19%
Barley	2.65%	3.99%	2.32%	0.86%	0.76%	0.21%
CPCS-54	6.87%	6.38%	0.41%	7.59%	0.0%	3.77%
Hailfinder	4.98%	4.64%	2.99%	3.05%	3.05%	3.90%
CPCS-179	6.39%	0.26%	3.57%	1.35%	1.41%	1.47%

Table 7. Standard deviation ratio of inference times for general bounding clique sizes (stddev/avg time*100)

While measuring inference time is very important because it tells how long an algorithm will take to report the posterior probabilities in a traditional execution, it does not provide a completely accurate comparison between different inference techniques

because it ignores any pre-processing steps the algorithm might make. While it is true that the pre-processing phase only needs to be done once, it could be that the pre-processing phase takes several days (or even years, for a large network) to complete – which would make the algorithm unusable even if the inference stage was quite fast. Because of this, it is also important to consider the amount of time required to compute *n* runs for the algorithm, adding the pre-processing time to *n* times the inference time to get the total time for *n* runs. Figure 22 shows the amount of time required for each algorithm – LS, AIS (with 2,000 samples), reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes with a bound of five – to run each network 100 times. If we let *IT* be the inference time for an algorithm and *TT* be the total time for an algorithm, then the amount of time required to run the algorithm 100 times would be

$$(TT - IT) + 100*IT.$$

Again, remember that AIS has no pre-processing step, so that $TT = IT$. Also, note that the system time scale is logarithmic.

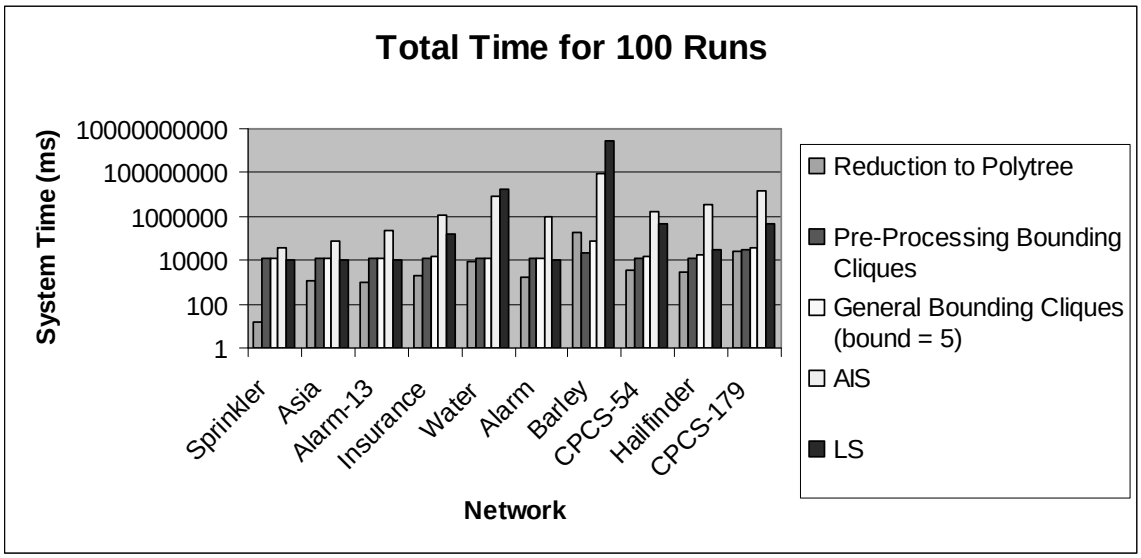

Figure 22. Comparison of total time for 100 runs of each algorithm. Time scale is logarithmic, and lower is better.

Figure 23 below shows a comparison between the time required for 100 runs on each network using the general bounding clique sizes algorithm with each of the different clique bounds: 4, 5, 6, 8, 10, and 12. Again, the system time scale is logarithmic.

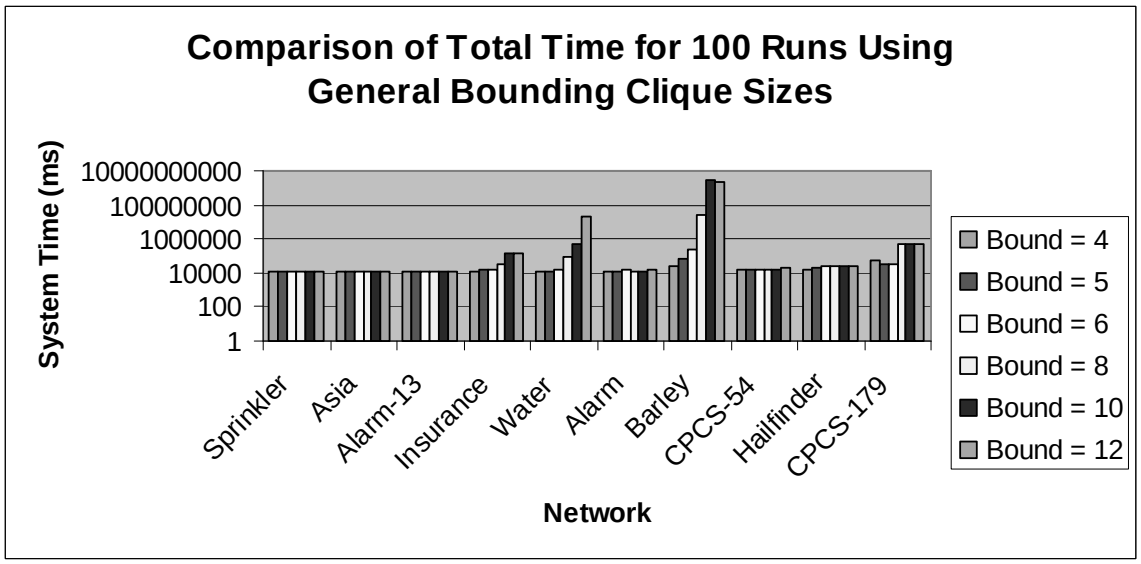

Figure 23. Total time for 100 runs using general bounding clique sizes with different desired clique bounds. Time scale is logarithmic, and lower is better.

5.3 Root-Mean-Squared Error Comparisons

The purpose of gathering root-mean-squared error (RMSE) metrics is to see if an approximate inference algorithm can produce a sufficiently accurate estimate of the posterior probabilities, as compared to the exact posterior probabilities produced by an algorithm such as LS. RMSE statistics were gathered for each network in Table 3 for the reduction to polytree algorithm, the bounding clique sizes with pre-processing algorithm, the general bounding clique sizes algorithm (with maximum clique bounds of 4, 5, 6, 8, 10, and 12), and AIS with 2,000 samples. In each case, the RMSE was computed using the formula in Section 3.5.3 by comparing the posterior probabilities of the given approximate inference algorithm to the posterior probabilities computed by LS.

Table 8 below shows the RMSE values computed for the reduction to polytree algorithm, the bounding clique sizes with pre-processing algorithm, the general bounding clique sizes algorithm with a maximum clique bound of five, and AIS. The AIS error values include the percentage of standard deviation over the five runs, computed by stddev/avg RMSE * 100. The other algorithms do not have standard deviation values reported because they have no random element and therefore perform exactly the same way every time.

Network	**RP**	**BCS -P**	**G-BCS (bound = 5)**	**AIS**
Sprinkler	0.02430	0.0	0.0	0.01022 +/- 40.3%
Asia	0.00118	0.0	0.0	0.00785 +/- 22.9%
Alarm-13	0.03125	0.0	0.0	0.00744 +/- 46.7%
Insurance	0.05497	0.08297	0.03267	0.00921 +/- 3.39%
Water	0.02098	0.17720	0.19147	0.00588 +/- 10.6%
Alarm	0.18992	0.0	0.0	0.01978 +/- 15.4%

Barley	0.01637	0.02213	0.01870	0.12508 +/- 0.41%
CPCS-54	0.03032	0.03098	0.02889	0.03010 +/- 6.43%
Hailfinder	0.00891	0.02196	0.00797	0.00947 +/- 3.05%
CPCS-179	0.09519	0.13863	0.08954	0.00508 +/- 14.9%

Table 8. RMSE comparison of all approximate algorithms. Lower is better.

The general bounding clique sizes algorithm was executed on each network with six different inputs for the bound for the maximum clique: 4, 5, 6, 8, 10, and 12. Table 9 below shows the RMSE values for the general bounding clique sizes algorithm for each network in Table 3 and for each of the six possible clique bounds. The inputted clique bound is abbreviated by *b* in the table.

Network	***b* = 4**	***b* = 5**	***b* = 6**	***b* = 8**	***b* = 10**	***b* = 12**
Sprinkler	0.0	0.0	0.0	0.0	0.0	0.0
Asia	0.0	0.0	0.0	0.0	0.0	0.0
Alarm-13	0.0	0.0	0.0	0.0	0.0	0.0
Insurance	0.04163	0.03267	0.00858	0.00499	0.0	0.0
Water	0.22122	0.19147	0.19136	0.00246	0.00245	0.0
Alarm	0.00379	0.0	0.0	0.0	0.0	0.0
Barley	0.01962	0.01870	0.03028	0.01075	0.0	0.0
CPCS-54	0.02904	0.02889	0.02875	0.02896	0.02898	0.02968
Hailfinder	0.01468	0.00797	0.0	0.0	0.0	0.0
CPCS-179	0.10694	0.08954	0.06093	0.00041	0.0	0.0

Table 9. RMSE for general bounding clique sizes with different clique bounds. Lower is better.

5.4 Master Table of Results

The previous sections in this chapter presented results for the total time, inference time, and error values for reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes. They also compared the three new algorithms to the existing exact inference algorithm LS and approximate inference algorithm AIS. Because evaluating the efficiency/accuracy tradeoff is difficult when

looking between different charts on time and error values, this section will contain a "master table" of all time and accuracy values for each algorithm.

Total Time										
BN	**LS**	**AIS**	**RP**	**BCSP**	***b*=4**	***b*=5**	***b*=6**	***b*=8**	***b*=10**	***b*=12**
Spr	135	394	16	128	141	137	140	137	141	138
Asia	138	722	44	134	141	141	144	144	146	141
A13	138	2122	109	137	156	160	159	153	156	156
Ins	1487	1.2e4	472	365	300	288	278	435	1519	1516
Wtr	1.9e5	8.9e4	7950	722	734	709	716	1166	5044	1.9e5
Alm	147	9519	284	169	222	203	200	197	203	203
Brly	2.6e7	8.4e5	1.5e5	1475	1684	2084	3731	2.5e5	2.6e7	2.6e7
C54	5091	1.7e4	1538	3388	3550	3462	3359	3250	3216	2906
Hail	331	3.3e4	1234	522	462	434	384	372	378	385
C179	4653	1.4e5	2.3e4	1.9e4	3.5e4	1.9e4	1.3e4	5906	5450	5616

Inference Time										
BN	**LS**	**AIS**	**RP**	**BCSP**	***b*=4**	***b*=5**	***b*=6**	***b*=8**	***b*=10**	***b*=12**
Spr	109	394	0	125	125	122	122	122	125	122
Asia	113	722	12	122	122	122	122	125	125	125
A13	113	2122	9	119	125	128	125	125	125	125
Ins	1459	1.2e4	16	112	128	138	141	313	1456	1453
Wtr	1.9e5	8.9e4	6	112	122	125	153	837	4750	1.9e5
Alm	110	9519	15	125	128	131	138	131	128	138
Brly	2.6e7	8.4e5	247	200	253	729	2469	2.5e5	2.6e7	2.6e7
C54	5053	1.7e4	19	103	106	112	109	115	125	184
Hail	287	3.3e4	16	125	137	181	288	287	287	281
C179	4528	1.4e5	44	141	134	172	241	4475	4628	4669

Ratio of Total Time to LS Total Time										
BN	**LS**	**AIS**	**RP**	**BCSP**	***b*=4**	***b*=5**	***b*=6**	***b*=8**	***b*=10**	***b*=12**
Spr	1.0	2.93	0.12	0.95	1.04	1.02	1.04	1.02	1.05	1.02
Asia	1.0	5.25	0.32	0.98	1.02	1.03	1.05	1.05	1.07	1.02
A13	1.0	15.4	0.79	1.0	1.13	1.16	1.16	1.11	1.13	1.13
Ins	1.0	7.94	0.32	0.25	0.20	0.19	0.19	0.29	1.02	1.02
Wtr	1.0	0.48	0.04	0.004	0.004	0.004	0.004	0.006	0.03	1.01
Alm	1.0	64.7	1.93	1.15	1.51	1.38	1.36	1.34	1.38	1.38
Brly	1.0	0.03	0.006	5.7e-5	6.5e-5	8e-5	1.4e-4	0.01	0.99	0.99
C54	1.0	3.27	0.30	0.67	0.70	0.68	0.66	0.64	0.63	0.57
Hail	1.0	98.5	3.73	1.27	1.40	1.31	1.16	1.12	1.14	1.16
C179	1.0	29.1	4.97	4.14	7.55	4.06	2.83	1.27	1.17	1.21

RMSE										
BN	**LS**	**AIS**	**RP**	**BCSP**	***b*=4**	***b*=5**	***b*=6**	***b*=8**	***b*=10**	***b*=12**
Spr	0	.010	.024	0	0	0	0	0	0	0
Asia	0	.008	.001	0	0	0	0	0	0	0
A13	0	.007	.031	0	0	0	0	0	0	0
Ins	0	.009	.055	.083	.042	.033	.009	.005	0	0
Wtr	0	.006	.021	.177	.221	.19	.191	.002	.002	0
Alm	0	.020	.190	0	.004	0	0	0	0	0

Brly	0	.125	.016	.022	.020	.019	.030	.011	0	0
C54	0	.030	.030	.031	.029	.029	.029	.029	.030	.03
Hail	0	.009	.009	.022	.015	.008	0	0	0	0
C179	0	.005	.095	.139	.107	.090	.061	.0004	0	0

Table 10. Master table with total time, inference time, time for 100 runs, and RMSE for all algorithms

Table 10 contains results for the total time, inference time, ratio of total time to LS total time, and RMSE for LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes (with desired clique bounds 4, 5, 6, 8, 10, and 12). The ratio of total time to LS total time for an algorithm is the algorithm's total time divided by LS's total time. This ratio will be close to zero if the algorithm is much faster than LS, close to one if the algorithm takes about the same amount of time as LS, and bigger than one if the algorithm is slower than LS. Note that this table uses the abbreviation "*b*=*i*" for the general bounding clique sizes algorithm with a desired clique bound of *i*. Finally, the names of each network are shortened to allow this table to fit on the page. For the full names of each network, see Table 3. Each time and RMSE value in the table is averaged over five runs, and all time values are in milliseconds. Essentially, this table combines the results in Figures 18-21 and Tables 8 and 9.

CHAPTER SIX: Discussion and Future Work

Chapter 5 presented results on the speed and accuracy comparisons between reduction to polytree, bounding clique sizes with pre-processing, general bounding clique sizes (with desired clique bounds 4, 5, 6, 8, 10, and 12), LS, and AIS (with 2,000 samples). The total time, inference time, and time required for 100 runs were compared for each algorithm. This chapter interprets each chart and table of speed and accuracy comparisons, and discusses what those results say about the usefulness of the three new algorithms. The chapter closes by summarizing the contents of this thesis and then discussing what future research is motivated by the results described here.

6.1 Interpretation of Results

This subsection will comment on each chart and graph of the speed and accuracy results presented in Chapter 5.

6.1.1 Time Results Discussion

This section contains commentary on the speed results from Chapter 5. Specifically, it discusses the total time comparison, inference time comparison, and time for 100 runs for each algorithm.

Total Time Discussion:

Consider first the graph for total time comparison between LS, AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes (with a clique bound of 5) in Figure 18. The total time for reduction to polytree and both bounding clique sizes techniques is less than that of AIS for every network. In some networks, the difference is striking. For Water, reduction to polytree finishes in 9% of

the time for AIS, and both bounding clique sizes techniques finish in each of the three new algorithms finish in 0.8% or less of the time it takes AIS. For Barley, reduction to polytree finishes in 18% of the time needed for AIS and the two bounding clique sizes techniques finish in 0.2% or less of the time for AIS.

Unlike for AIS, the three new algorithms do not always perform faster than LS. For Hailfinder and CPCS-179, LS performs faster than all three algorithms. Though these algorithms have the largest number of nodes of any of the networks tested, they are not nearly as complex as the Water and Barley networks. Even on these networks, LS does not perform more than five times faster than the three new algorithms, and each of the new algorithms still finish in 25 seconds or less. By comparison, reduction to polytree finishes in 4% of the time required for LS on the Water network, and the two bounding clique sizes algorithms finish in 0.4% or less of the time for LS on Water. Furthermore, LS requires 7.5 hours to run the Barley network, while the reduction to polytree technique finishes in 0.6% of that time and the two bounding clique sizes techniques only require 0.008% of the LS time.

Total Time for General Bounding Clique Sizes Discussion:

Next, consider the total time comparison of the general bounding clique sizes algorithm (for clique bounds 4, 5, 6, 8, 10, and 12) in Figure 19. Notice that for some networks (namely, Sprinkler, Asia, Alarm-13, Alarm, CPCS-54 and Hailfinder), there is little difference between the total times with different clique bounds. This is because the initial maximum clique size using MCS for these networks was already low (see Table

3), and so the general bounding clique sizes algorithm did not need to delete more edges as the clique bound got smaller.

For Water and Barley, however, there is a significant difference between the time required for a clique bound of eight and the time required for a clique bound of 10 or 12. For Barley, the time required for a clique bound of eight is 1% the time required for a clique bound of 10. For Water, the time required for a clique bound of eight is 23% the time required for a clique bound of 10 and 0.6% of the time required for a clique bound of 12. For both the Water and Barley networks, the chart in Figure 19 shows how the total time is exponential in the size of the maximum clique.

Inference Time Discussion:

Consider the inference time comparison between reduction to polytree, bounding clique sizes with pre-processing, general bounding clique sizes (with a clique bound of five), and LS in Figure 20. (Recall that AIS was not included in this comparison because it did not have a separate inference phase.) In this chart, LS is either slower or equivalent to the three new algorithms on all networks. The difference is again most notable on the Water and Barley networks. For the Water network, reduction to polytree finishes in 0.003% of the time required for LS, and the two bounding clique sizes techniques finish in at most 0.06% of the time for LS. For Barley, all three algorithms finish in at most 0.003% of the time required for LS. While LS takes 7.5 hours to complete inference on Barley, the three new algorithms take at most 7 seconds.

Inference Time for General Bounding Clique Sizes Discussion:

The comparison of the inference time required for the general bounding clique sizes algorithm (with different clique bounds 4, 5, 6, 8, 10, and 12) appears in Figure 21. Again, notices that several networks have no difference in inference time between different clique bounds. This is because the initial maximum clique size provided to the algorithm by MCS was already close to the minimum clique bound of four – hence the algorithm performs similarly on different clique bounds for those networks. However, notice that Insurance, Water, Barley, and CPCS-179 all have noticeable differences between the inference time with a clique bound of four and a clique bound of 12.

For the Water and Barley networks, the general bounding clique sizes algorithm still performs significantly faster than LS even with a larger desired clique bound (see Figure 20 for the inference time of LS). For Water, the algorithm with a clique bound of eight still completes inference in 0.45% of the time required for LS; for Barley, general bounding clique sizes with a clique bound of eight completes inference in 0.96% of the time for LS.

100 Runs Time Discussion:

Consider the time values for 100 runs of reduction to polytree, bounding clique sizes with pre-processing, general bounding clique sizes (with a clique bound of five), LS, and AIS presented in Figure 22. Recall that the time to run an inference algorithm 100 times is computed by adding the pre-processing time to 100 times the inference time. As was true for inference time, the three new algorithms are faster than or equivalent to AIS and LS over 100 runs for every network. Again, Water and Barley show the most

significant differences between the techniques. On Water, the new algorithms complete 100 runs in at most 0.07% of the time needed for LS, and at most 0.15% of the time needed for AIS. For Barley, each of the three algorithms completes 100 runs in at most 0.0009% of the time needed for LS, and at most 0.2% of the time needed for AIS.

100 Runs Time for General Bounding Clique Sizes Discussion:

Finally, consider the chart in Figure 23 that compares the time for 100 runs of the general bounding clique sizes algorithm (with desired clique bounds 4, 5, 6, 8, 10, and 12). Notice that for the networks Insurance, Water, Barley, and CPCS-179, there is a significant difference between the time for 100 runs at a lower clique size to the time at a higher clique size. However, even a more generous clique bound can produce savings over LS (see Figure 22 for the time to run LS 100 times on different networks). For Water, the time for 100 runs with a clique bound of eight is 0.45% the time needed for LS. For Barley, the time for 100 runs with a clique bound of eight is 0.96% the time needed for LS.

6.1.2 Accuracy Results Discussion

This section contains commentary on the accuracy results from Chapter 5. It discusses the RMSE values shown in Chapter 5 for each approximate inference algorithm – AIS, reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes.

RMSE Discussion:

Consider the RMSE values for the approximate inference algorithms reduction to polytree, bounding clique sizes with pre-processing, general bounding clique sizes (with

a clique bound of five), and AIS presented in Table 8. The first notable characteristic of the RMSE values is that AIS does produce smaller or similar error values those for all three of the new algorithms on all networks except Barley. However, AIS is between three and 500 times slower (and usually closer to the higher end) than the three new algorithms on every network, and the error values for the three new algorithms are generally no more than ten times higher than those of AIS.

Furthermore, the reduction to polytree algorithm produces a poor RMSE value of 0.1 or higher on only one out of ten networks (Alarm). The general bounding clique sizes also only produces a poor RMSE value on one network (Water), and the bounding clique sizes with pre-processing algorithm has poor accuracy on only two networks (Water and CPCS-179). Perhaps more importantly, the RMSE values for Barley are quite low (0.022 or lower) for each of the three new algorithms, while AIS produces a poor error value of 0.125. Recall that the two bounding clique sizes techniques compute inference for Barley in 0.2% of the time for AIS and in 0.008% of the time for LS.

Lastly, note that several RMSE values are 0.0 for the two bounding clique sizes techniques. This occurs when the maximum clique size in the original MCS triangulation is less than the desired clique bound of five. In this case, no edges need to be deleted to reach the bound, so the bounding clique sizes algorithms perform just like LS.

RMSE for General Bounding Clique Sizes Discussion:

Finally, consider the RMSE values for the general bounding clique sizes algorithms (with clique bounds of 4, 5, 6, 8, 10, and 12) in Table 9. Most of the results in this table are unsurprising – generally, as the allowed clique size increases, the RMSE

slightly decreases. However, the one interesting network in this table is Water. Recall that Water was the only network on which the general bounding clique sizes algorithm (with a clique bound of five) was fairly inaccurate – with an RMSE of 0.19. This algorithm continues to perform inaccurately with a clique bound of six, but the RMSE drops to a very low 0.00246 with a clique bound of eight. Furthermore, running this algorithm with a clique bound of eight on Water still takes 0.6% of the time for LS.

6.1.3 Standard Deviation Discussion

Consider Tables 4-7, which report the percent standard deviation for the average total time and average inference time over five runs of LS, AIS, bounding clique sizes with pre-processing, and general bounding clique sizes (with desired clique bounds of 4, 5, 6, 8, 10, and 12). These results were reported simply to show that the standard deviation of algorithm time was disinteresting. Indeed, because every algorithm (including the three new ones) except AIS has no random element and hence runs exactly the same way every time, any variance at all in system time would not be a result of the algorithm. These tables do report some high percent standard deviation values for the algorithms (such as 137% for the inference time of reduction to polytree on the Water network), but these high values are a result of the Java current time measurement not being accurate enough. In the case of reduction to polytree on the Water network, inference time for each of the five runs was either 15 ms or 0 ms. This is because the Java current time measurement is unable to accurately measure such a brief period of time.

The percent standard deviation values for AIS are a bit more interesting, because AIS does perform differently each time it is run (due to the random sampling step).

However, AIS has relatively low percent standard deviation values (less than 6%) for time on most networks. On the two networks it has slightly higher percent standard deviation values (Alarm-13 and Alarm), AIS finished in an average time of 113 ms. Since this is still a relatively low total time, the higher standard deviation can again be blamed on the coarse measurements of Java current time.

6.2 Conclusion and Future Work

In conclusion, the reduction to polytree, bounding clique sizes with pre-processing, and general bounding clique sizes algorithms showed promising results. They were no slower than either LS or AIS in the 100 run test on all networks tested, and ran up to 10,000 times faster for a single run on the more complex networks. Furthermore, the error introduced by these two techniques was minimal – reduction to polytree and general bounding clique sizes had RMSE values under 0.1 for nine out of ten networks, and bounding clique sizes with pre-processing had low RMSE values for eight out of ten networks. Furthermore, the general bounding clique sizes algorithm had RMSE values below 0.1 for all ten networks if the appropriate maximum clique size bound was used.

The results of the general bounding clique sizes algorithm when run with different desired clique bounds showed that even when the desired clique bound got as high as eight, its time for 100 runs was no slower than LS and again, often much faster. For example, Water still performed 160 times faster than LS with a clique bound of eight, but had a much lower RMSE value when allowed the more relaxed clique bound. It appears that even a slight reduction in the original maximum clique size for a network can still yield a tremendous speedup from LS.

As future research, one could examine how to predict which clique bound to give the bounding clique size algorithm, based on network properties, that would yield the best compromise between speedup and introduced error. As part of researching these network properties, one might develop a "complexity metric" for a network that takes into account properties such as number of nodes, number of edges, size of probability tables, and other factors to determine how complex a network is. While exact inference algorithms such as LS are exponential in the size of the maximum clique, the Barley network (with 48 nodes and a maximum clique size of 8) takes much longer to run than CPCS-54 (with 54 nodes and a maximum clique size of 18). A complexity score could help to appropriately determine how hard inference is for a particular network, and could be the first step in determining the best clique bound. It would also be useful to compare the three new algorithms presented here with a larger set of exact and approximate inference algorithms, including several different implementations of LS when triangulated with optimized triangulation algorithms. Finally, one could test my three algorithms with different initial evidence values to see how they perform with unlikely evidence values.

Another point that should be addressed in the future is the experimental design for the comparison of my three new algorithms with AIS. I always tested AIS with 2,000 samples – a number I chose so the AIS running time would be similar to the running time of my new techniques. However, the AIS running time is always slower than the new techniques, while its RMSE values are lower. Clearly, 2,000 was not a good choice for the number of samples. A better experiment would be to let AIS run for the same amount of time as my techniques, and then to compare the error values between the algorithms

with that fixed amount of time. I did a few preliminary tests where I limited the AIS time to the time required by my algorithms, and in those cases the AIS error values were higher than for my techniques.

The techniques of reducing a network to a polytree or bounding the clique sizes in a junction tree are novel approaches to creating efficient approximate inference algorithms for Bayesian networks. As such, there is much room for research in constraining the structure of a network in order to speed up the inference process.

BIBLIOGRAPHY

[Amir01] E. Amir. Efficient approximation for triangulation of minimum treewidth. *Uncertainty in Artificial Intelligence (UAI2001),* pp. 7-15. Morgan Kaufmann, 2001.

[Arnborg87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a *k*-tree. *SIAM Journal on Algebraic and Discrete Methods*, vol. 8, pp. 277-284, 1987.

[Bodlaender01] H. Bodlaender, A. Koster, F. van den Eijkhof, and L. van der Gaag. Pre-processing for triangulation of probabilistic networks. *Uncertainty in Artificial Intelligence (UAI2001),* pp. 32-39. Morgan Kaufmann, 2001.

[CanoMoral94] A. Cano and S. Moral. Heuristic Algorithms for the Triangulation of Graphs. In *International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems, Advances in Intelligent Computing*, pp. 98-107, 1994.

[ChengDruzdzel00] J. Cheng and M. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, vol. 13, pp. 155-188, 2000.

[Cooper90] G. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, vol. 42, pp. 393-405. Elsevier Science Publishers, 1990.

[CoverThomas91] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New York, 1991.

[DagumLuby93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artifiical Intelligence*, vol. 60, pp. 141-153. Elsevier Science Publishers, 1993.

[Dasgupta99] S. Dasgupta. Learning polytrees. *Uncertainty in Artificial Intelligence (UAI1999).* Morgan Kaufmann, 1999.

[Diestel00] R. Diestel. *Graduate Texts in Mathematics: Graph Theory*. Springer, New York, 2000.

[FungChang88] R. Fung and K. Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence*, vol. 5, pp. 209-219. Elsevier Science Publishers, New York, 1988.

[Henrion88] M. Henrion. Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling. In *Uncertainty in Artificial Intelligence*, vol. 2, pp. 149-163. Elsevier Science Publishers, New York, 1988.

[Kjaerulff93] U. Kjaerulff. Approximation of Bayesian networks through edge removals. *Technical Report IR-93-2007, Department of Mathematics and Computer Science, Institute for Electronic Systems, Denmark*, August 1993.

[LauritzenSpiegelhalter88] S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society,* vol. B 50, pp. 253 - 258, 1988.

[Murphy98] K. Murphy. *A Brief Introduction to Graphical Models and Bayesian Networks*, 1998. Retrieved December 16, 2004, from http://www.ai.mit.edu/~murphyk/Bayes/bayes.html.

[Neapolitan04] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, Upper Saddle River, NJ, 2004.

[Pearl88] J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, San Mateo, CA, 1988.

[SchachterPeot90] R. Shachter and M. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence*, vol. 5, pp. 311-318. Elsevier Science Publishers, New York, 1990.

[TarjanYannakakis84] R. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, check acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. In *SIAM Journal of Computing*, vol. 13, pp. 566-579, 1984.

[vanEngelen96] R. van Engelen. Approximating Bayesian belief networks by arc removal. *Technical Report TR-96-15, Department of Computer Science, Leiden University, the Netherlands*, June 1996.

[Wen90] W. Wen. Optimal decomposition of belief networks. In *Uncertainty in Artificial Intelligence*, pp. 245-256, 1990.

APPENDIX

This appendix contains detailed walkthroughs of some of the algorithms presented in the background section in Chapter 1. Included in this section are walkthroughs of Pearl's algorithm, the Lauritzen-Spiegelhalter algorithm, the maximum cardinality search triangulation algorithm, a computation of the optimized Kullback-Leibler information divergence for an edge in a Bayesian network, and Bodlaender's triangulation algorithm with pre-processing rules.

A.1 Walkthrough of Pearl's Algorithm

Below I provide a walkthrough of Pearl's algorithm on a small example.

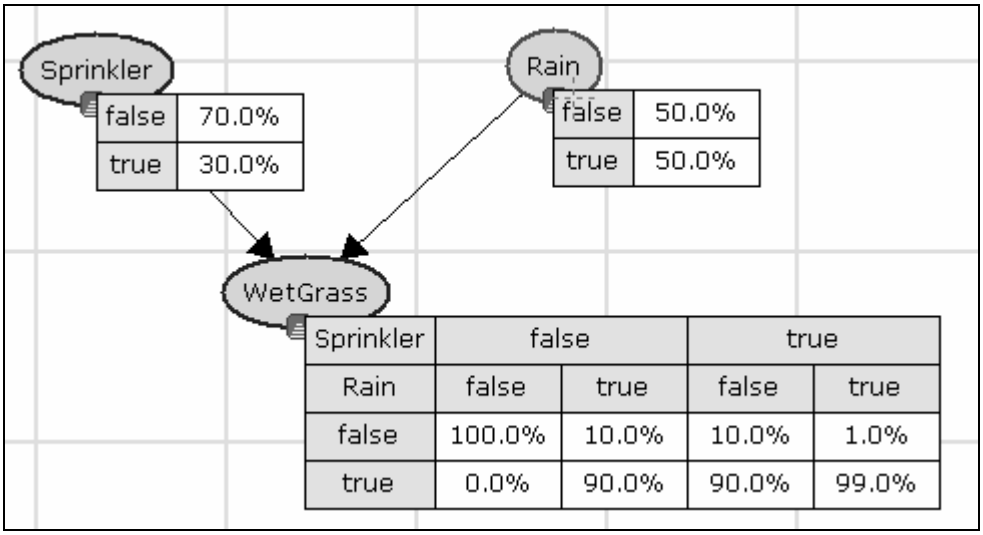

Figure 24. Small Sprinkler-Rain network for the walkthrough of Pearl's algorithm

Consider the Bayesian network in Figure 24 above. First, suppose there is no evidence. Then Pearl's algorithm would compute the posterior probabilities for each node as follows. For simplicity, I will denote the node Sprinkler as "*S*", the node Rain as "*R*",

and the node WetGrass as "*W*". I will also denote the value false as "F" and the value true as "T".

Initialization:

1) Set all λ values, λ messages, and π messages to 1:

$\lambda(S = \text{T}) = 1$, $\lambda(S = \text{F}) = 1$

$\lambda(R = \text{T}) = 1$, $\lambda(R = \text{F}) = 1$

$\lambda(W = \text{T}) = 1$, $\lambda(W = \text{F}) = 1$

$\pi_W(S = \text{T}) = 1$, $\pi_W(S = \text{F}) = 1$

$\pi_W(R = \text{T}) = 1$, $\pi_W(R = \text{F}) = 1$

$\lambda_W(S = \text{T}) = 1$, $\lambda_W(S = \text{F}) = 1$

$\lambda_W(R = \text{T}) = 1$, $\lambda_W(R = \text{F}) = 1$

2) For all roots *A*, if *A* has *m* possible values, then for $1 \leq j \leq m$, set $\pi(A = a_j) = P(aj)$:

$\pi(S = \text{T}) = 0.3$, $\pi(S = \text{F}) = 0.7$

$\pi(R = \text{T}) = 0.5$, $\pi(R = \text{F}) = 0.5$

3) For all roots *A* and for all children *B* of *A*, post a new π message to *B*:

$\pi_W(S = \text{T}) = 0.3$, $\pi_W(S = \text{F}) = 0.7$

$\pi_W(R = \text{T}) = 0.5$, $\pi_W(R = \text{F}) = 0.5$

Main Flow:

1) W receives a π message from S (note that the new π message from R is still (1, 1)):

$$\pi(W = \mathrm{T}) = P(W = \mathrm{T}|S = \mathrm{T}, R = \mathrm{T})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{T})$$
$$+ P(W = \mathrm{T}|S = \mathrm{T}, R = \mathrm{F})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{F})$$
$$+ P(W = \mathrm{T}|S = \mathrm{F}, R = \mathrm{T})\,\pi_W(S = \mathrm{F})\,\pi_W(R = \mathrm{T})$$
$$+ P(W = \mathrm{T}|S = \mathrm{F}, R = \mathrm{F})\,\pi_W(S = \mathrm{F})\,\pi_W(R = \mathrm{F})$$
$$= 0.99*0.3*1+0.9*0.3*1+0.9*0.7*1+0*0.7*1 = 1.197$$

$$\pi(W = \mathrm{F}) = P(W = \mathrm{F}|S = \mathrm{T}, R = \mathrm{T})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{T})$$
$$+ P(W = \mathrm{F}|S = \mathrm{T}, R = \mathrm{F})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{F})$$
$$+ P(W = \mathrm{F}|S = \mathrm{F}, R = \mathrm{T})\,\pi_W(S = \mathrm{F})\,\pi_W(R = \mathrm{T})$$
$$+ P(W = \mathrm{F}|S = \mathrm{F}, R = \mathrm{F})\,\pi_W(S = \mathrm{F})\,\pi_W(R = \mathrm{F})$$
$$= 0.01*0.3*1+0.1*0.3*1+0.1*0.7*1+1*0.7*1 = 0.803$$

$$P'(W = \mathrm{T}) = \alpha*1*1.197 = 1.197\alpha$$

$$P'(W = \mathrm{F}) = \alpha*1*0.803 = 1.197\alpha$$

Normalizing, we get $P'(W = \mathrm{T}) = 0.5985$, $P'(W = \mathrm{F}) = 0.4015$.

2) W receives a π message from R:

$$\pi(W = \mathrm{T}) = P(W = \mathrm{T}|S = \mathrm{T}, R = \mathrm{T})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{T})$$
$$+ P(W = \mathrm{T}|S = \mathrm{T}, R = \mathrm{F})\,\pi_W(S = \mathrm{T})\,\pi_W(R = \mathrm{F})$$
$$+ P(W = \mathrm{T}|S = \mathrm{F}, R = \mathrm{T})\,\pi_W(S = \mathrm{F})\,\pi_W(R = \mathrm{T})$$

$$+ P(W = \mathrm{T}|S = \mathrm{F}, R = \mathrm{F})\ \pi_W(S = \mathrm{F})\ \pi_W(R = \mathrm{F})$$

$$= 0.99*0.3*0.5+0.9*0.3*0.5+0.9*0.7*0.5+0*0.7*0.5 = 0.5985$$

$$\pi(W = \mathrm{F}) = P(W = \mathrm{F}|S = \mathrm{T}, R = \mathrm{T})\ \pi_W(S = \mathrm{T})\ \pi_W(R = \mathrm{T})$$

$$+ P(W = \mathrm{F}|S = \mathrm{T}, R = \mathrm{F})\ \pi_W(S = \mathrm{T})\ \pi_W(R = \mathrm{F})$$

$$+ P(W = \mathrm{F}|S = \mathrm{F}, R = \mathrm{T})\ \pi_W(S = \mathrm{F})\ \pi_W(R = \mathrm{T})$$

$$+ P(W = \mathrm{F}|S = \mathrm{F}, R = \mathrm{F})\ \pi_W(S = \mathrm{F})\ \pi_W(R = \mathrm{F})$$

$$= 0.01*0.3*0.5+0.1*0.3*0.5+0.1*0.7*0.5+1*0.7*0.5 = 0.4015$$

$$P'(W = \mathrm{T}) = \alpha*1*1.197 = 0.5985\alpha$$

$$P'(W = \mathrm{F}) = \alpha*1*0.803 = 0.4015\alpha$$

Normalizing, we get $P'(W = \mathrm{T}) = 0.5985$, $P'(W = \mathrm{F}) = 0.4015$.

3) There are no more messages to send, so propagation ends. Since no messages were sent to R or S, their posterior probabilities are the same as their prior probabilities. Thus we have that:

$$P'(S = \mathrm{T}) = 0.3, \qquad P'(S = \mathrm{F}) = 0.7$$

$$P'(R = \mathrm{T}) = 0.5, \qquad P'(R = \mathrm{F}) = 0.5$$

$$P'(W = \mathrm{T}) = 0.5985, \quad P'(W = \mathrm{F}) = 0.4015$$

Continuing the walkthrough above, consider next that the node WetGrass is instantiated to false. Pearl's algorithm would then calculate the new posterior probabilities for each node given this new evidence as follows:

1) W is instantiated to F:

$P'(W = \mathrm{T}) = 0, \qquad P'(W = \mathrm{F}) = 1$

$\lambda(W = \mathrm{T}) = 0, \qquad \lambda(W = \mathrm{F}) = 1$

Send λ messages to S and R:

$$\lambda_W(S = \mathrm{T}) = \pi_W(R = \mathrm{T})*[P(W = \mathrm{T}|R = \mathrm{T}, S = \mathrm{T})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{T}, S = \mathrm{T})\,\lambda(W = \mathrm{F})] + \pi_W(R = \mathrm{F})*[P(W = \mathrm{T}|R = \mathrm{F}, S = \mathrm{T})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{F}, S = \mathrm{T})\,\lambda(W = \mathrm{F})]$$

$$= 0.5*[0.99*0 + 0.01*1] + 0.5*[0.9*0 + 0.1*1] = 0.055$$

$$\lambda_W(S = \mathrm{F}) = \pi_W(R = \mathrm{T})*[P(W = \mathrm{T}|R = \mathrm{T}, S = \mathrm{F})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{T}, S = \mathrm{F})\,\lambda(W = \mathrm{F})] + \pi_W(R = \mathrm{F})*[P(W = \mathrm{T}|R = \mathrm{F}, S = \mathrm{F})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{F}, S = \mathrm{F})\,\lambda(W = \mathrm{F})]$$

$$= 0.5*[0.9*0 + 0.1*1] + 0.5*[0*0 + 1*1] = 0.55$$

$$\lambda_W(R = \mathrm{T}) = \pi_W(S = \mathrm{T})*[P(W = \mathrm{T}|R = \mathrm{T}, S = \mathrm{T})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{T}, S = \mathrm{T})\,\lambda(W = \mathrm{F})] + \pi_W(S = \mathrm{F})*[P(W = \mathrm{T}|R = \mathrm{F}, S = \mathrm{T})\,\lambda(W = \mathrm{T}) + P(W = \mathrm{F}|R = \mathrm{F}, S = \mathrm{T})\,\lambda(W = \mathrm{F})]$$

$$= 0.3*[0.99*0 + 0.01*1] + 0.7*[0.9*0 + 0.1*1] = 0.073$$

$$\lambda_W(R = \text{F}) = \pi_W(S = \text{T})*[P(W = \text{T}|R = \text{T}, S = \text{F})\,\lambda(W = \text{T}) + P(W = \text{F}|R = \text{T}, S = \text{F})\,\lambda(W = \text{F})] + \pi_W(S = \text{F})*[P(W = \text{T}|R = \text{F}, S = \text{F})\,\lambda(W = \text{T}) + P(W = \text{F}|R = \text{F}, S = \text{F})\,\lambda(W = \text{F})]$$

$$= 0.3*[0.9*0 + 0.1*1] + 0.7*[0*0 + 1*1] = 0.73$$

2) S receives a λ message from W:

$\lambda(S = \text{T}) = \lambda_W(S = \text{T}) = 0.055$

$\lambda(S = \text{F}) = \lambda_W(S = \text{F}) = 0.55$

$P'(S = \text{T}) = \alpha*0.055*0.3 = 0.0165\alpha$

$P'(S = \text{F}) = \alpha*0.55*0.7 = 0.385\alpha$

Normalizing, we get $P'(S = \text{T}) = 0.0411$, $P'(S = \text{F}) = 0.9589$.

3) R receives a λ message from W:

$\lambda(R = \text{T}) = \lambda_W(R = \text{T}) = 0.073$

$\lambda(R = \text{F}) = \lambda_W(R = \text{F}) = 0.73$

$P'(R = \text{T}) = \alpha*0.073*0.5 = 0.0365\alpha$

$P'(R = \text{F}) = \alpha*0.73*0.5 = 0.365\alpha$

Normalizing, we get $P'(R = \text{T}) = 0.0909$, $P'(R = \text{F}) = 0.909$.

4) There are no more messages to send, so propagation ends. The posterior probabilities (gathered in the steps above) are as follows:

$P'(S = \text{T}) = 0.0411, \quad P'(S = \text{F}) = 0.9589$

$P'(R = \text{T}) = 0.0909, \quad P'(R = \text{F}) = 0.909$

$P'(W = \text{T}) = 0, \quad P'(W = \text{F}) = 1$

A.2 Walkthrough of the Lauritzen-Spiegelhalter Algorithm

Because the message-passing portion of LS is very similar to Pearl's algorithm, I will only go through an example of creating a clique tree from a Bayesian network here. For a more detailed description of LS, see [LauritzenSpiegelhalter88].

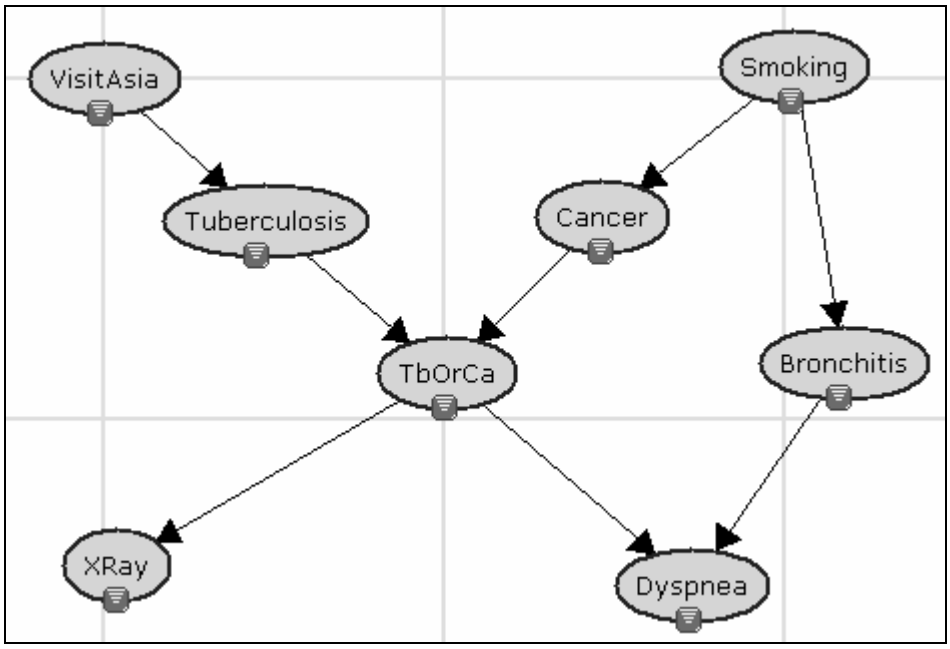

Figure 25. The Asia Bayesian network for the walkthrough of LS

In Figure 25 above is our original Bayesian network, called Asia. This "toy" network is used as an example in many AI textbooks and papers. The first step in the LS algorithm is to moralize the graph by adding undirected edges between common parents and removing directionality of the edges. In Asia, the nodes Tuberculosis and Cancer are both parents of the TbOrCa node, so an edge is added between them. Also, the nodes

TbOrCa and Bronchitis are both parents of the Dyspnea node, so an edge is added between them.

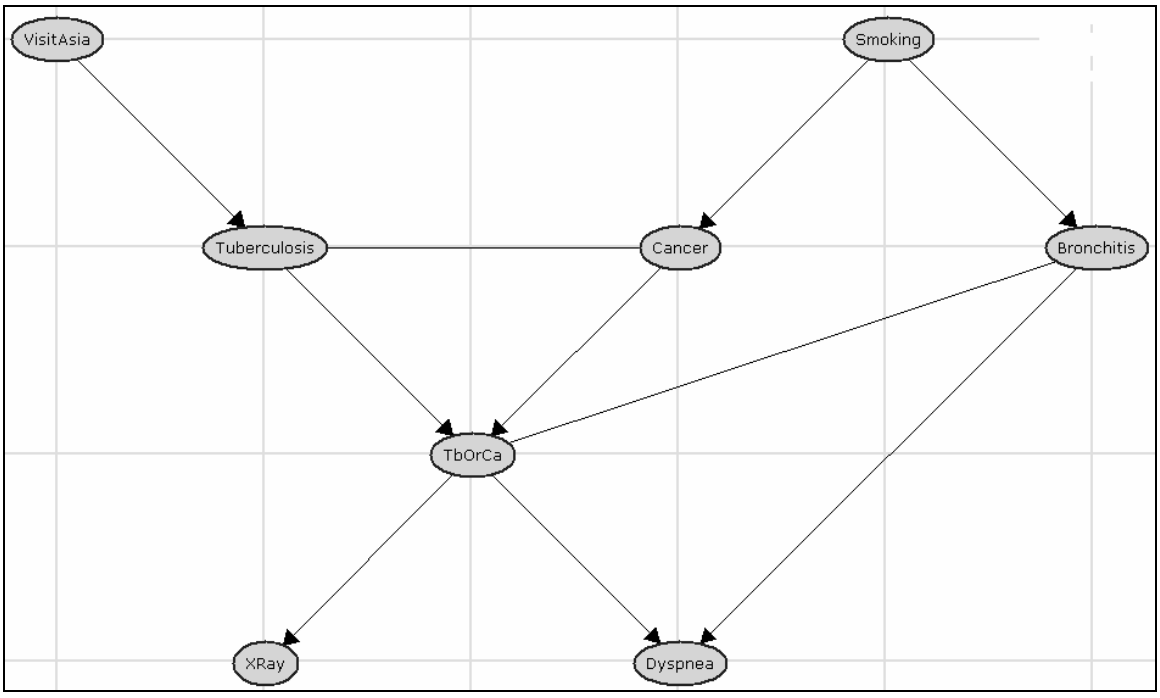

Figure 26. The moralized graph for the Asia network

Figure 26 above shows the result of adding the undirected edges {Tuberculosis, Cancer} and {TbOrCa, Bronchitis} to moralize the graph.

The next step in the LS algorithm is to triangulate the moralized graph. A graph is triangulated if and only if it contains no induced subgraph of that is a simple cycle of length at least four. Triangulation algorithms are covered in detail in Section 1.3, but for now we can examine the graph visually and add the necessary edges.

Notice that in the moralized graph in Figure 26, the cycle {TbOrCa, Cancer, Smoking, Bronchitis} has length 4. Thus, we must add a chord to break this cycle. If we add the edge {Cancer, Bronchitis}, then the graph will be triangulated. However,

because of the triangulation algorithm that I used to model this example, the edge {Tuberculosis, Bronchitis} is also added. (Note that if unnecessary edges are added to a triangulated graph, the resulting graph will still be triangulated. The extra edges, however, may increase the size of the largest clique.)

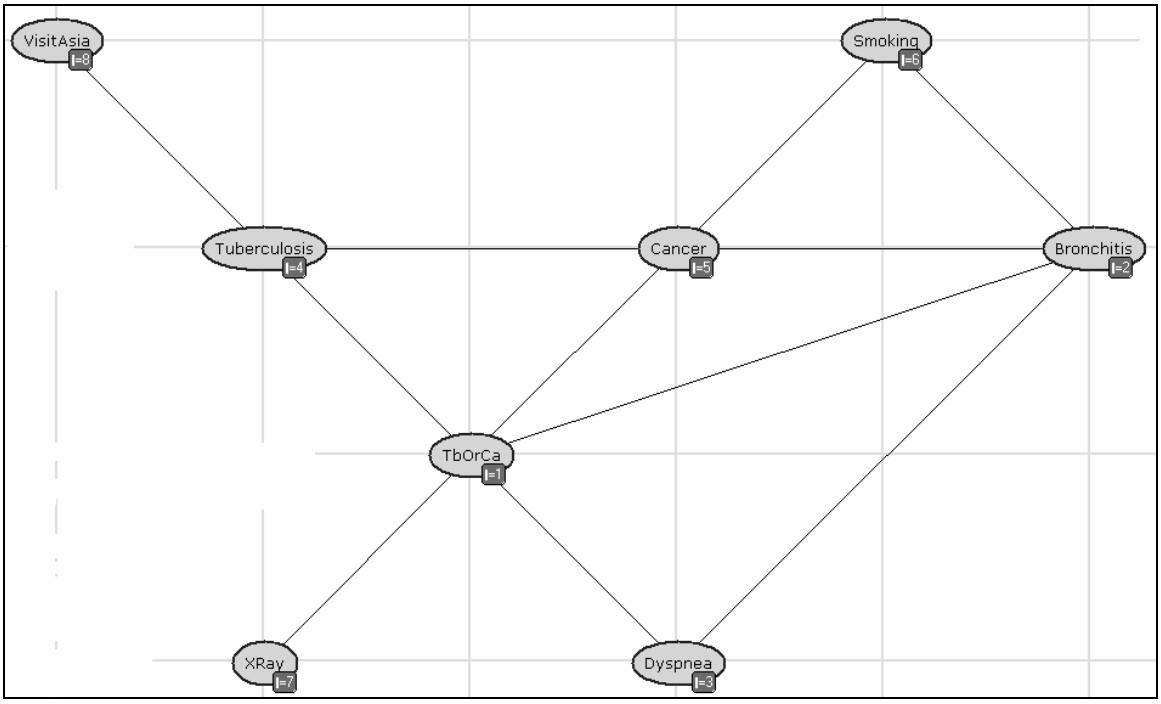

Figure 27. A triangulation of the Asia Bayesian network

Figure 27 above shows the Asia network after triangulating the moralized graph by adding the chord {Cancer, Bronchitis} and the additional edge {Tuberculosis, Bronchitis}.

Finally, we must identify cliques and construct a clique-tree from them. Again, a detailed description appears in Section 1.3 on how to identify cliques in a triangulated graph, but for now we will pick them out visually. By looking at the graph in Figure 27, we can pick out the cliques {TbOrCa, Dyspnea, Bronchitis}, {TbOrCa, Bronchitis,

Tuberculosis, Cancer}, {TbOrCa, XRay}, {Bronchitis, Cancer, Smoking}, and {VisitAsia, Tuberculosis}. Once we have identified the cliques, we must place them in a clique-tree. Two cliques in a clique-tree can only be connected by an edge if they have at least one node in common. The clique-tree must also maximize the total number of "separator" nodes that are common between cliques. Essentially, we build the clique-tree by first making a graph with the different cliques as nodes and with all possible edges between cliques. Each edge is weighted with the number of separator nodes that are common between the two cliques on that edge. The clique-tree is then the maximal spanning tree of the constructed graph of cliques.

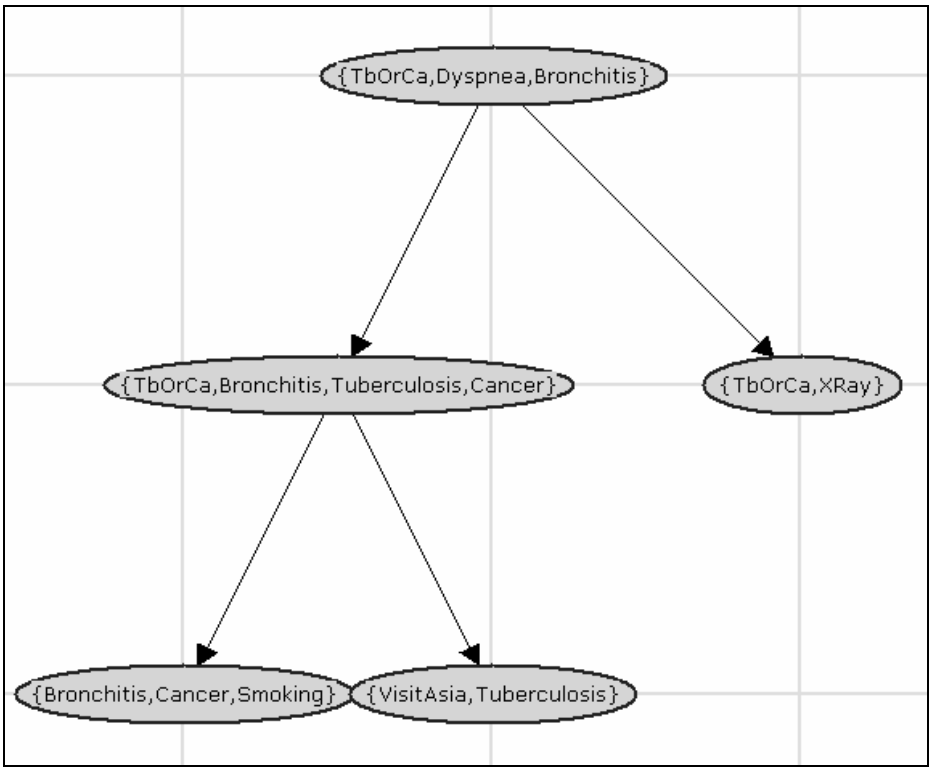

Figure 28. The clique-tree for the Asia network

Figure 28 above shows the construction of the clique tree from the five cliques in the triangulated graph that maximizes the number of separator nodes. LS would now

initialize the clique potentials by using the conditional probabilities of each node in a clique, perform message-passing to update each clique potential to reflect the evidence above and below it, and then to extract the posterior probabilities of each node by marginalizing the final clique potentials.

A.3 Walkthrough of the Maximum Cardinality Search Algorithm

Consider again the moralized graph for the Asia network in Figure 26. Assuming that the directionality of each edge has been removed, a walkthrough of the MCS algorithm on the undirected, moralized version of Asia appears below. Note that a vertex v is considered *unnumbered* if $f(v)$ is undefined.

Initialization: for all vertices $v \in V$, $w(v) \leftarrow 0$

w(VisitAsia) $\leftarrow 0$, w(Smoking) $\leftarrow 0$,

w(Tuberculosis) $\leftarrow 0$, w(Cancer) $\leftarrow 0$,

w(Bronchitis) $\leftarrow 0$, w(TbOrCa) $\leftarrow 0$,

w(XRay) $\leftarrow 0$, w(Dyspnea) $\leftarrow 0$.

Main step 1, i = 8:

choose z = VisitAsia, set f(VisitAsia) = 8

VisitAsia's unnumbered adjacent vertices include {Tuberculosis}:

w(Tuberculosis) $\leftarrow 1$.

Main step 2, i = 7:

choose z = Tuberculosis, set f(Tuberculosis) = 7

Tuberculosis's unnumbered adjacent vertices include {Cancer, TbOrCa}:

w(Cancer) ← 1, w(TbOrCa) ← 1.

Main step 3, i = 6:

choose z = Cancer, set f(Cancer) = 6

Cancer's unnumbered adjacent vertices include {TbOrCa, Smoking}:

w(TbOrCa) ← 2, w(Smoking) ← 1.

Main step 4, i = 5:

choose z = TbOrCa, set f(TbOrCa) = 5

TbOrCa's unnumbered adjacent vertices include {XRay, Dyspnea, Bronchitis}:

w(XRay) ← 1, w(Dyspnea) ← 1, w(Bronchitis) ← 1.

Main step 5, i = 4:

choose z = Smoking, set f(Smoking) = 4

Smoking's unnumbered adjacent vertices include {Bronchitis}:

w(Bronchitis) ← 2.

Main step 6, i = 3:

choose z = Bronchitis, set f(Bronchitis) = 3

Bronchitis's unnumbered adjacent vertices include {Dyspnea}:

w(Dyspnea) ← 2.

Main step 7, i = 2:

choose z = Dyspnea, set f(Dyspnea) = 2

Bronchitis has no unnumbered adjacent vertices.

Main step 8, i = 1:

choose z = XRay, set f(XRay) = 1

XRay has no unnumbered adjacent vertices.

MCS algorithm ends.

Next, we must compute the fill-in of the graph given f by stepping through each vertex and adding edges so that its higher ordered neighbors form a clique. Table 11 below shows the current vertex in each step of the fill-in, its neighbors, its higher-ordered neighbors, and the new edges that must be added to the graph to make its higher-ordered neighbors form a clique. Note that a node's neighbors at a given stage in this process might include a neighbor added in a previous step.

Node order	**Current Vertex**	**Neighbors**	**Higher-Ordered Neighbors**	**New Edges Added**
1	VisitAsia	Tuberculosis	Tuberculosis	none
2	Tuberculosis	VisitAsia, Cancer, TbOrCa	Cancer, TbOrCa	none (edge {Cancer, TbOrCa} already there)
3	Cancer	Tuberculosis, Smoking, TbOrCa	Smoking, TbOrCa	{Smoking, TbOrCa}
4	TbOrCa	Tuberculosis, Cancer, Bronchitis, Smoking, XRay, Dyspnea	Bronchitis, Smoking, XRay, Dyspnea	{Bronchitis, XRay}, {Smoking, XRay}, {Smoking, Dyspnea}, {XRay, Dyspnea}

5	Smoking	Cancer, Bronchitis, TbOrCa	Bronchitis	none
6	Bronchitis	Smoking, TbOrCa, Dyspnea	Dyspnea	none
7	Dyspnea	TbOrCa, Bronchitis	none	none
8	XRay	TbOrCa	none	none

Table 11. The fill-in construction on the Asia network using an ordering from MCS

If we add the edges from the table above to the original graph, we get the following result:

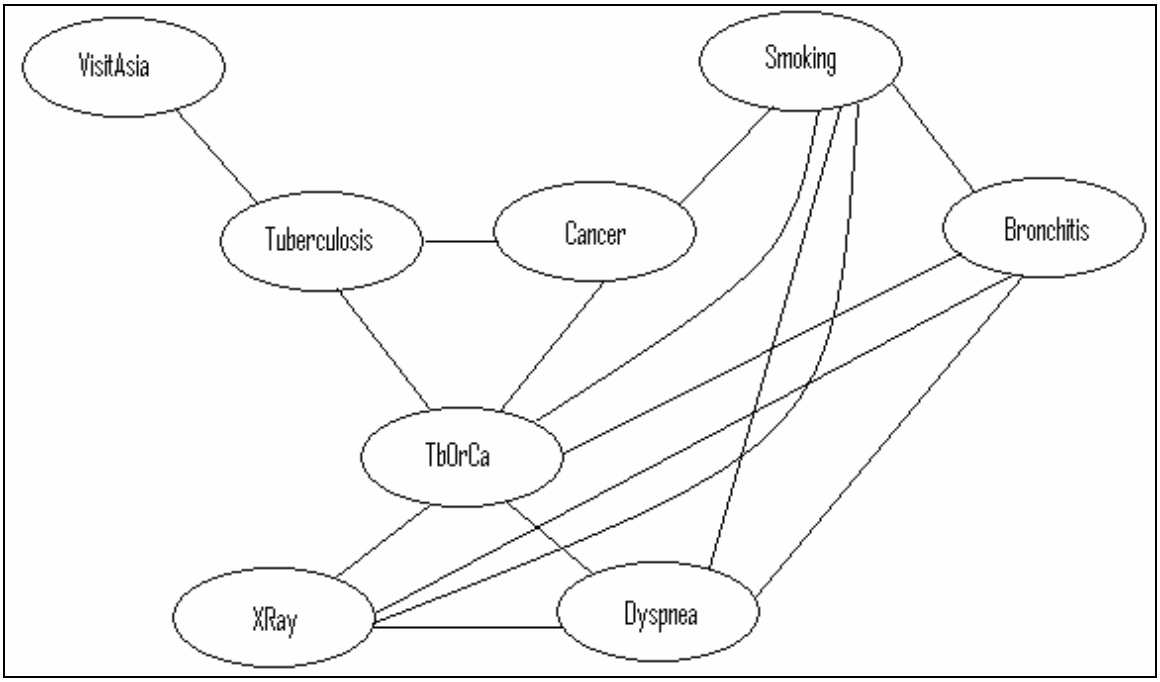

Figure 29. The triangulation of the Asia network using MCS

Figure 29 above shows the result of the moralized, undirected Asia network after triangulation with the MCS algorithm. Triangulation resulting in adding the edges {Smoking, TbOrCa}, {XRay, Bronchitis}, {Smoking, XRay}, {Smoking, Dyspnea}, and

{XRay, Dyspnea} to the moralized, undirected graph, which yields a maximum clique size of five in the triangulated graph. (Note that this triangulation of the Asia Bayesian network is different than the triangulation given in Figure 27 as part of the LS walkthrough because the nodes were initially chosen in a different order in the two triangulations.)

A.4 Example Calculation of Optimized KL Divergence

Consider the Sprinkler Bayesian network in Figure 1. Suppose that we want to compute the KL divergence between the original probability distribution and the probability distribution that results from deleting the edge (Cloudy, Rain).

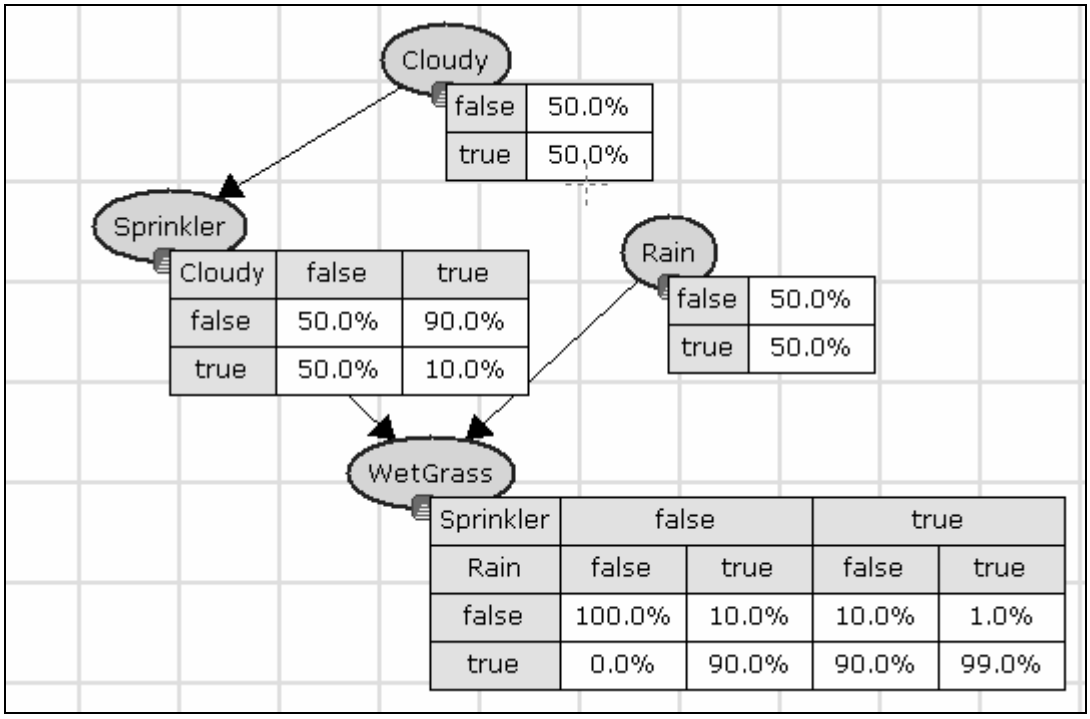

Figure 30. Sprinkler network with edge (Cloudy, Rain) deleted ← Sprinkler'

Figure 30 above shows the network (now called Sprinkler' with probability distribution *P'*) and corresponding probability distribution that results from deleting edge (Cloudy,

Rain) from the Sprinkler Bayesian network. Now, to compute the optimized KL divergence between the probability distribution for Sprinkler and *P'*, we must step through each possible instantiation of the parents of Rain and Rain itself. This means we must step through each instantiation of {Cloudy, Rain}. To save space, I will refer to Cloudy as C and Rain as R:

P(Rain = T | Cloudy = T) * P(Cloudy = T) *

log(P(Rain = T | Cloudy = T) / P'(Rain = T) +

P(Rain = F | Cloudy = T) * P(Cloudy = T) *

log(P(Rain = F | Cloudy = T) / P'(Rain = F) +

P(Rain = T | Cloudy = F) * P(Cloudy = F) *

log(P(Rain = T | Cloudy = F) / P'(Rain = T) +

P(Rain = F | Cloudy = F) * P(Cloudy = F) *

log(P(Rain = F | Cloudy = F) / P'(Rain = F)

By looking up the appropriate values for the probabilities in Figures 22 and 1, the above formula becomes the following:

0.8*0.5*log(0.8/0.5) + 0.2*0.5*log(0.2/0.5) +

0.2*0.5*log(0.2/0.5) + 0.8*0.5*log(0.8/0.5)

which is 0.1927. By comparing this number with the KL divergence for deleting other edges in the Sprinkler network, we can determine which edges contain the most information. If deleting an edge yields a comparatively high KL divergence, then it is

more important to the network. If it yields a comparatively low KL divergence, then that edge is less important to the network.

A.5 Walkthrough of Triangulation Pre-Processing Rules

Consider again the moralized graph for the Asia network in Figure 26. Assuming that the directionality of each edge has been removed, a walkthrough of the pre-processing algorithm on the undirected, moralized version of Asia appears below.

1) Initialize values: $low \leftarrow 1$, $S \leftarrow \{\}$.
2) VisitAsia is a twig, so it is removed. $S \leftarrow$ {VisitAsia}.
3) XRay is a twig, so it is removed. $S \leftarrow$ {XRay, VisitAsia}.
4) No more reduction rules can be applied, so $low \leftarrow low + 1 = 2$.
5) The series rule can be applied to Dyspnea. Since its only neighbors, TbOrCa and Bronchitis, are already adjacent, no edges need to be added. $S \leftarrow$ {Dyspnea, XRay, VisitAsia}.

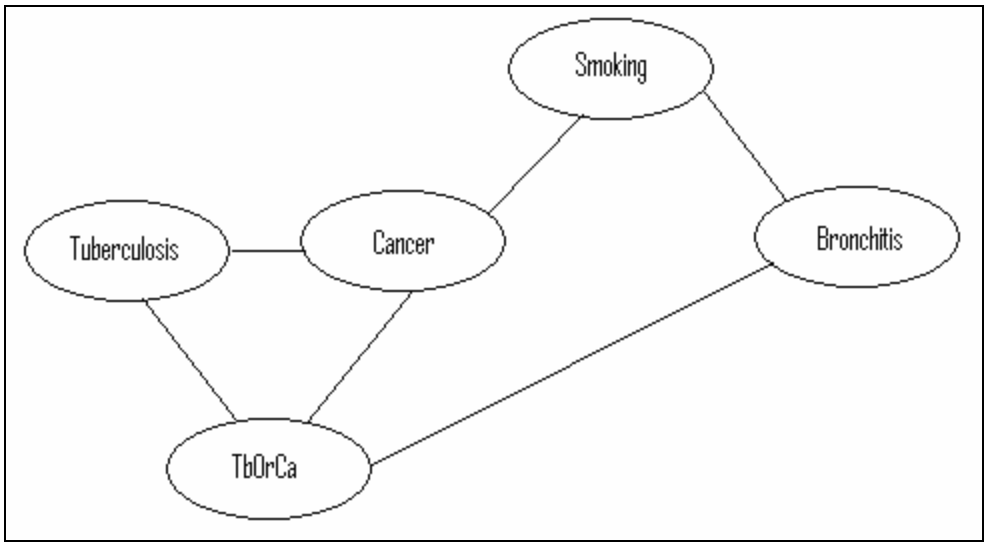

Figure 31. The remaining graph in Asia after step 5 in the triangulation pre-processing rules

6) The modified graph at this point in the algorithm appears above in Figure 31. The series rule can now be applied to Tuberculosis. Since its only neighbors, Cancer and TbOrCa, are already adjacent, no edges need to be added. $S \leftarrow$ {Tuberculosis, Dyspnea, XRay, VisitAsia}.

7) The series rule can be applied to TbOrCa. Its neighbors, Cancer and Bronchitis, are not adjacent, so the edge {Cancer, Bronchitis} must be added to the graph. $S \leftarrow$ {TbOrCa, Tuberculosis, Dyspnea, XRay, VisitAsia}.

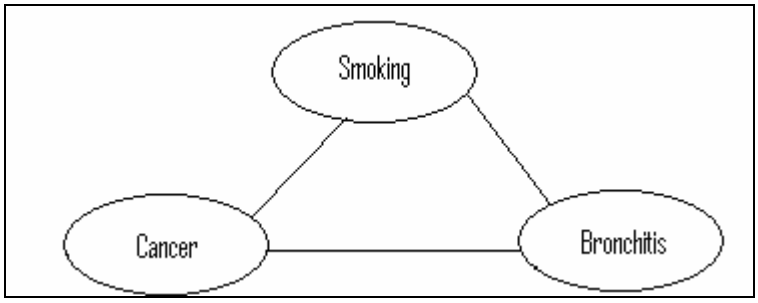

Figure 32: The remaining graph in Asia after step 7 in the triangulation pre-processing rules

8) The modified graph at this point in the algorithm appears above in Figure 32. The series rule can now be applied to Cancer. Since its only neighbors, Smoking and Bronchitis, are already adjacent, no edges need to be added. $S \leftarrow$ {Cancer, TbOrCa, Tuberculosis, Dyspnea, XRay, VisitAsia}.

9) Smoking is a twig, so it is removed. $S \leftarrow$ {Smoking, Cancer, TbOrCa, Tuberculosis, Dyspnea, XRay, VisitAsia}.

10) Bronchitis is a twig, so it is removed. $S \leftarrow$ {Bronchitis, Smoking, Cancer, TbOrCa, Tuberculosis, Dyspnea, XRay, VisitAsia}.

11) All vertices have been removed, so the reduction rules stop. Also, since all vertices have been removed, the perfect elimination scheme *f* for the original graph is given by:

f(VisitAsia) = 1, *f*(XRay) = 2,

f(Dyspnea) = 3, *f*(Tuberculosis) = 4,

f(TbOrCa) = 5, *f*(Cancer) = 6,

f(Smoking) = 7, *f*(Bronchitis) = 8.

12) The fill-in of the original moralized graph given *f* is now constructed, just like it was in the MCS algorithm in Table 3. (The only edge added in the fill-in stage is {Cancer, Bronchitis}.) The resulting triangulated graph appears below in Figure 33.

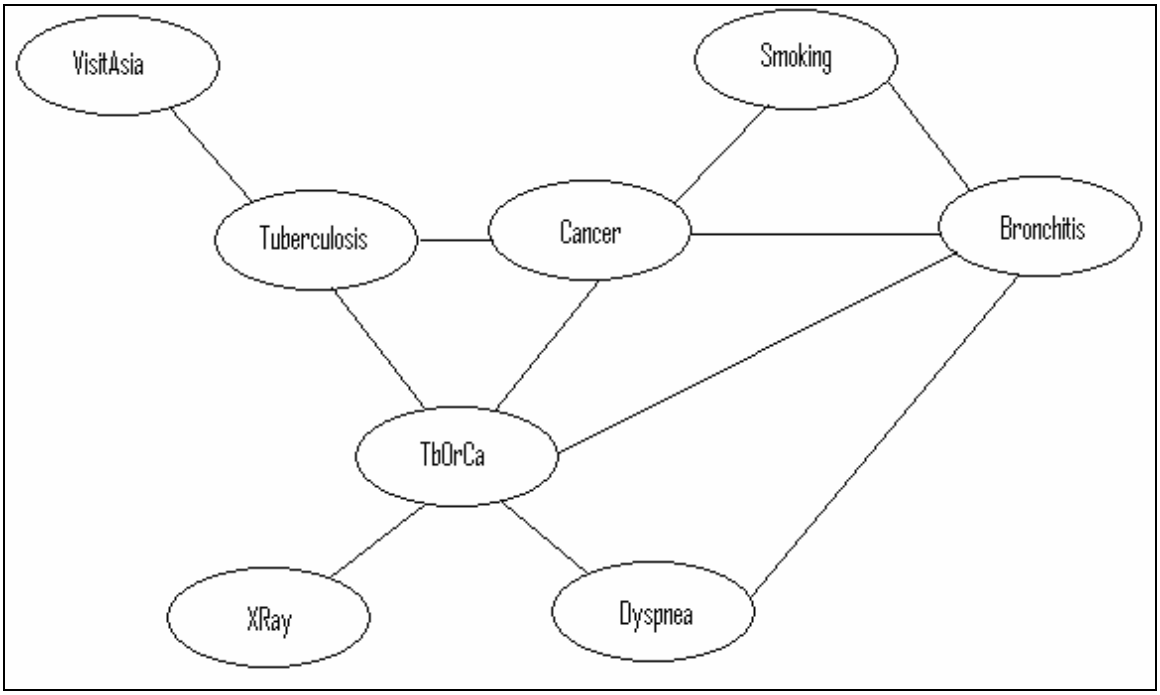

Figure 33. The triangulation of the Asia network using pre-processing rules