EXPLOITING THE PROBABILITY OF OBSERVATION FOR EFFICIENT BAYESIAN NETWORK INFERENCE

FOUZIA ASHRAF MOUSUMI
Bachelor of Science, University of Chittagong, Bangladesh, 2008

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

© Fouzia Ashraf Mousumi, 2013

I wish to dedicate this thesis to my loving family.

Abstract

It is well-known that the observation of a variable in a Bayesian network can affect the effective connectivity of the network, which in turn affects the efficiency of inference. Unfortunately, the observed variables may not be known until runtime, which limits the amount of compile-time optimization that can be done in this regard. This thesis considers how to improve inference when users know the likelihood of a variable being observed. It demonstrates how these *probabilities of observation* can be exploited to improve existing heuristics for choosing elimination orderings for inference. Empirical tests over a set of benchmark networks using the Variable Elimination algorithm show reductions of up to 50% and 70% in multiplications and summations, as well as runtime reductions of up to 55%. Similarly, tests using the Elimination Tree algorithm show reductions by as much as 64%, 55%, and 50% in recursive calls, total cache size, and runtime, respectively.

Acknowledgments

I take much pleasure to express my profound gratitude to my supervisor Dr. Kevin Grant for his persistent and inspiring supervision and helping me learn the ABC of Bayesian Networks. Without his endless help and continuous reassurance at the most difficult times, this thesis would not have been a reality. I am indebted to him for his support, encouragement, suggestions, generosity, and the invaluable knowledge he shared with me.

I would also like to express my deep acknowledgement to my co-supervisor Dr. Stephen Wismath for his continuous support, inspiration, and guidelines.

I am also grateful for having an exceptional supervisory committee and wish to thank Dr. Stacey Wetmore and Dr. John Sheriff for their support, valuable suggestions, and comments.

A special thanks to Dr. Michael Horsch for his valuable suggestions regarding this research project.

My cordial thanks to the University of Lethbridge and NSERC for the financial and travel support. I also would like to thank specially the School of Graduate Studies and the department of Mathematics and Computer Science for supporting me throughout the years and adding to my strength as I continue on the road ahead.

I am forever grateful to my loving family. The continuous word of encouragement from my parents and my sister always boosted my confidence and guided me through difficult times. I thank my husband, Rubaiyat and my loving daughter, Mehnaz, for their unconditional love, support, sacrifices and encouragement.

I would like to take this opportunity to thank all of my friends and well-wishers near and abroad who had faith in me even when I did not have faith in myself.

Above all, my sincere gratitude to the Almighty, who creates and makes things happen.

Contents

List of Abbreviations

CPT	Conditional Probability Table
ElimTree	Elimination Tree
JTP	Junction Tree Propagation
NP-hard	Non-deterministic Polynomial-time hard
Lex-BFS	Lexicographic Breadth-First Search
Lex-M	Lexicographic Breadth-First Search, variant minimal
MCS	Maximum Cardinality Search
MEO	Minimal Elimination Ordering
MinN	Minimum Neighbours
MinW	Minimum Weight
MinF	Minimum Fill
MMNW	Maximum Minimum Neighbourhood Weight
PEO	Perfect elimination ordering
VE	Variable Elimination
WMinF	Weighted Minimum Fill
WMCS	Weighted Maximum Cardinality Search
WEEA	Weighted Edge Elimination Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

The world is an uncertain place; often, knowledge is incomplete or imperfect, which causes uncertainty. Reasoning under uncertainty has become an important research topic in artificial intelligence (AI) over the last three decades, since there are many situations in which decision-making agents must act with incomplete information. Probability theory provides a sound mathematical basis for reasoning under uncertainty. It allows a system to maintain beliefs with incomplete (partial) knowledge. It also allows belief update by incorporating new pieces of evidence obtained from various sensors in a coherent way.

A Bayesian network [41, 38] is a graphical probabilistic representation of a domain of interest. Bayesian networks handle uncertainty by allowing multiple states of a problem to be possible, by specifying how likely each of the possible states is, by updating beliefs about the state of the problem based on our observations of the world, and by incorporating multiple sources of evidence in a coherent way. To deal with uncertainty, Bayesian networks also determine the degree of belief over some particular property of the domain, based on general beliefs about the problem and observations. Bayesian networks have gained significant popularity in recent decades, and have been used in modeling many real-time applications. For example, they have been used for troubleshooting in Microsoft Windows [15, 6], for predicting the likelihood of meeting attendance in an automated presence and availability forecasting service named COORDINATE [27], and in molecular biology for discovering interactions between genes based on multiple expression measurements [21].

Given a Bayesian network, we are typically interested in computing a particular piece of information given a set of observations, or *evidence*. For example, in a medical diagnosis system a user can query the probability of a particular disease given the results of

a physical examination and other medical tests. Computing posterior probabilities from a Bayesian network (henceforth referred to as *inference*) is at least NP-hard [9]. Popular inference algorithms, such as Junction Tree Propagation (JTP) [35, 29] and Variable Elimination (VE) [48, 13], perform reasonably well on real world problems, but still require considerable runtime and memory. Certain factors affect the efficiency of the inference task. One factor that has a significant impact on most inference algorithms is the choice of elimination ordering for its variables. Elimination orderings are used in query-based algorithms like VE when marginalizing variables from the Bayesian network, and in pre-compilation algorithms like JTP when constructing its associated cluster tree. A primary goal of Bayesian network practioners is to find variable orderings that minimize the runtime and memory costs of inference. Finding an optimal ordering is also NP-hard [31], however, approximation algorithms exist that produce good orderings in most situations. These algorithms dynamically select a variable from the network. Several factors such as network connectivity affect the chosen variable ordering, which in turn affects inference efficiency.

Certain actions can change the properties of a Bayesian network. For example, the observation of variables in a network reduce the effective connectivity of a Bayesian network; when a variable is observed, it effectively removes the outgoing edges from its associated node in the network (explained in Chapter 3). Since elimination order algorithms attempt to optimize the chosen ordering based on network connectivity, it stands to reason that the best orderings will be computed once observations are known. However, observations are typically not known until runtime, and recomputing variable orderings (or reconstructing cluster trees for JTP) each time new evidence is received can add significantly to the overhead of the online algorithm. Hence, elimination orders are often computed offline at compile-time, using heuristics that assume the worst-case, which is that no variables are observed.

If it was certain that a variable would be observed, then this information could be used at compile-time to compute better variable orderings; such guarantees may be impossible in practice. Consider a family history variable in a medical diagnosis system. Such a history may be unknown to a certain subset of patients (e.g., adoptees), or those immigrating from countries without adequate medical records. However, if the likelihood of a variable being observed is high enough, then assuming this observation at compile-time may produce better *expected* outcomes. For example, a sensor in a fault-detection network may have a known fail rate of 1%, hence, its associated variable should be observed 99% of the time.

In this thesis, the situation is considered where the *probability of observation* is known for a subset of the variables in a network. The primary goal is to exploit this piece of information to compute better elimination orderings at compile-time. It is shown how such knowledge can, on average, improve the elimination ordering produced at compile time compared to standard methods. This thesis demonstrates several variations on current elimination order algorithms, of varying sophistication, that are designed to exploit these probabilities of observation. The primary contribution of this thesis is *Weighted Edge Elimination Algorithms* (WEEA), which incorporate the probability of observation directly into the cost functions of existing greedy approaches to generate elimination orderings. The experimental results empirically suggest that the WEEA method improves the expected cost of inference in two popular inference algorithms.

The rest of the thesis is structured as follows. Chapter 2 presents necessary background information regarding Bayesian networks, with particular focus on computing elimination orderings. This chapter also discusses related work to understand the current state of research on elimination order algorithms. Chapter 3 introduces the probability of observation in further detail, and outlines a novel *thresholding* approach designed to exploit the probabilities of observation. We also present *Weighted Edge Elimination Algorithms* (WEEA), where we augment both the Bayesian network and the elimination heuristics to exploit the

probabilities of observation. Chapter 4 outlines our experiments and discusses the results. Finally, we present our conclusions, as well as future directions, in Chapter 5.

Published Work

Some of the material presented in this thesis has been previously published. In particular, Chapters 3 and 4 expand on materials published in [37].

Chapter 2

Background and Related Work

In this chapter, we review the necessary background knowledge to understand the methods and motivations of this dissertation. The beginning section introduces the terms and notation that will be used for the remainder of the document. We then present a light review of Bayesian networks, including inference methods for computing probabilities from these networks. The final section of the chapter is devoted to reviewing current research regarding finding efficient elimination orders.

2.1 Notation

We denote random variables with capital letters (e.g., X, Y, Z), and sets of variables with boldface capital letters $\mathbf{X} = \{X_1, \ldots, X_n\}$. Each random variable X has a discrete domain $\mathcal{D}(X) = \{x_1, \ldots, x_k\}$ of finite size $|X|$. An instantiation of a variable X, which is the assignment of variable X to a value x in its domain, is denoted X=x, or x for short. The assignments of values to the variables in a set are denoted by boldface lower case letters. A *context* over a set of variables $\mathbf{X} = \{X_1, \ldots, X_k\}$ is the conjunction of an instantiation of each variable in $\mathbf{X}$, and is denoted **X=x** or simply **x**. The set of all possible contexts over a set $\mathbf{X}$ is denoted as $\mathcal{D}(\mathbf{X})$. The size of a set $\mathbf{X}$ is denoted by $|\mathbf{X}|$.

P(X) is used to denote the probability distribution for a variable X. We use *P(X = x)* or *P(x)* to denote the probability that X takes the value x. We use $P(\mathbf{X}|\mathbf{Y})$ to denote the conditional probability of $\mathbf{X}$ given $\mathbf{Y}$, and $P(\mathbf{X}|\mathbf{Y} = \mathbf{y})$ denotes the conditional distribution of $\mathbf{X}$ given $\mathbf{Y} = \mathbf{y}$. Here $\mathbf{X}$ and $\mathbf{Y}$ are two different sets of random variables.

2.2 Bayesian Networks

A Bayesian network [41, 28] is a compact representation of a joint probability distribution over a finite set of random variables $\mathbf{X}= \{X_1, \ldots, X_n\}$. It contains a qualitative part and a quantitative part. The qualitative part is a directed acyclic graph (DAG) $\mathcal{G}$, and describes the structure of the network. Each node in $\mathcal{G}$ represents a random variable and the directed edges represent (in some sense) informational or direct causal dependencies among the variables. We use the terms *node* and *random variable* interchangeably. The parents of node X, denoted by $Pa(X)$, are all of the nodes in $\mathcal{G}$ that have directed edges terminating at X. The quantitative part describes the strength of these relations, using a conditional probability. Each variable X in a Bayesian network is associated with a conditional probability distribution (CPD), denoted by $P(X|Pa(X))$. If a variable X has no parents, then it is a prior probability of X, denoted by $P(X)$. The CPD for a variable X is often specified explicitly in tabular format by listing the probability distribution over the values of X given every possible instantiation of its parents $Pa(X)$. In this case, the CPD is called a *conditional probability table* (CPT).

Let P be a joint probability distribution over $\mathbf{X}= \{X_1, \ldots, X_n\}$. We can express P in terms of conditional probabilities as follows:

$$P(X_1, \ldots, X_n) = P(X_n|X_{n-1}, \ldots, X_1)P(X_{n-1}, \ldots, X_1) \tag{2.1}$$

which can be recursively factorized into:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|X_{i-1}, \ldots, X_1) \tag{2.2}$$

The graph $\mathcal{G}$ represents a set of conditional independencies: every node in the graph is independent of its non-descendants given its parents. We assume that the sequence $X_1, \ldots, X_n$ constitutes a topological ordering[1] of the graph $\mathcal{G}$ (when $\mathcal{G}$ is a DAG, it is always possible to find a topological ordering). Let us consider any factor $P(X_i|X_{i-1}, \ldots, X_1)$ from Equation 2.2. The conditional independence assumption from above implies that:

$$P(X_i|X_{i-1}, \ldots, X_1) = P(X_i|Pa(X_i)) \tag{2.3}$$

After applying the conditional independence assumption defined in Equation 2.3 to all factors in Equation 2.2, the joint probability distribution $P(X_1, \ldots, X_n)$ can be represented as follows:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \tag{2.4}$$

Thus Bayesian networks offer a compact representation of a joint probability distribution.

Example 2.1 *Consider an example Bayesian network "*Diagnosis*", shown in Figure 2.1 (modified from [39]) that models lung cancer. The domain has six Boolean variables: Bronchitis (B), Lung Cancer (L), Smoking History (S), Pollution (P), Fatigue (F), and Positive Chest X-Ray (X). The graph structure of a Bayesian network reflects the causal structure of the domain. Lung Cancer and Bronchitis are more often attributed to smokers; this fact is represented by the causal arcs from S to L and B. Other relationships are as follows: Pollution can increase the probability of Lung Cancer, both Lung Cancer and Bronchitis can cause Fatigue, and a person with Lung cancer usually has a positive chest X-ray.*

Figure 2.2 portrays a set of possible conditional probability tables for the Bayesian

[1] A sequence $X_1, \ldots, X_n$ of nodes in a graph is a topological ordering if the following condition is satisfied: if $X_i \rightarrow X_j$ is an edge in the graph, then X_i precedes X_j in the sequence.

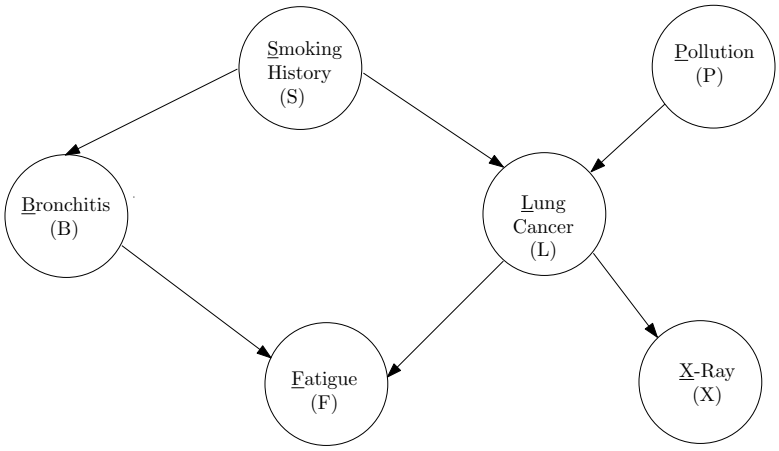


Figure 2.1: A small Bayesian network "Diagnosis".

S	
true	*false*
0.2	0.8

P	
true	*false*
0.8	0.2

S	B	
	true	*false*
true	0.25	0.75
false	0.05	0.95

S	P	L	
		true	*false*
true	*true*	0.05	0.95
true	*false*	0.03	0.97
false	*true*	0.02	0.98
false	*false*	0.001	0.999

B	L	F	
		true	*false*
true	*true*	0.75	0.25
true	*false*	0.10	0.90
false	*true*	0.5	0.5
false	*false*	0.05	0.95

L	X	
	true	*false*
true	0.9	0.1
false	0.2	0.8

Figure 2.2: Conditional probability tables (CPT) for "*Diagnosis*" Bayesian network.

network illustrated in Figure 2.1. The entries in the CPT encode the parameters of the probability distribution. For example, $P(S = false) = 0.8$, $P(L = false|S = true, P = false) = 0.97$.

The example in Figure 2.2 and 2.1 demonstrates how Bayesian networks can compactly represent a joint probability distribution. To represent the joint probability without any independence assumption, $P(S,\ B,\ L,\ P,\ F,\ X)$ would require a table of $2^6 = 64$ values. Recall the conditional independence assumption that every node in the graph is independent of its non-descendants given its parents. For example, in the Bayesian network shown in Figure 2.1, variable F is not independent of S; however, it is independent of S given its parents L and B. Using the independence assumptions, $P(S,\ B,\ L,\ P,\ F,\ X)$ can be expressed in terms of the factorization of the joint as: $P(S)P(B|S)\ P(L|S,P)\ P(F|B,L)P(P)P(X|L)$ that requires tables totaling only 28 values. From above, we can see that the size of these tables grows exponentially with the number of parents, where before the size of the (joint) table grew exponentially with the number of nodes in the Bayesian network.

2.3 Inference

Bayesian networks are a knowledge representation tool used to present information about a problem domain in which there is uncertainty. A common task in reasoning with Bayesian networks is the calculation of the *posterior probability* of a query event, or a posterior distribution over a set of *query* variables, given a set of observations on *evidence* variables. Computing the probability of an event in a Bayesian network is a task often referred to as *probabilistic update*, or *probabilistic inference*.

Evidence is a collection of findings on variables in a Bayesian Network. *Evidence* is an observation, or set of observations, that a variable is in a particular state. For example, if a patient has an allergy, $Allergy = True$ would be an example of evidence. Note that this is

an example of *hard evidence*. Other types of evidence exist, such as *soft evidence* [46, 34] and *virtual evidence* [40]. Briefly, these types of evidence assign a probability distribution (soft evidence) or likelihood ratios (virtual evidence) to the domain values of a variable. For example, suppose that a doctor is only 80% sure that his/her patient has an allergy; a probability of 80% would be assigned to (Allergy=True). This type of evidence is a generalization of hard evidence, where evidence commits a variable to a specific value. Soft evidence and virtual evidence are intended to handle situations where an observation for a variable is not strictly one value or another, but a distribution over values. These are less popular in practice, and applying the techniques of this thesis to soft and virtual evidence is considered as future work.

Given a Bayesian network $\mathcal{B}$ over the variables $\mathbf{X}$, we typically are interested in computing the single value $P(\mathbf{Q} = \mathbf{q}|\mathbf{E} = \mathbf{e})$ or the distribution $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$, where $\mathbf{Q} \subseteq \mathbf{X}$ are called query variables, and $\mathbf{E} = \mathbf{e}$ is our evidence. For example, given the Diagnosis network in Figure 2.1, one might be interested in the probability that a patient has Lung Cancer in light of a positive chest X-ray result, $P(L|X = True)$. A common method for absorbing evidence in Bayesian networks involves replacing every conditional probability table $P(X_i|Pa(X_i))$ with a new table (denoted by $P(X_i|Pa(X_i))_{\mathbf{E}=\mathbf{e}}$), which contains only rows that are compatible with evidence, and only variables that are not in the evidence set. An example of such a new table is shown in Figure 2.3.

Formally, after inserting evidence our joint probability factorization becomes as below:

$$P(X_1,\ldots,X_n)_{\mathbf{E}=\mathbf{e}} = \prod_{i=1}^{n} P(X_i|Pa(X_i))_{\mathbf{E}=\mathbf{e}} \tag{2.5}$$

B	L	F	$P(F\|B,L)$
$true$	$true$	$true$	0.75
$true$	$true$	$false$	0.25
$true$	$false$	$true$	0.10
$true$	$false$	$false$	0.90
$false$	$true$	$true$	0.50
$false$	$true$	$false$	0.50
$false$	$false$	$true$	0.05
$false$	$false$	$false$	0.95

The initial CPT for $P(F|B,L)$

B	L	$f(B,L)_{F=false}$
$true$	$true$	0.25
$true$	$false$	0.90
$false$	$true$	0.50
$false$	$false$	0.95

A new table after observing $F = false$

Figure 2.3: An example of creating a new table due to observing an evidence *F=False*.

In our evaluations, we will use this method of evidence absorption. However, other methods exist for absorbing evidence such as zeroing out those rows in the joint probability distribution that are incompatible with evidence [11], or the use of auxiliary tables that contain only 0s and 1s, known as *lambda* (λ) functions [41].

Let $P(\mathbf{Q}|\mathbf{E}=\mathbf{e})$ be the conditional probability distribution over the query variable $\mathbf{Q}$, given the evidence $\mathbf{E}=\mathbf{e}$. Let $\{Y_1,\ldots,Y_s\}$ be the non-query, non-evidence random variables $\in \mathbf{X}$. The query $P(\mathbf{Q}|\mathbf{E}=\mathbf{e})$ can be computed as follows [32, 11]:

$$P(\mathbf{Q}|\mathbf{E}=\mathbf{e}) = \frac{\sum_{Y_s}\cdots\sum_{Y_1}\prod_{i=1}^{n}P(X_i|Pa(X_i))_{\mathbf{E}=\mathbf{e}}}{\sum_{\mathbf{Q}}\sum_{Y_s}\cdots\sum_{Y_1}\prod_{i=1}^{n}P(X_i|Pa(X_i))_{\mathbf{E}=\mathbf{e}}} \tag{2.6}$$

In Equation 2.6, the denominator can be computed from the numerator, hence, we need only to compute $\sum_{Y_s}\ldots\sum_{Y_1}P(X_1,\ldots,X_n)_{\mathbf{E}=\mathbf{e}}$, which is a function of $\mathbf{Q}$:

$$f(\mathbf{Q}) = \sum_{Y_s}\ldots\sum_{Y_1}\prod_{i=1}^{n}P(X_i|Pa(X_i))_{\mathbf{E}=\mathbf{e}} \tag{2.7}$$

The conditional probability $P(\mathbf{Q}|\mathbf{E}=\mathbf{e})$ can be computed as follows:

$$P(\mathbf{Q}|\mathbf{E}=e) = \frac{f(\mathbf{Q})}{\sum_{\mathbf{Q}} f(\mathbf{Q})} \tag{2.8}$$

where $\sum_{\mathbf{Q}} f(\mathbf{Q})$ is a normalizing constant that normalizes the values of $f(\mathbf{Q})$ so that they sum to 1. Thus, the problem of probabilistic inference reduces to the problem of computing $f(\mathbf{Q})$. Theoretically, $f(\mathbf{Q})$ can be computed from the joint probability distribution by summing out non-query variables one by one. Summing out a variable from a joint probability distribution is not practical, and there have been many algorithms designed to compute posterior probability distributions without expanding CPTs to a full joint representation. In the next section, we discuss two popular exact inference algorithms.

2.4 Some Common Techniques for Inference

A number of inference techniques are available for computing probabilities from Bayesian networks: *clustering* algorithms [35, 29, 28] such as *Junction Tree Propagation* (JTP), *query-based* algorithms [48, 13, 14] such as *Variable Elimination* (VE), and *conditioning* algorithms [10, 36, 24] such as *Recursive Conditioning*; each has particular unique advantages. Clustering algorithms efficiently compute posterior distributions over multiple variables simultaneously, query-based algorithms are able to apply query-specific optimizations, while conditioning algorithms have flexible runtime-memory tradeoffs. These techniques have much in common, and thus their runtime and memory requirements are similar. In particular, most of the modern inference techniques depend on an *elimination ordering* over its variables, which is discussed in detail later in this chapter. The goal of this thesis is to produce efficient variable orderings, that are applicable to *any* inference technique that relies on such elimination orders. We evaluate the performance of elimination

orders using query-based and conditioning algorithms, which are briefly described here.

2.4.1 Variable Elimination

Variable Elimination (VE) [48, 13, 14] is a popular query-based inference method in which variables are dynamically marginalized from the network one at a time. Recall that when computing the query $P(\mathbf{Q}|\mathbf{E} = \mathbf{e})$, we need to compute the function $f(\mathbf{Q})$ (see Equation 2.7). The problem of probabilistic inference is thus reduced to the problem of summing out variables from a product of functions.

Variable Elimination begins by creating a pool of factors, which initially contains the CPTs of the Bayesian network. For example, for the Diagnosis network we have this pool of factors initially: $\Phi = \{\phi_S(S),\ \phi_P(P),\ \phi_L(L,\ S,\ P),\ \phi_B(B,\ S),\ \phi_X(X,\ L),\ \phi_F(F,\ B,\ L)\}$. The non-query variables of the Bayesian network are dynamically marginalized one by one. The order in which the variables are eliminated is called the *elimination ordering*. A variable is removed by multiplying together all the factors that include the variable in its definition, and then marginalizing that variable from the distribution. The result is a new factor that does not include that variable – this is returned to the pool of available factors. This process is repeated until all non-query variables have been marginalized. When all variables have been removed, the product of the remaining factors is our probability of interest.

For example, suppose we apply the VE algorithm to compute $P(X)$. We will use the elimination ordering $\{S,\ P,\ L,\ B,\ F\}$. To eliminate S, we remove $\phi_S(S),\ \phi_L(L,\ S,\ P)$, and

$\phi_B(B,\ S)$ factors from the pool, and compute a new factor by multiplication and summation.

$$\zeta_S(S,\ L,\ P,\ B) = \phi_S(S) \cdot \phi_L(L,\ S,\ P) \cdot \phi_B(B,\ S)$$
$$\psi_S(L,\ P,\ B) = \sum_S \zeta_S(S,\ L,\ P,\ B)$$

The new factor $\psi_S(L,\ P,\ B)$ is returned to pool Φ. To eliminate P, we now compute:

$$\zeta_P(L,\ P,\ B) = \phi_P(P) \cdot \psi_S(L,\ P,\ B)$$
$$\psi_P(L,\ B) = \sum_P \zeta_P(L,\ P,\ B)$$

We can eliminate the rest of the non-query variables in the same way (not shown here for brevity). A full example is shown later, when we will discuss elimination orders.

2.4.2 Conditioning Algorithms

Conditioning algorithms are another inference technique, based loosely on reasoning by cases. There are different types of conditioning algorithms such as Cutset Conditioning [41, 42, 16], and Recursive Decomposition [8, 36, 10, 11]; we focus on Recursive Decomposition methods.

Recursive Decomposition is a class of query-based conditioning algorithms for computing probabilities of interest from Bayesian networks [8, 36, 12]. Recursive Decomposition decomposes the network into smaller and smaller problems. These smaller problems are then solved, and their results combined to compute a global solution. Recursive Decomposition algorithms typically have an associated recursive tree structure. The leaves and internal nodes of this tree correspond to the CPTs and variables of a Bayesian network, respectively. This tree is structured such that all CPTs containing variable V_i in their domain

are contained in the subtree of the node labeled with V_i. The tree structures of recursive decomposition algorithms differ from technique to technique, and each has a specific set of properties. For example, a *dtree* [10, 11] is a full binary tree – all non-leaf nodes have exactly two children. By contrast, elimination trees (*etrees*) have no restriction on the number of children each node can have, although each internal node must be labeled by exactly one variable [24, 25]. Figure 2.4 shows one of the possible etrees for the Bayesian network of Figure 2.1.

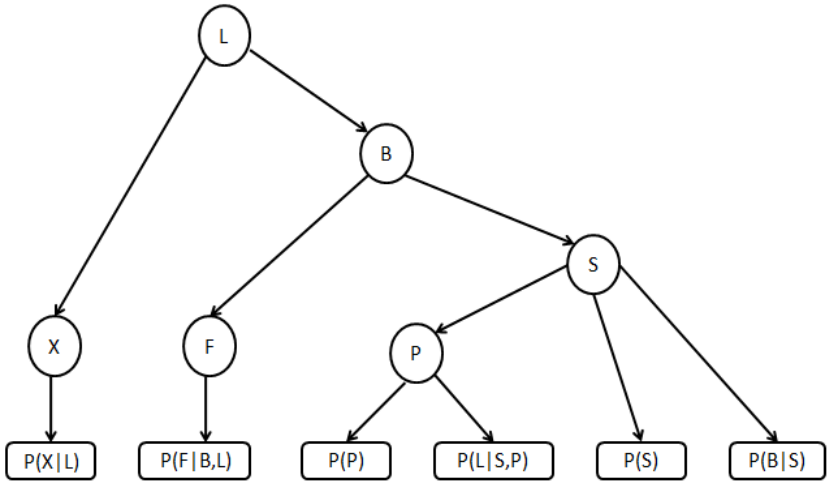


Figure 2.4: The network from Figure 2.1, arranged in an etree following an elimination ordering $\{X, P, F, S, B, L\}$.

To compute a probability from a recursive decomposition, we traverse it depth-first, generating a context as we descend by conditioning over unobserved variable such that when we reach a leaf node, we can extract a probability from the corresponding CPT based on the generated context. The primary difference with VE is that Recursive Decompositions only pass single values around rather than entire factors. Hence, this entails multiple recursive calls, which dominate runtime.

Recursive Decomposition algorithms permit a straightforward time-space tradeoff. By storing previously computed values at each node (called *caching* [10, 30]) we avoid recom-

putations, and thus reduce run time at the cost of memory. However, caching is memory intensive, and we can greatly reduce memory requirements by recomputing rather than storing, which is appropriate in space-constrained environments [10, 11].

Recursive Decompositions have a close correspondence with variable elimination algorithms [48, 14]. In fact, one can construct an etree directly from a variable ordering by using a variant of Variable Elimination [23]. The algorithms for building recursive decompositions parallel variable elimination. In particular, for etrees, an internal node represents the marginalization of its variable label, and the children of the node represent the distributions that would be multiplied together [24]. Thus, an internal node is labeled with a variable, but represents a factor. Better elimination orders produce etrees with faster computation and small memory requirements.

2.5 Finding Efficient Elimination Orderings

The central idea of this dissertation is that the efficiency of most inference algorithms (such as VE, Recursive Decomposition, etc.) is dependent on a variable elimination ordering. As an example, consider again the network of Figure 2.1; suppose a user is interested in the posterior probability distribution over the variable X-ray (X), given that the patient suffers from Fatigue (i.e., F= True). To compute the query $P(X|F = True)$, we need to compute

$$P(X|F = True) = \frac{1}{\mathcal{Z}} \cdot \sum_{\mathbf{X}-\{X,F\}} P(S)P(P)P(L|S,P)P(B|S)P(F = True|B,L)P(X|L)$$

where $\mathcal{Z} = P(F = True)$ and $\mathbf{X}$ represents the union of all variables in the given Bayesian network. Rather than performing an entire recombination of the joint distribution, we will use the VE algorithm. If we choose the elimination order $\{P,\ S,\ L,\ B\}$, then we get the behavior of Table 2.1. In the table, the third column lists the variables that are involved

in multiplications to create intermediate multiplication factors. For example, the first intermediate multiplication factor is $\zeta_P(L,\ P,\ S)$. The last column of the table lists the new factors, which are constructed at each step.

Variable Eliminated	Factors Used	Variables Involved	New Factor
P	$\phi_P(P)$, $\phi_L(L,S,P)$	$L,\ P,\ S$	$\psi_P(L,\ S)$
S	$\phi_S(S)$, $\phi_B(B,S)$ $\psi_P(L,\ S)$	$B,\ L,\ S$	$\psi_S(B,\ L)$
L	$\phi_X(X,L)$, $\phi_F[F=true](B,L)$, ψ_S(B, L)	$B,\ L,\ X$	$\psi_L(B,\ X)$
B	$\psi_L(B,\ X)$	$B,\ X$	ψ_B(X)

Table 2.1: A run of VE for the query $P(X|F=True)$.

Variable Eliminated	Factors Used	Variables Involved	New Factor
L	$\phi_X(X,L)$, $\phi_F[F=true](B,L)$, $\phi_L(L,S,P)$	$B,L,\ P,\ S,\ X$	$\psi_L(B,\ P,\ S,\ X)$
B	$\phi_B(B,S)$, $\psi_B(B,\ P,\ S,\ X)$	$B,\ P,\ S,\ X$	$\psi_B(P,\ S,\ X)$
S	$\phi_S(S)$, $\psi_B(P,\ S,\ X)$	$P,\ S,\ X$	$\psi_S(P,\ X)$
P	$\phi_P(P)\psi_S(P,\ X)$	$P,\ X$	ψ_P(X)

Table 2.2: Another run of VE for the query $P(X|F=True)$.

Notice that no intermediate factor computed during this process contained more than 3 variables. The efficiency of this process is affected by the order in which the variables are selected to be marginalized. For instance, suppose the order $\{L,\ B,\ S,\ P\}$ was used, the first marginalization took place over a factor of five variables as shown in Table 2.2. Because the runtime and memory requirements depend on the size of the tables, choosing a bad ordering effects the table size and can increase the computation cost of the VE algorithm substantially.

As mentioned, the other algorithms are also affected by variable orderings. For example, the cluster size in Junction Tree Propagation and cache sizes in Recursive Decomposition algorithms depends on the elimination orderings. Hence, it is very important to compute efficient orderings whenever possible.

2.5.1 Elimination Orders

We start with some notation and terminology that are required to define *elimination orders* and some of its properties, which are frequently used throughout this dissertation. The notation that is used is based on Darwiche [11].

We denote an undirected graph by G. A graph G is a pair (V,E), where V is a finite set of vertices and E is a set of edges, represented as pairs of vertices. Two vertices $u,v \in V$ are said to be adjacent (or neighbours) if and only if $(u,v) \in E$. We denote by $N_G(v) = \{u \in V | (u,v) \in E\}$ the set of vertices which are adjacent to v in G. A set of vertices $C \subseteq V$ of graph is a clique if it induces a complete subgraph of G. A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. We denote a clique by $G[C]$. A maximal clique in a graph G is a clique $G[C]$ such that there exists no vertex set $C' \supset C$ for which $G[C']$ is a clique.

A vertex u in G is called *simplicial* if $N_G(u)$ induces a complete graph in G (i.e., $N_G(u)$ is a clique) [20]. The process of making $N_G(u)$ a complete subgraph of $G(V,E)$ and removing u and its incident edges from G, is called *eliminating* vertex u from G. Let $G_i = (V_i, E_i)$, $i = 1,\ldots,n$ denote the graphs obtaining from $G_1 = G$ by eliminating a vertex at each step. The sequence of vertices that is generated by this process is called an *elimination order* [20, 11], and typically is denoted by ρ .

Definition 1 *Let G be a graph and U be a vertex in G. The result of eliminating vertex U from graph G is another graph obtained from G by first adding an edge between every pair of nonadjacent neighbors of U and then deleting vertex U from G with its incident edges. The edges that are added during the elimination process are called* ***fill edges*** *[11].*

Definition 2 *The elimination of vertices from graph G according to order* ρ *induces a graph sequence* G_1, $G_2,\ldots,$ G_n *where* $G_1 = G$ *and graph* G_{i+1} *is obtained by eliminating*

node $\rho(i)$ *from graph* G_i *. Moreover, the elimination process induces a clique sequence* $G[C_1], \ldots, G[C_n]$ *where* $G[C_i]$ *consists of vertex* $\rho(i)$ *and its neighbors in graph* G_i *[11].*

Figure 2.5 depicts a graph associated with the corresponding graph and clique sequence induced by the elimination order $\rho = \{T, S, R, Q, P\}$.The width of an elimination order can be defined in terms of the largest clique it induces.

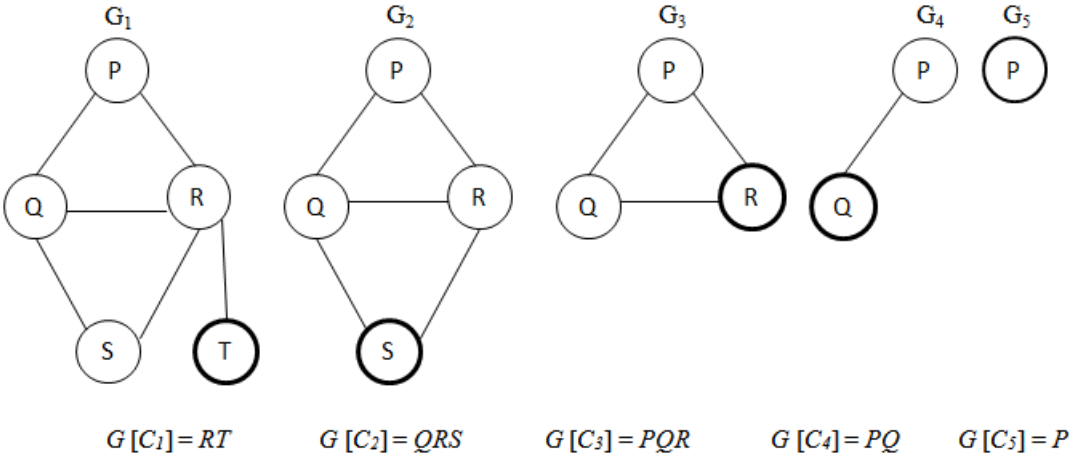


Figure 2.5: The graph with corresponding clique sequence induced by elimination order $\rho = \{T, S, R, Q, P\}$.

Definition 3 *Let* ρ *be an elimination order for graph G and let* $G[C_1], \ldots, G[C_n]$ *be the clique sequence induced by applying the elimination order* ρ *to graph G. The* induced width *of elimination order* ρ *with respect to graph G is [11]:*

$$\textit{induced width}(\rho, G) = \max_{i=1}^{n} |G[C_i]| - 1.$$

Consider Figure 2.5, the induced width of elimination order ρ is 2 since the largest clique it induces contains three variables. Also we can define the *treewidth* of a graph as the width of its best elimination order.

Definition 4 *The treewidth of a graph G is [11]:*

$$treewidth(G) = \min_{\rho} width(\rho, G),$$

where ρ *is an elimination order of graph G.*

When the induced width of an elimination order ρ equals the treewidth of graph G, that order is optimal.

The induced width of an elimination order relates to the complexity of different inference algorithms. For example, the runtime and space complexity of the VE algorithm are $O(n\ exp(w_\rho))$, where n is the number of variables in the Bayesian network, and w_ρ is the induced width of the variable elimination ordering, equivalent to the number of variables in the largest intermediate factor. Similarly, the cache space required by the ElimTree algorithm is $O(n\ exp(w_\rho))$,where w_ρ is the induced width of the variable ordering used to construct the elimination tree [23, 30]. The time required to compute a value from an elimination tree with full caching is $O(n\ exp\ (w_\rho))$ [23, 30]. Note that $exp(w)$ means exponential on some quantity, not e^w, following the notation of Darwiche [11]. A well-chosen elimination order reduces induced width as well as cache size, and hence improves the cost of computation using VE and Recursive Decompositions.

2.5.2 Common Methods for Elimination Orders

Unfortunately, finding the optimal elimination order that produces the best runtime is NP-hard [31]. However, a variety of approximation algorithms exist that find reasonable orders in a feasible amount of time. A popular choice is based on an algorithm designed to *triangulate* an undirected graph such that the cliques in the resulting graph are of minimal size. Computing elimination orderings in Bayesian networks via the triangulation process can

be described as follows [32]:

1. *Moralize* the Bayesian network by adding an edge between any two nodes that share a common child and do not already have an edge between them.

2. Drop the direction of any existing arcs, creating an undirected network.

3. Choose a node N that minimizes a particular cost function $cf()$ (described below).

4. Remove the node N from the network, append it to the current variable ordering, and add edges (called *fill edges*) between any two neighbours of N that are not currently connected.

5. Repeat previous two steps until all nodes are removed from the network.

Algorithm 2.1 Triangulation $(\mathcal{B}, cf())$

Input: $\mathcal{B}$: A Bayesian network, *cf()* : A cost evaluation function
Output: An elimination order ρ of network variables

1: Convert DAG $\mathcal{G}$ of $\mathcal{B}$ into a moral graph as previously described, (i.e., an undirected graph $G = (V, E)$)
2: **foreach** i= 1 to $|V|$ **do**
3: Select a vertex v of graph G as $v \leftarrow \arg\min_{v \in V} cf(v)$, breaking ties randomly
4: Set $\rho(i) \leftarrow v$
5: Add an edge between every pair of non-adjacent neighbours of v in G
6: Eliminate v from the graph G
7: **end foreach**
8: **return** ρ

Algorithm 2.1 illustrates this triangulation method. The quality of orders produced by the triangulation approach depends on the cost function $cf()$ of Step 3. Four cost functions that are commonly cited are:

1. *Minimum Neighbours (MinN)* [31, 32]. The cost of a node is the number of neighbours it has in a graph.

2. *Minimum Weight (MinW)* [31, 32]. The cost of a node is the product of the domain sizes of each of its neighbouring nodes.

3. *Minimum Fill (MinF)* [31, 32]. The cost of a node is the number of new edges that will be added as a result of removing that node.

4. *Weighted Minimum Fill (WMinF)* [20, 32]. This cost function is similar to MinF, with the exception that each new edge has a weight associated with it. The weight of an edge is the product of the domain sizes of its incident nodes. The total cost is computed as the sum of the weights of the added new edges.

The best choice of cost function is network-dependent, however [32] suggests that MinF and WMinF tend to work better in general, and WMinF in particular when domain sizes vary significantly. While other elimination techniques exist (discussed next), triangulation based on greedy cost heuristics continues to be popular [32], and hence will constitute the focus of this thesis.

2.5.3 Related Research

Much research has been done on the problem of finding low treewidth triangulations or (equivalently) good elimination orderings. Arnborg and Proskurowski show that the problem of finding minimal treewidth elimination orders is NP-hard [2]. Several practical algorithms exist for finding low treewidth triangulation.

One of the earliest algorithms is *Lexicographic Breadth-First Search* (Lex-BFS) [43], which is a simplification of a recursive BFS (*Breadth First Search*) algorithm to recognize the *chordal* graphs[2] in linear time [43]. They proved that Lex-BFS applied to a graph

[2] An undirected graph is *chordal* if it contains no cycle of length greater than 3; that is, every minimal loop in the graph is of length 3.

produces a perfect elimination ordering (PEO) if and only if the graph is chordal. Their chordal graph recognition algorithm simply applies Lex-BFS and then determines if PEO was produced, which can be done in linear-time.

In the standard description of the Lex-BFS algorithm [43], vertices are assigned labels that are updated each time a neighbour is visited. Each label is a string of digits, and to decide which vertex to visit next, the algorithm picks an unvisited vertex whose label is largest according to the standard lexicographic order, hence the name of the algorithm. Later Bretscher et al. [7] presented a simple linear time implementation of Lex-BFS using partition refinement that replaces the queue of vertices of a standard breadth-first search with an ordered sequence of sets of vertices. In this approach, a partition of the unnumbered vertices is maintained and refined each time a vertex v is visited. At each step, the vertex v from the first set in the sequence is removed from that set, and if that removal causes the set to become empty then the set is removed from the sequence. Then, each set in the sequence is replaced by two subsets: the neighbors of v and the non-neighbors of v. The subset of neighbors is placed earlier in the sequence than the subset of non-neighbors. Each vertex is processed once; each edge is examined only when its two endpoints are processed.

Though Lex-BFS was generally ignored for a decade after it was first discovered, it has recently experienced a surge in popularity, and become a standard tool for working with restricted classes of graphs. Typically, this algorithm is used as part of other graph algorithms such as the recognition of *chordal graphs* and *optimal coloring* of distance-hereditary graphs.

Lexicographic Breadth-First Search, variant minimal (Lex-M) [43] is a variant of Lex-BFS. Lex-M works similarly to Lex-BFS, assigning labels to vertices, iteratively selecting a vertex with lexicographically maximum label, and appending a digit to the labels of nearby vertices. The distinction is that Lex-M uses the vertex labels to deduce where the fill edges would be added, and appends appropriate labels to both real neighbours (in

the original graph) and would-be neighbours had we included those fill edges. Lex-M basically mimics a Lex-BFS of the filled graph. Rose, Tarjan and Lueker [43] showed Lex-M could efficiently compute minimal elimination ordering (MEO) and its associate minimal triangulation.

In 1984 Tarjan and Yannaskakis presented *Maximum Cardinality Search* (MCS) [45, 33, 32] as a simplified implementation of Lex-BFS and its applications for recognition of chordal graphs, testing acyclicity of hypergraphs, and how to selectively reduce an acyclic hypergraph to more efficiently compute database queries. The MCS algorithm is similar to Lex-BFS but uses a single integer as a vertex label instead of a string. Simply stated, MCS iteratively chooses an unnumbered vertex, which has the maximum number of already numbered neighbours, hence the name of the search. We can track how many neighbours have been visited for each vertex by attaching an integer label to each vertex. MCS computes a PEO of a chordal graph that introduces no fill edges. However, it is not guaranteed to produce minimal triangulation (i.e., triangulation in $O(\text{nm})$ time where $n = |V|$, the number of the nodes and $m = |E|$, the number of edges) as Lex-BFS.

For non-chordal graphs, a chordal graph can be constructed by adding edges during the MCS algorithm. One of the variants of this heuristic is *Weighted Maximum Cardinality Search* (WMCS) [19] in which the cost of a vertex is the product of weights of visited neighbours. Though MCS and WMCS can be used for non-chordal graphs, it turns out that the orderings, generated by these heuristics are not as good as those produced by other heuristics [32, 19].

Recently Berry et al. [4] have introduced a new algorithm called MCS-M. This is a simplification of Lex-M so that cardinality weights are used instead of lexicographic labels. This algorithm computes a minimal elimination ordering and a minimal triangulation of a graph. In many cases, MCS-M can produce triangulations with less than half the fill of other minimal triangulations. But like Lex-M, MCS-M does not always perform well. For

example, for the graph representing an $N \times N$ square grid, MCS-M produces exactly the same fill as Lex-M does, and as pointed out in [43], the minimum fill for such graphs is $O(n^2 \log n)$ whereas the fill produced by Lex-M and MCS-M is $O(n^3)$ [4].

There are also algorithms for choosing elimination orders that target Bayesian networks specifically, rather than general graphs. Shoikhet and Geiger [44] describe a relatively efficient algorithm, known as "QUICK TREE" for finding optimal elimination ordering – one whose cost of inference is approximately the same as the cost of inference with resulting order. This is the first known algorithm that can optimally triangulate only smaller size (up to 100 nodes) graphs in a reasonable time frame. This algorithm adopts the dynamic programming algorithm of Arnborg [2] with slight modifications. It is a two phase algorithm. At the first step, the given graph is decomposed into smaller parts by generating *cliques* and *separators*[3] of minimal size. In the second phase, these smaller parts are repeatedly combined into optimal triangulations of smaller parts, obtaining optimal triangulations of larger ones. This combination operation is called *simultaneous composition*. While this algorithm has interesting theoretical properties, it still is only practical for graphs of reduced size. Becker and Geiger [3] present another algorithm that finds close to optimal ordering. However, this algorithm also is restricted only to those graphs whose domain size is small.

As mentioned, many implementations continue to use one of the standard heuristic methods, using cost functions described in Subsection 2.5.2. A good survey of these heuristic methods is presented by Koller and Friedman [32] as well as Kjaerulff [31], who also provides an extensive empirical comparison. For finding elimination orderings, the greedy search described earlier can be used either deterministically (as shown in Algorithm 2.1) or stochastically. In the stochastic variant [20, 19], at each step some number of low-scoring nodes are selected, and a single node is selected using their score (where lower-scoring

[3]Given a connected graph G with vertex set $V = V(G)$, a subset $S \subset V$ is called a vertex separator for non-adjacent vertices v' and v'' in $V \backslash S$ if v' and v'' are in different connected components of $G[V \backslash S]$.

nodes are selected with higher probability). In the stochastic variants, multiple iterations of the algorithm occur, and the ordering that leads to the most efficient elimination is selected – the one where the sum of the sizes of the factors produced is the smallest. Stochastic techniques can also be used in combining heuristics, although the cost is multiple runs.

Fishelson and Geiger [20] suggest the use of stochastic search as a heuristic and provide a set of comprehensive experimental comparisons focusing on the problem of genetic linkage analysis. A novel modification of the minimum-fill heuristic [31] has been introduced in their paper where the cost of eliminating a vertex is the sum of weights of edges that need to be added due to its elimination. They focused on the constrained elimination problem, where the weight of the heaviest clique created is below a certain threshold. The stochastic greedy heuristic uses a deterministic greedy search in each of the iterations, which finds a constrained elimination ordering. If the cost of the currently obtained elimination sequence is smaller than the cost of the best previously found sequence, then this new sequence and its cost would be recorded. In comparison to Lex-P, Lex-M, and MCS, this heuristic reduces the size of largest cliques.

A new heuristic, *Maximum Minimum Neighbourhood Weight* (MMNW) [47] has been proposed to obtain a lower bound of weighted treewidth. This heuristic repeatedly removes the vertex with the minimum neighborhood weight until the graph is empty; and reports the maximum of the minimum neighborhood weights, seen during the process.

The cost of triangulation can be reduced by pre-processing steps. Bodlaender et al. [5] provided a series of simple pre-processing steps for Bayesian networks to reduce the cost of triangulation; this cost reduction is especially effective for hard combinatorial problems. A set of "*safe reduction*" rules are applied as a pre-processing step. Experiments with these pre-processing rules revealed that the graphs of some well-known Bayesian networks can be triangulated optimally just by pre-processing. In addition, it has also been shown that some algorithms such as Lex-P, Lex-M, and MCS tended to yield better results for graphs

that are reduced by pre-processing. Eijkhof et al. [47] presented a variant of safe rules for preprocessing weighted graph problem and demonstrated their effectiveness on a well known class of Bayesian networks for genetic linkage analysis problems.

Fishelson et al. [19] devised a new algorithm for finding combined order for variable elimination and conditioning for both haplotyping and likelihood computations. The algorithm is composed of two stages. First, a set of reduction rules are applied on the graph as a preprocessing step. Second, several stochastic-greedy algorithms are applied sequentially to determine an elimination order for the residual graph. They observe from their experimental results that for this kind of observation optimization problem, two rules (i.e., *Simplicial* and *Almost Simplicial* rules), are more effective to use. The run time of these rules is negligible compared to the total run time for finding an elimination order, and by reducing the size of the graph, each of the iterations of the stochastic-greedy algorithms applied in the later phase is shorter.

Algorithms that perform state-space searches have also been devised. One of the DFS-based algorithms [22] is marked by its modest space requirements and anytime behavior. Another algorithm, based on best-first search, which has a better time complexity, but consumes more space, was presented by Dow and Korf in 2007 [17], where they showed that the best-first search with a max-consistent heuristic is admissible (a solution is guaranteed to be optimal) on max-cost problems like finding elimination orderings. It was later discovered that the admissibility of this algorithm is strongly order preserving. Additionally, Dow and Korf incorrectly stated that the heuristic used by BestTW was max-consistent, when, in fact, it is not [17]. One variant of this algorithm with a modified maximum edge cost function was highlighted in a later paper [18].

2.6 Summary

This chapter introduced Bayesian networks and associated notation. It included a brief overview of inference algorithms, including those that we will use to test the heuristics presented in this thesis. A detailed description of the elimination order problem was illustrated in this chapter. We also presented a comprehensive review of the related literature to understand the current state of research on elimination orderings.

Chapter 3

Application-Specific Knowledge Base Elimination Algorithms

In Chapter 2, we discussed how inference algorithms for Bayesian networks are affected by elimination orderings over their variables. The observation of these variables can change the network connectivity, which in turn may affect elimination order algorithms. We may know that certain variables in the network are likely or unlikely to be observed, and this information can be used in offline computation to generate elimination orders. In this chapter, we demonstrate techniques to exploit this application-specific knowledge – *probability of observation* – at compile-time to compute such elimination orders that work well over all cases on average, rather than a specific case. Sections 3.1 and 3.2 discuss the probability of observation concept. Section 3.3 outlines a novel *thresholding* approach designed to exploit the probabilities of observations on standard elimination algorithms. However, the primary contribution of this dissertation is in Section 3.4 where we modify existing elimination algorithms to exploit this application-specific information.

3.1 Observation

When a variable is part of the evidence, we say that it is observed. It is well known that one can condition a Bayesian network on its evidence before performing inference [48]. Specifically, observing a variable instantiates it to a particular value, and has the effect of removing its outgoing edges in the network. This reduces network connectivity, resulting in smaller cutset widths, and eliminates the evidence nodes from the CPTs, resulting in fewer marginalizations. Therefore, the observation of a variable in Bayesian network can affect the efficiency of inference algorithms.

Recall that an edge between a parent and child in a Bayesian network means that the parent is a conditioning variable in the child's local distribution. Hence, if an edge exists from X_i to X_j, then $X_i \in Pa(X_j)$. Observing event $X_i = x_i$ effectively creates a new factor in which X_i is not a part of the domain. Since the factor is no longer defined over X_i, the edge between X_i and X_j can be considered to be removed. As an example, consider a simple Bayesian network which contains only three variables P, Q, and R:

$$P \longrightarrow Q \longleftarrow R$$

By observing P, the factor of R now depends only on Q, and the network effectively becomes:

$$P \qquad Q \longleftarrow R$$

By observing R, the factor of P now depends only on Q, and the network effectively becomes:

$$P \longrightarrow Q \qquad R$$

By applying observations to a Bayesian network, the network structure can be effectively changed, which in turn affects the time and memory requirements of inference. However, elimination orders are typically computed in advance under the assumption of no observations, since we may be uncertain about which variable will be observed. Because observations reduce network connectivity, and because the triangulation algorithms from the previous chapter depend on network connectivity, we may end up with a suboptimal elimination ordering *after* the evidence variables are known. If we knew in advance which variables will be observed, then this information could be exploited to generate better elimination orders. For example, if we knew that P was always going to be observed, then the elimination order could be chosen using the pruned network and obtain a more appropriate elimination ordering.

3.2 Probability of Observation

Typically, observations remain unknown until runtime. However, while we may not know that a variable will be assigned in advance, we may know with high certainty that it *will* be observed. Examples of these include the presence of a contagion in blood for a medical diagnosis system, and a gauge's reading in car diagnosis. However, these observable variables may not always be assigned a value. For instance, a sensor variable in a fault detection network may fail from time to time, during which time it offers no reading. Because of this, we cannot exploit this information in advance and thus must either: 1) compute elimination orders on a query-by-query basis at runtime or 2) choose a single elimination order in advance. Computing elimination orders for each query at runtime is the approach that produces the most efficient inference times; however, this adds a substantial amount of overhead to the problem. The added overhead is even worse for algorithms like junction tree propagation that use a secondary data structure. These secondary structures have to be reconstructed when a new elimination order is chosen. As mentioned, when the single elimination order is chosen in advance, it is typically generated without any consideration of future observations. In other words, it is computed under the assumption that no variables are observed, as this is the worst-case scenario, computationally.

Although we may not know with certainty whether a variable will be observed, previous data and/or expert analysis may be able to provide us with the *probability* that a particular variable will be observed. For example, in a patient monitoring system, the output for a blood pressure monitoring device might be unknown 2% of the time due to mechanical failure. Thus, the likelihood of observing this variable in the network is 98%. Several other hypothetical examples of this were considered in Chapter 1.

We define the *likelihood of observing a variable* in the network as the *probability of observation*. If this probability of observation is adequately high, then it seems reasonable

to choose an ordering assuming the variable will be observed (and hence has no outgoing edges), as this will be the most probable case.

The research question that we address in this thesis is: *" Assume the existence of accurate probabilities of observation over a set of variables $\boldsymbol{E}_p$ in our Bayesian network. Can we choose variable elimination orderings that, on average, improve inference runtime and memory requirements? If yes, how?"* In other words, we would like to generate elimination orders in advance that work well in the expected case rather than the worst case.

As a more concrete example, we consider a small network named "Student- Scholarship" (modified from [32]), shown in Figure 3.2a. The problem domain of this network is described briefly in Example 3.1.

Example 3.1 *The Scholarships & Student Finance Office at the University of Lethbridge asks for scholarship applications from current graduate students. The goal of this application is to provide funding to those students who are both talented and enthusiastic about research. However, there is no direct way to test a student's talent and enthusiasm. For that reason, the scholarship office also requires a student's transcript and GRE test result. The student's talent is clearly correlated both with his/her grades and GRE score. The enthusiasm in research depends on willingness to study, and reflected indirectly by whether a student enrolls in advanced courses. A student's grade also depends on talent and hardness of the course. The Scholarship office also asks for a reference letter. This letter is dependent on their performance on courses as well as the assessment about student's enthusiasm in research. Good grades and obtaining this research award affects the cheerfulness of the student.*

This problem can be represented as the Bayesian network shown as Figure 3.1. The problem domain is modeled with eight random variables: Willingness (W), Hardness (H), Talent (T), Scholarship (S), Letter (L), GRE (R) and Grade (G). All variables except G, L,

and R are binary valued. G takes five values representing the grades A, B, C, D, and F. Similarly variable L takes four values representing Excellent, Good, Fair, and Poor. The ternary value of variable R represents High, Medium, and Low scores on the GRE exam.

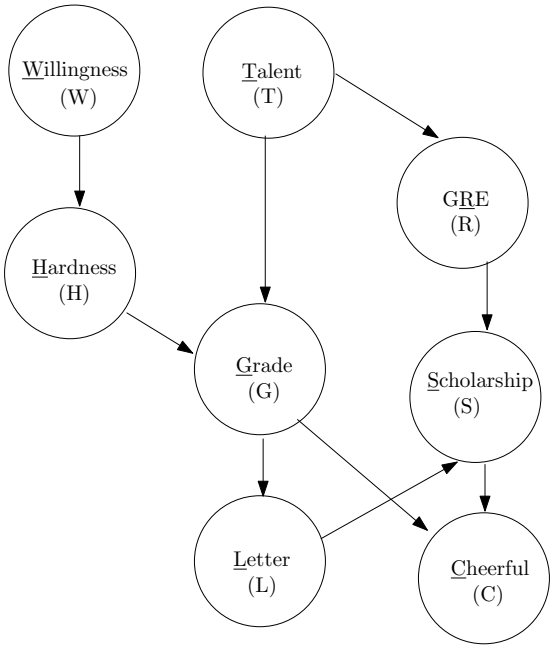


Figure 3.1: The "Student-Scholarship" Bayesian network.

Building on our previous discussion, *Grade* and *GRE* seem like variables that are likely to be observable. However, *Grade* of course may be unknown for a certain subset of students whose transcripts are missing or may not be applicable due to unfamiliar grading structures of foreign universities. Similarly, *GRE* may also be unknown for those whose test result is outdated or missing. From the previous annual case analysis, an approximation of seeing this kind of applicant can be determined.

One of the assumptions that we make for our proposed approaches is stated here: *"the probability of observation for a variable is independent of the probability of observation for any other variable"*. This assumption significantly simplifies our forthcoming computations, and the algorithms we propose here perform well even under this assumption.

Without this assumption, we must model dependence between these probabilities of observation, and the result is basically a secondary probabilistic model, whose complexity is potentially greater than the primary model over which we compute currently. We consider modeling dependencies between the probabilities of observations as future work.

3.3 Thresholding

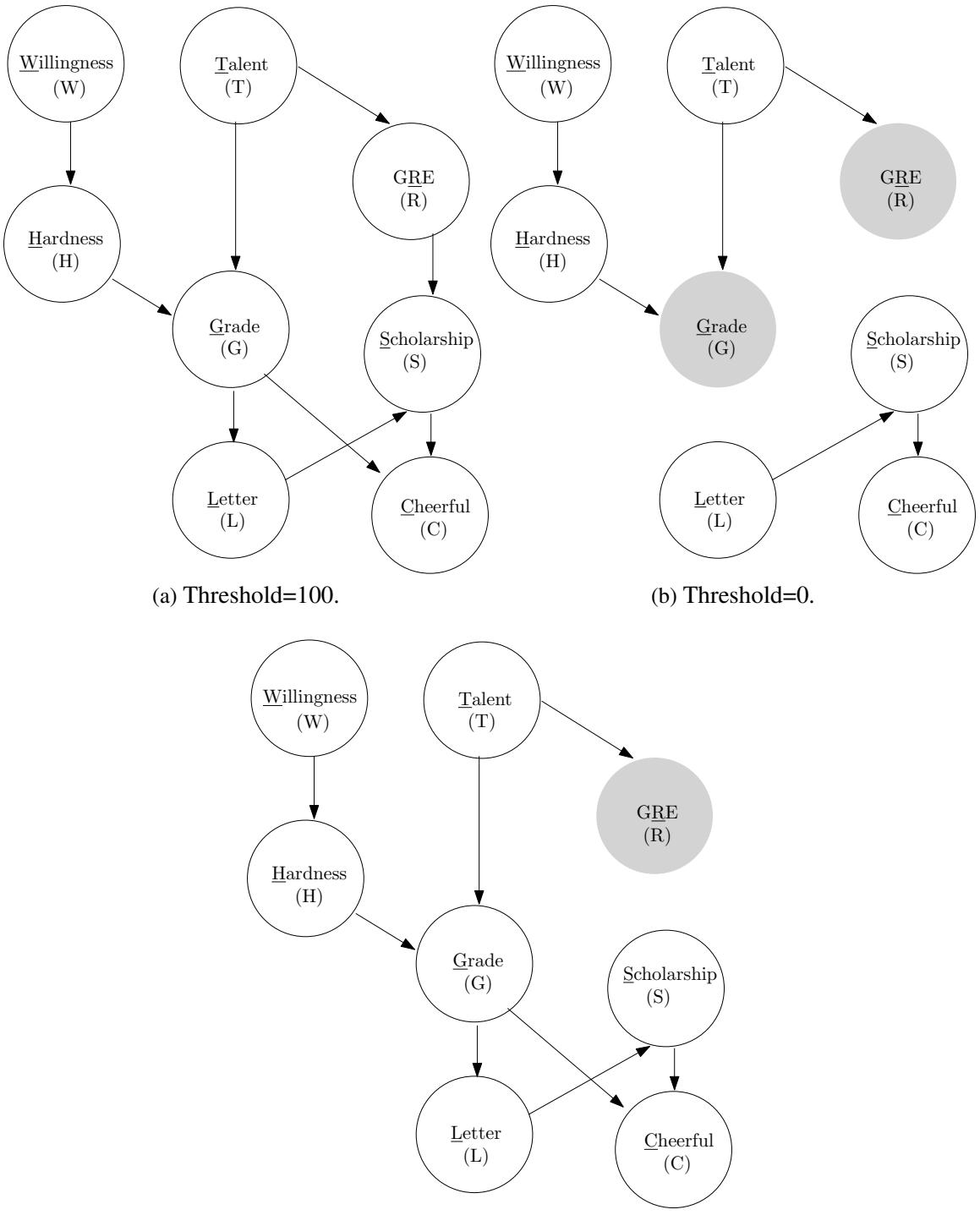
To exploit the probability of observation for producing elimination orders, we first consider a simple approach, referred to henceforth as *Thresholding*. This approach commits variables in the potential observation set $\mathbf{E}_p$ to either being observed or unobserved for the purposes of computing a variable ordering.

Let $\mathbf{E}_p = \{X_0, \ \ldots, \ X_{m-1}\}$ in $\mathcal{B}$, and suppose the probability of observation of the variables in the potential set is $\{\alpha_1, \ldots, \alpha_{m-1}\}$ respectively, where $\alpha_i > 0$. We define a threshold T whose value is between 0% and 100%. If the probability of observation of a variable is greater than T, we assume that it is observed when computing a variable ordering. We consider three possibilities for T as shown below:

1. ***T*=100%.** That is, simply we assume that no variable is observed, which is basically the approach that the standard heuristic algorithms currently take. We will refer to this as the ***standard*** approach.

2. ***T*=0%.** At the other extreme, another possibility is simply to assume that every variable is in the potential observation set that can be observed (i.e., its probability of observation is $> 0\%$) is observed, and compute the ordering that way.

3. ***T*=50%.** That is, we assume a variable to be observed if it is more likely to be observed than not.



(a) Threshold=100.

(b) Threshold=0.

(c) Threshold=50.

Figure 3.2: Effects of thresholding over "Student-Scholarship" network.

We do not consider other thresholds here, as they are meant merely as a comparison to a more sophisticated technique, presented in the next section.

As an example of the differences that these thresholds can make, consider the student scholarship network of Fig.3.2a. Suppose further that our potential observation set is $\{G,R\}$ and the probability of observation for each variable is 48% and 52% respectively. The standard approach, which assumes no observations ($T = 100\%$), computes an elimination order of $\rho_{100} = \{W, H, C, S, T, G, R, L\}$, using MinF and breaking ties randomly. If we assume that all variables in the potential set are observed ($T = 0\%$), then we compute over the network shown in Figure 3.2b, and the same heuristic results in an elimination order of $\rho_0 = \{L, W, R, C, G, T, S, H\}$. Of course, the values of these observations are not known at compile-time, and hence the independence between the observed variable and its children will only occur at runtime. However, since it is at runtime that the inference operation will take place, we choose an ordering that optimizes for this situation. Furthermore, the independence between the variable and its children depends only on the variable being observed, and not what the observed value is. If we use this elimination order ρ_0, the expected number of multiplications performed using VE is 115.66, while using the elimination order ρ_{100} it is 119.15.

Threshold $T = 50\%$ is of particular interest. We would expect this threshold to work well, as it uses the most probable state (observed/not observed) of the potential observation set when computing its elimination ordering. The result of applying this threshold over our example network is shown in Figure 3.2c. Applying MinF on this network, we obtain an order, $\rho_{50} = \{L, C, W, H, R, T, S, G\}$, and the expected number of multiplications using VE is 112.55.

Note that the example elimination orders were obtained by breaking ties randomly, and other orders were possible. If we average the number of multiplications over 12 different computed orderings from each threshold approach (breaking ties randomly each time), we

obtain costs of 113.0, 110.0, and 108.2, respectively for thresholds of 100%, 0%, and 50%, respectively. In the forthcoming chapter, we will compare the results of these three thresholds on a set of benchmark Bayesian networks.

3.4 Weighted Edge Elimination Algorithms

The thresholding algorithm of the preceding section was a simple approach to exploit the probability of observation. This approach generalizes traditional algorithms and we obtained elimination orders from an altered network. However, it is a coarse technique. In this section, we present a different method, that utilizes a modification to the graph structure, as well as updated elimination heuristics.

Consider again the simple 3-variable network:

$$P \longrightarrow Q \longleftarrow R$$

Figure 3.3: A simple 3-variable Bayesian network.

Suppose that our potential observation set is P and R, and the probability of observing each variable is 40% and 85%, respectively. Recall that observing a variable effectively removes its outgoing edges. Since both P and R have a non-zero probability of being observed, then all four network structures shown in Figure 3.4 are possible.

$P \longrightarrow Q \longleftarrow R$
Neither P nor R observed (9% of time)

$P \quad Q \longleftarrow R$
Only P observed (6% of time)

$P \longrightarrow Q \quad R$
Only R observed (51% of time)

$P \quad Q \quad R$
Both P and R observed (34% of time)

Figure 3.4: Four possible query specific network structures.

Under our assumption of independence, we can compute their relative frequencies, shown in each caption.

$$P \xrightarrow{0.60} Q \xleftarrow{0.15} R$$

Figure 3.5: Network structure with edge weights.

In Figure 3.5, the probability assigned to each edge shows the chance of edge existence at runtime. Therefore, the edge between P and Q would exist at least 60% of the time, while the edge between Q and R would exist 15% of the time.

The standard algorithms generate elimination orders based on connectivity of the graph. These algorithms are not designed to consider the probability of edge existence. We propose modifications to the standard algorithms that account for these probabilities when considering their local neighborhoods. Recall that the operation of standard heuristics depends on two major issues: a cost function for choosing the next vertex for elimination and a procedure for removing the selected one from a graph. We will modify each of the four (MinN, MinF, WMinF, and MinW) cost functions of standard heuristics as well as vertex removal procedure. These modified algorithms will henceforth be collectively referred to as the *Weighted Edge Elimination Algorithms* (WEEA).

3.4.1 Pre-Processing

As in the standard approach, we apply a set of pre-processing steps to a Bayesian network, with the aim of making it ready to exploit the probability of observation. We introduce a new pre-processing step to assign weights to each edge of the graph. The goal of these pre-processing steps is to model the weighted version of the graph to take advantage of the probability of observation in the network.

1. Assign Weights over Edges: Consider a Bayesian network $\mathcal{B}$ which consists of n random variables, $\mathbf{X}_n = \{X_0, \ldots, X_{n-1}\}$. Let $\mathcal{G}$ be the DAG of the Bayesian network $\mathcal{B}$. Let $\mathbf{E}_p$ be a set of the potential observable variables $\mathbf{E}_p = \{E_0, \ldots, E_{m-1}\}$ where $\mathbf{E}_p \subseteq \mathbf{X}_n$. Let X_i be a random variable, where $X_i \in \mathbf{E}_p$ is observed with the probability α_i.

As an example consider the following simple network, where n=3. Suppose that our potential observation set is X_1 and X_2, and the probability of observing each variable is α_1 and α_2, respectively.

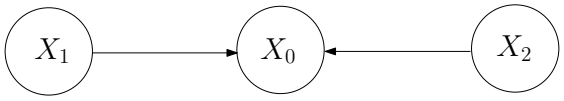


Figure 3.6: A simple Bayesian network.

Recall that observing a variable effectively removes its outgoing edges. Hence, the arc between X_1 and X_0 exists $(1-\alpha_1)$ of the time, while the arc between X_0 and X_2 exists $(1-\alpha_2)$ of the time. The probability of the existence of an edge is considered as an *edge weight*. Therefore, we assign $(1-\alpha_1)$ and $(1-\alpha_2)$ as edge weights. Thus, the DAG $\mathcal{G}$ of the given Bayesian network of Figure 3.6 transformed into a weighted DAG, $\mathcal{G}$ is shown in Figure 3.7.

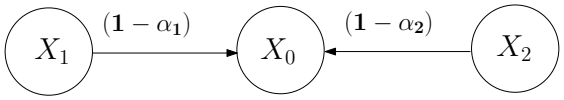


Figure 3.7: After assigning weights on edges.

2. Convert into Weighted Moral Graph: To obtain a weighted moral graph we apply the following moralization procedure which is modified from the standard algorithms:

a. **Insertion of Moralization Edges:** Recall that a moralization edge is added between any two vertices who share a child in the network if no edge already exists between

them. Therefore, similar to standard heuristics, we first insert moralization edges in $\mathcal{G}$. For example, in Figure 3.7 we see that X_1 and X_2 have a common child X_0. Thus, a moralization edge is added between X_1 and X_2 since there is no existing edge.

In addition, we have to assign weights over the newly added moralization edges. Suppose we consider the existence of moralization edges in a similar way, as we did in the earlier pre-processing step. During moralization, X_1 and X_2 will only be attached by a moralizing edge if neither variable is observed since observations remove their outgoing edges. Recalling our assumption of independence about probabilities of observation, the probability of this moralization edge being added is $(1-\alpha_1)\times(1-\alpha_2)$. Hence, the weight of this moralized edge is $(1-\alpha_1)(1-\alpha_2)$ where α_1 and α_2 are the probability of observing X_1 and X_2, respectively. Figure 3.8 portrays this conversion from weighted DAG $\mathcal{G}$ to weighted moral graph G_{WM}. The moralized edge is shown as a dashed line in this figure. As another example, in the network of Figure 3.5 the weight of a moralization edge between P and Q is $0.60 * 0.15 = 0.09$.

b. **Drop Edge Direction:** The remaining step that WEEA follows is exactly the same as in standard algorithms, by dropping the direction of edges (Figure 3.8).

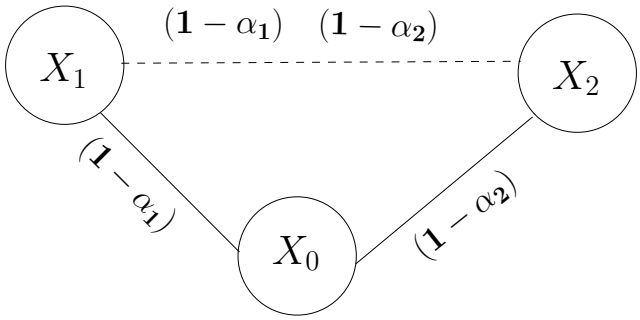


Figure 3.8: Converted into weighted moral graph G_{WM},where a moralized edge is inserted and directions of all edges are dropped.

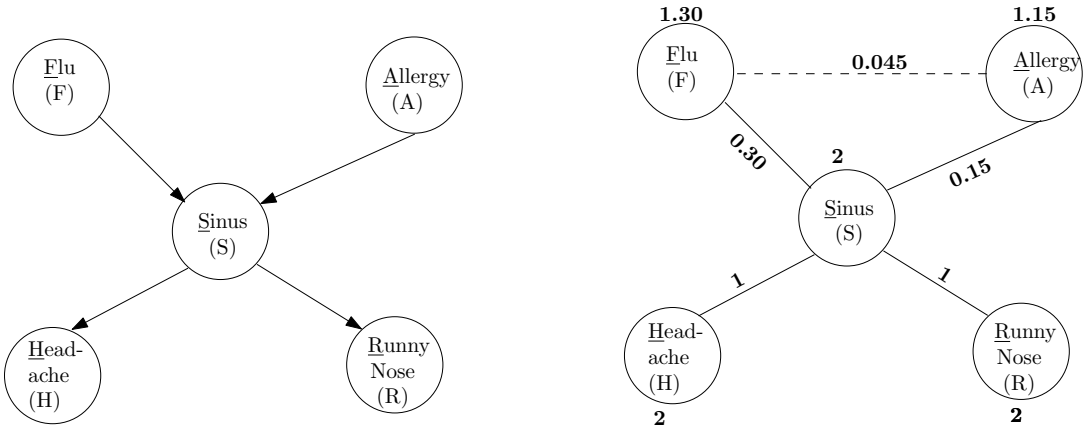
3. Assign weights on vertices: To modify the cost functions of MinW and WMinF heuristics, we perform an additional pre-processing task. Recall that these two heuristics consider the domain size of variables when computing their respective costs. When a variable is observed at runtime, its domain size is effectively reduced to 1, as all values that are not the observed value are no longer part of the problem domain. Hence, using the probability of observation, we can compute what the *expected* size of each variable in the potential observation set will be at runtime and assign that as its *vertex weight*.

Let the domain size of a variable X_i be d_i. If there is chance α_i of observing this variable, then we can compute the expected domain size using the formula:

$$w(X_i) = [1 \times \alpha_i + d_i \times (1 - \alpha_i)]$$

As an example, assuming that each variable of the Figure 3.5 network is Boolean, the expected sizes of P's and R's domain would be $[1 \times (0.4) + 2 \times (0.6)] = 1.6$ and $[1 \times (0.85) + 2 \times (0.15)] = 1.15$, respectively. In the modified MinW and WMinF algorithms, we will use the expected domain sizes, which we have found to give an added performance benefit.

Example 3.2 *A patient has been suffering from strong headaches, so he visits a pharmacy for medication. In that pharmacy, an automated medication system is installed for prescribing medication. He is worried that he may have a sinus infection, which can be caused by Flu or some sort of Allergy. In addition, a sinus problem can cause a runny nose. The Bayesian network of this problem is shown in Figure 3.9(a). Assume that each variable in this network is Boolean.*



(a) The Bayesian Network of sinus infection problem.

(b) The weighted moral graph of sinus infection network after pre-processing.

Figure 3.9: Effects of pre-processing steps.

One of the common scenarios is that a number of patients may not know if they suffer from allergies or flu. Confirmations of having allergies require medical tests that may not have been performed. Also, the flu may be undiagnosed. The probability of seeing each syndrome can be learned from historical data. For the sake of example, suppose that the potential observation set is Flu and Allergy, and the probability of observing each variable is 70% and 85% respectively. After undergoing the mentioned pre-processing steps, the resulting graph is shown in Figure 3.9(b).

3.4.2 The Cost Evaluation Functions

Like standard triangulation algorithms, the aim of each cost function of WEEA is to select a variable in a greedy manner whose elimination cost is the minimum. The general algorithm of WEEA is shown in Algorithm 3.1. The modifications applied to the original triangulation algorithm (Algorithm 2.1) are minor and straightforward. Although many current techniques are more sophisticated than the triangulation approach, we use the latter because they are widely cited and well-known in the BN community. We conjecture

that the presented modifications to the triangulation heuristics could also be made in other elimination order algorithms, and this is left as future work.

Our remaining tasks are to supply modified cost functions (Line 5) and vertex removal (Line 7) for the WEEA approach. In this subsection, we will illustrate the modifications of cost evaluation functions. A vertex removal procedure will be considered in the next subsection.

Algorithm 3.1 WEEA for constructing an elimination order

Input: $\mathcal{B}$: A Bayesian network with n variables, cf: A cost evaluation function

Output: An elimination order $\rho[1,\ldots,n]$ of network variables

1: Convert DAG $\mathcal{G}$ of $\mathcal{B}$ into a weighted undirected graph G, using the pre-processing method described in Subsection 3.4.1

2: $i \leftarrow 1$

3: $G' \leftarrow G$

4: **while** G' is not the empty graph **do**

5: $\quad v_{G'} \leftarrow cf(G')$ // the vertex in G' with the minimum expected elimination cost

6: $\quad \rho[i] \leftarrow v_{G'}$

7: $\quad$ ADD_FILL_EDGES_AND_REMOVE_NODE $(G', v_{G'})$

8: $\quad i \leftarrow i+1$

9: **end while**

10: **return** ρ

---

1. Minimum Neighbours (MinN) : Consider the traditional MinN heuristic, which chooses vertices that have the smallest number of neighbours. Suppose MinN is measuring the cost of vertex Q from one of our previous examples shown in Figure 3.5, assuming no other eliminations have yet taken place. Vertex P and Q will be neighbours 60% of the time, while vertex Q and R will be neighbours 15% of the time. The probability

that Q has two neighbours is $0.6 * 0.15 = 0.09$, the probability that it has one neighbour is $0.6 * 0.85 + 0.4 * 0.15 = 0.57$, while the probability that it has no neighbours is $0.4 * 0.85 = 0.34$. Hence, the *expected* number of neighbours that vertex Q has in the initial moral graph is, $E(Neighbours) = [2 \times (0.09) + 1 \times (0.57) + 0 \times (0.34)] = 0.75$. This number is more easily calculated as the sum of the probabilities associated with each edge that is incident to Q: $E(Neighbours) = 0.60 + 0.15 = 0.75$. It is very straightforward to modify MinN to choose the vertex with the minimum expected number of neighbours. We show the modified version in Algorithm 3.2.

Algorithm 3.2 The Minimum Neighbours (MinN) cost function, $cf(G')$

Input: A weighted undirected graph $G'(V_{G'}, E_{G'}, w_{G'})$

Output: A vertex $v_{G'}$ with the smallest *expected* number of neighbours.

1: **foreach** $v_i \in V_{G'}$ **do**
2: Find the neighbour list $N_{G'}(v_i)$ of vertex v_i
3: $pncount(v_i) \leftarrow \sum_{n_i \in N_{G'}(v_i)} w(v_i, n_i)$
4: $expnbr(v_i) \leftarrow pncount(v_i)$
5: **end foreach**
6: Select $v_{G'} \leftarrow \arg\min_{v_i \in V_{G'}} expnbr(v_i)$, breaking ties randomly
7: **return** $v_{G'}$

2. Minimum Fill (MinF) : MinF can also be modified to choose the vertex that results in the minimization of the expected number of fill edges due to its elimination. Along with our assumption of independence about the probability of observation, we make another simplifying assumption: *the probabilities associated with the edges are independent of one another*. It is clear that this assumption is not valid: the existence of a moralizing edge is certainly dependent on the existence of the other edges. The dependence between edges is omitted as an approximation. The reasons for making this assumption is as before: to main-

tain dependence information between edges requires a sophisticated algorithm, particularly when we update the graph subsequent to the elimination of a variable. As well, the resulting performance of the heuristic is quite good even under this assumption. Maintaining such dependencies is considered a subject of future work.

Algorithm 3.3 The Minimum Fill (MinF) cost function, $cf(G')$

Input: A weighted undirected graph $G'(V_{G'}, E_{G'}, w_{G'})$

Output: A vertex $v_{G'}$ that requires the fewest number of *expected* fill edges to make that vertex simplicial

1: **foreach** $v \in V_{G'}$ **do**
2: $pfillcount \leftarrow 0.0$
3: **foreach** $\{(v', v'') \in (V_{G'} \times V_{G'}) \mid \{(v, v') \in E_{G'} \wedge (v, v'') \in E_{G'} \wedge (v' \neq v'')\}\}$ **do**
4: **if** $((v', v'') \in E_{G'})$ **then**
5: $pfillcount \leftarrow pfillcount + [w(v, v') * w(v, v'') * (1 - w(v', v''))]$
6: **else**
7: $pfillcount \leftarrow pfillcount + [w(v, v') * w(v, v'')]$
8: **end if**
9: **end foreach**
10: $expfilledge(v) \leftarrow pfillcount$
11: **end foreach**
12: Pick $v_{G'} \leftarrow \arg\min_{v \in V_{G'}} expfilledge(v)$, breaking ties randomly
13: **return** $v_{G'}$

Recall that traditional MinF selects that variable whose elimination from the network requires the fewest number of fill edge insertions. Reconsidering the opening example of this section (see Figure 3.5), suppose that Q was being considered for elimination. A fill edge would then be required if (a) P and R were connected to Q and (b) there was not already an existing edge between P and R. Under the independence assumption, we

calculate the probability of added fill as $[0.60 * 0.15 * (1 - 0.09)] = [0.60 * 0.15 * 0.91] = 0.0819$.[1]Hence, the modified MinF heuristic computes the number of *expected* fill edges required by each variable, and picks that one that minimizes that value. This modified cost evaluation function is illustrated in Algorithm 3.3.

3. Weighted Minimum Fill (WMinF) : Recall that in standard WMinF, the weight of an edge is the product of the domain sizes of its incident vertices. Moreover, the total cost of elimination is computed as the sum of the weights of the added new edges. We can use

---

Algorithm 3.4 The Weighted Minimum Fill (WMinF) cost function, $cf(G')$

Input: A weighted undirected graph $G'(V_{G'}, E_{G'}, w_{G'})$

Output: A vertex $v_{G'}$ whose total *expected* weight of *expected* fill edges is the smallest

1: **foreach** $v \in V_{G'}$ **do**

2: $\quad pwfill\_count \leftarrow 0.0$

3: $\quad$ **foreach** $\{(v', v'') \in (V_{G'} \times V_{G'}) \mid \{(v, v') \in E_{G'} \wedge (v, v'') \in E_{G'} \wedge (v' \neq v'')\}\}$ **do**

4: $\quad\quad$ **if** $((v', v'') \in E_{G'})$ **then**

5: $\quad\quad\quad pwfill\_count \leftarrow pwfill\_count + [\{w(v, v') * w(v, v'') * (1 - w(v', v''))\} * \{w(v') * w(v'')\}]$

6: $\quad\quad$ **else**

7: $\quad\quad\quad pwfill\_count \leftarrow pwfill\_count + [\{w(v, v') * w(v, v'')\} * \{w(v') * w(v'')\}]$

8: $\quad\quad$ **end if**

9: $\quad$ **end foreach**

10: $\quad expfillw(v) \leftarrow pwfill\_count$

11: **end foreach**

12: Pick $v_{G'} \leftarrow \arg\min_{v \in V_{G'}} expfillw(v)$, breaking ties randomly

13: **return** $v_{G'}$

---

[1]Here, we already see the independence assumption being violated, as the true probability of a fill edge being required here is in fact 0.0. However, this assumes that we know the relationship between the edges from Q and the moralizing edge between P and R.

the edge existence probabilities in WMinF in the same way as in MinF. Therefore, to compute the cost to eliminate any vertex in the graph we use the *expected domain sizes* like standard algorithms along with these *edge probabilities*.

WMinF can be modified similarly as MinF, to choose the vertex that results in the minimization of the total weight of the expected fill edges. We also recall the same assumption of independence about edge probabilities for it. Algorithm 3.4 describes this modified version of WMinF. The modified WMinF heuristic computes the cost of each vertex in the current graph as the *sum of weights of the expected fill edges* and selects the one that costs the minimum.

4. Minimum Weight (MinW) : Recall that, the cost of a vertex in MinW is the product of the domain sizes of each of its neighbouring vertices. To modify MinW we will use the expected domain sizes.That is, the cost of a vertex is calculated as the product of *expected* domain sizes of each of its *expected neighbours*. The expected domain sizes were already

---

Algorithm 3.5 The Minimum Weight(MinW) cost function, $cf(G')$

Input: A weighted undirected graph $G'(V_{G'}, E_{G'}, w_{G'})$

Output: A vertex $v_{G'}$ with the smallest product of weights of *expected* neighbours

1: **foreach** $v_i \in V_{G'}$ **do**

2: Find the neighbor list $N_{G'}(v_i)$ of vertex v_i

3: $pwprod(v_i) \leftarrow \prod_{n_i \in N_{G'}(v_i)} [w(v_i, n_i) * w(n_i)]$

4: $expnw\_prod(v_i) \leftarrow pwprod(v_i)$

5: **end foreach**

6: Select $v_{G'} \leftarrow \arg\min_{v_i \in V_{G'}} expnw\_prod(v_i)$, breaking ties randomly

7: **return** $v_{G'}$

---

calculated and assigned as vertex weights on the graph at the pre-processing step. Hence, the modified MinW chooses that vertex whose product of the weights of the expected neighbours is the smallest. Algorithm 3.5 illustrates this cost function.

3.4.3 Procedure for Removing a Vertex

Once a variable with minimum cost is chosen for elimination, it needs to be removed from its network. Recall that in the standard approach, when a variable is selected for removal, all neighbouring nodes are connected by a fill edge if one is not present already. Therefore, it remains only to determine how to update the probabilities of edges when a variable is removed.

Because modeling dependencies adds a considerable amount of overhead in the algorithm, we use a simple approximation formula under which our proposed algorithms perform well. Hence, these probabilities of edges are not necessarily true probabilities any longer, and we will henceforth refer to them as *weights*.

This approximation formula calculates the *total* probability of an edge existence between those vertices that were neighbours of the eliminated vertex in the current updated graph. To compute this value we consider the existing edge weight (if already there is an edge) as well as the probability of fill edge insertion. Therefore, the weights are updated as follows: suppose we are eliminating vertex Q of the example network of Figure 3.5, which is connected to node P and R. We compute the new weight of an edge between P and R as the old weight of the edge (w_{old}, which is 0.0 if no edge previously existed) plus the probability of inserting a new filler edge (w_{new}), using the formula that we defined in the new description of $MinF$. The existing edge weight between P and R is 0.09, while the probability of inserting a new edge was computed as $[0.60 * 0.15 * (1 - 0.09)] = 0.0819$. Hence, the new edge weight is 0.1719. Formally, this approximation formula to update the

weight of any fill edge is:

$$w_{update} = w_{old} + w_{new} \tag{3.1}$$

$$where\ w_{old} = 0.0 \text{ if no edge exists already}$$

Algorithm 3.6 ADD_FILL_EDGES_AND_REMOVE_NODE $(G', v_{G'})$

Input: A weighted undirected graph $G'(V_{G'}, E_{G'}, w_{G'})$ and a vertex $v_{G'} \in V$

Output: A filled-in graph G'_{ρ} from which $v_{G'}$ is eliminated and insert fill edges if required and update *weights* of edges

1: **foreach** $\{(v', v'') \in (V_{G'} \times V_{G'}) \mid \{(v, v') \in E_{G'} \ \wedge (v, v'') \in E_{G'} \ \wedge (v' \neq v'')\}\}$ **do**
2: **if** $((v', v'') \notin E_{G'})$ **then**
3: Insert edge between v' and v''
4: $w(v', v'') \ \leftarrow (w(v, v') * w(v, v''))$
5: **else**
6: $w(v', v'') \ \leftarrow w(v', v'') + [(w(v, v') * w(v, v'') * (1 - w(v', v'')))]$
7: **end if**
8: **end foreach**
9: Remove all incident edges from $v_{G'}$
10: Remove $v_{G'}$ from G'
11: **return** G'_{ρ}

We describe the procedure in Algorithm 3.6. The purpose of updating edge weights in this way is to provide cost functions for the updated local neighborhoods approximation in the next iteration; that in turn affects choosing of the vertex with the lowest elimination cost more precisely.

3.4.4 Example

We apply the modified cost functions of WEEA over our example "Sinus" network (Figure 3.9(a)). Example 3.3 presents the elimination process of WMinF.

Example 3.3 *We evaluate the cost of eliminating each vertex of Figure 3.9(b) according to the WMinF (Algorithm 3.4) as follows:*

- $cf(A) = [0.045 \times 0.15 \times (1 - 0.30)] \times (1.30 \times 2) = 0.026393$
- $cf(F) = [0.045 \times 0.30 \times (1 - 0.15)] \times (1.15 \times 2) = 0.012285$
- $cf(S) = [0.30 \times 0.15 \times (1 - 0.045) \times (1.30 \times 1.15)] + [(0.30 \times 1) \times (1.30 \times 2)] + [(0.30 \times 1) \times (1.30 \times 2)] + [(0.15 \times 1) \times (1.15 \times 2)] + [(0.15 \times 1) \times (1.15 \times 2)] + [(1 \times 1) \times (2 \times 2)] = 5.97760$
- $cf(H) = 0$
- $cf(R) = 0$

As the cost of elimination for vertex H and R is equal, we choose one (H) randomly. The vertex H has only one neighbor. Therefore, no edge insertion is required. Again we evaluate the elimination cost of each of the remaining vertices - {A, F, S and R} as follows:

- $cf(A) = [0.045 \times 0.15 \times (1 - 0.30)] \times (1.30 \times 2) = 0.026393.$
- $cf(F) = [0.045 \times 0.30 \times (1 - 0.15)] \times (1.15 \times 2) = 0.012285$
- $cf(S) = [0.30 \times 0.15 \times (1 - 0.045) \times (1.30 \times 1.15)] + [(0.30 \times 1) \times (1.30 \times 2)] + [(0.15 \times 1) \times (1.15 \times 2)] = 0.18925$
- $cf(R) = 0$

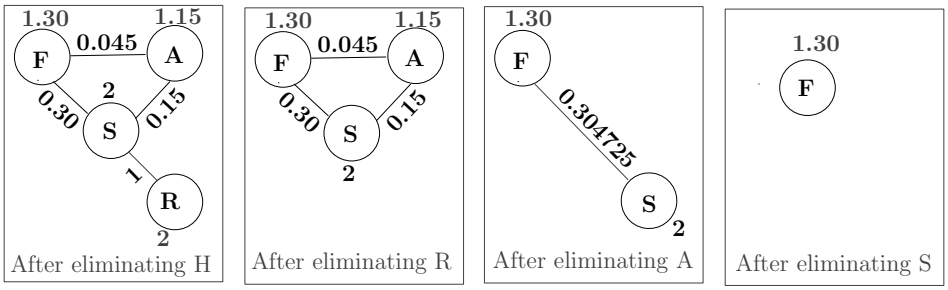


Figure 3.10: Elimination Process over sinus infection network using WMinF.

In this iteration, R is selected for elimination due to its lowest cost. Now in the graph F, A, and S are three remaining vertices and their costs of elimination are 0.026393, 0.012285, and 0.064248, respectively. Hence, A is selected for elimination. Then we update the edge weight between F and S which are adjacent neighbours of deleted vertex A. We calculate the update weight over the edge (F,S) *using the approximation formula (Equation 3.1),* $w_{update(F,S)} = w_{old} + w_{new} = 0.30 + 0.004725 = 0.304725$. *Continuing in this fashion this network has an elimination order* $\rho = \{H,\ R,\ A,\ S,\ F\}$. *Figure 3.10 illustrates this elimination process.*

3.4.5 Features of WEEA

WEEA has several striking features that are illustrated below:

- Maintaining edge weights adds only a constant cost factor to existing algorithms, so doesn't increase the asymptotic complexity of the existing standard approach.
- If the observation set is empty, or (equivalently) the probability of observation for all variables is set to 0%, then this algorithm degrades gracefully to the standard approach.

- The described approach is a simple modification. For that reason, it is easily accessible to general programmers and researchers of other related fields without requiring deep expertise in Bayesian networks.

- Perhaps most significantly, the algorithm performs well in comparison to thresholding approaches. For example, using the modified MinF heuristic, the obtained elimination order is $\rho_{=}$ $\{W, H, C, L, S, R, T, G\}$, and using this order the expected number of multiplications performed by the VE algorithm is 106.64. Since during elimination order generation we break tie situations randomly, when we average the expected number of multiplications, we obtain a multiplication cost of 105.15. And WEEA are much better than traditional algorithms, even less than the T50% approach. Note that we obtain average costs of 113.0, 110.0, and 108.2, respectively for thresholds of 100%, 0%, and 50%, respectively.

3.5 Summary

This chapter described two methods for exploiting the probability of observation to obtain better expected elimination orders. First, we illustrated a technique called thresholding that generalizes standard elimination algorithms. Three variants of this approach were illustrated with examples. We then proposed a more sophisticated approach called WEEA (Weighted Edges Elimination Algorithms). The probabilities of observations were incorporated as weights of edges in WEEA to explore the benefit of inclusion of this application-specific information. Both methods are explained via examples. In the following chapter, we will present an empirical performance analysis of these methods versus standard methods over larger benchmark networks.

Chapter 4

Experiments and Evaluation

In this chapter, we present the results of experimental evaluations comparing thresholding and WEEA to the standard approach. We use Variable Elimination (VE) and Elimination Tree (ElimTree) inference methods for evaluating the performance of the elimination orders produced from these approaches. We choose VE due to its popularity as a query-based inference algorithm among the Bayesian network community. We select ElimTree as a representative of conditioning algorithms and one that uses a secondary data structure that is built from an elimination ordering. All experiments presented in this chapter were run on an Intel(R) Core (TM2) Duo CPU T6500 @ 2.10GHZ processor with 4.00 GB RAM in a Linux environment. We chose the C++ programming language for implementing these algorithms.

4.1 Experiments with Benchmarks

Recall from Chapters 2 and 3 that elimination orders affect the runtime and memory requirements of inference algorithms. In the previous chapter, we presented two approaches in which application-specific knowledge – *the probability of observation* – was exploited to generate elimination orders that improved the expected efficiency of inference algorithms. In this section, we compare their performance using VE and ElimTree. Each approach was tested on a set of well-known standard benchmark networks from the Bayesian Network Repository [1].

The aim of this research work is to find such an elimination order that improves the expected cost of inferences over all possible assignments of the potential observation set. Formally, let *assignments* $(\mathbf{E_p})$ be the set of possible assignments to the potential ob-

servation set, where an assignment maps each variable to either *observed* or *unobserved*. Note how this differs from a *context* over a set of variables, which assigns each variable to a specific value from its domain. Given an elimination order ρ over the variables, let $cost(A, \rho)$ be the runtime cost associated with assignment A given the variable ordering ρ. The expected cost of inference can then be computed using the familiar equation:

$$E(cost(\rho)) = \sum_{A \in assignments(\mathbf{E_p})} P(A) * cost(A, \rho) \tag{4.1}$$

where the probability of an assignment $P(A)$ is simply calculated as the product of the probability of its individual assignments (due to our assumption of independence). For example, suppose that our potential observation set consists of two variables X_1 and X_2 that have an observation probability of 75% and 82%, respectively. There are four possible assignments of this set as shown in Table 4.1.

Possible Cases		Assignments
X_1	X_2	
Observed	Observed	A_1
Observed	Unobserved	A_2
Unobserved	Observed	A_3
Unobserved	Unobserved	A_4

Table 4.1: Four possibilities of observing X_1 and X_2.

The *expected cost* of an elimination order ρ using Equation 4.1 is:

$$\begin{aligned} E(cost(\rho)) = {} & (0.75) * (0.82) * cost(A_1, \rho) + (0.75) * (0.18) * cost(A_2, \rho) \\ & + (0.25) * (0.82) * cost(A_3, \rho) + (0.25) * (0.18) * cost(A_4, \rho) \end{aligned} \tag{4.2}$$

The goal, of course, is to find an ordering ρ that minimizes this expected cost.

The described approach, which computes the expected cost over all possible assignments of the potential observation set and chooses the best one, is straightforward. However, the number of assignments is exponential on the number of variables in the set, so this approach does not scale well for large observation sets, and may simply be infeasible if the set becomes sufficiently large. In this thesis, we consider only simple approximation techniques that incur roughly the same cost as current techniques. Finding efficient methods for computing the optimal assignment is left as future work.

We performed two experiments. The purpose of the first experiment was a comparative analysis of the performance of the standard approach to the thresholding approach and the WEEA method. We compared the expected cost of the elimination orders generated by these methods while they were used by the VE and ElimTree algorithms, respectively. We tested each elimination approach over a variety of benchmark networks for different evidence variable sets with different percentages of probability of observation. To avoid any kind of bias, these evidence sets and their probabilities were chosen randomly. Subsection 4.1.1 deals with the details of this experiment. The aim of the second experiment was to analyze how the performances of those approaches fluctuate with the change of magnitude of the probability of observation. We describe the details of this experiment in Subsection 4.1.2.

4.1.1 Experimental Cost Comparison between Approaches

We compared the elimination orders generated from T100, T0, T50, and WEEA approaches, described in Chapter 3. Each method was tested on 12 standard Bayesian network benchmarks (see Appendix A), ranging from 27 to 1,044 nodes.

For each network, 55 potential observation sets of 3 to 7 variables were chosen randomly, and the probability of observation for each variable in the set was assigned ran-

domly. For each potential observation set, we computed elimination orders using T100, T0, T50, and WEEA approaches using each of the four heuristics – MinF, MinN, MinW, and WMinF, respectively. Because random tie-breaking was utilized, we computed 8 elimination orders for each approach and observation set, for a total of 440 combinations. The expected cost of each order was computed, and averaged over the 8 trials.

The expected costs of VE compared were total multiplication count (T_ζ), total summation count (T_ψ), and runtime required by the VE algorithm to compute the probability of the evidence using the computed orderings, as well as the induced width of the variable ordering (ω_ρ). We chose arithmetic operation (multiplication and summation) counts as the evaluation metrics because they are machine independent. Note that these are a reasonable, but not perfect indication of runtime. Tables 4.2 to 4.5 show the results from these trials using MinF, MinN, MinW, and WMinF heuristics, respectively. The costs are shown as a ratio to T100 to illustrate comparative improvement.

	T_ζ			T_ψ			w_ρ			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.89	0.85	0.82	0.89	0.88	0.87	0.95	0.89	0.87	0.89	0.82	0.69
Barley	0.98	0.72	0.47	1.13	0.79	0.42	0.99	0.96	0.88	1.09	0.83	0.52
Hailfinder	1.06	0.88	0.55	1.03	0.88	0.45	0.99	0.94	0.82	1.12	0.92	0.68
Insurance	0.87	0.83	0.65	0.87	0.78	0.62	0.96	0.93	0.91	0.87	0.84	0.72
Link	1.28	0.81	0.59	1.00	0.84	0.50	1.00	0.94	0.84	1.37	0.81	0.58
Mildew	1.04	0.86	0.53	1.32	0.93	0.55	1.03	0.95	0.85	1.49	0.94	0.48
Munin1	1.99	0.89	0.81	1.45	0.92	0.64	1.09	0.94	0.86	1.26	0.92	0.44
Munin2	1.68	0.85	0.78	1.33	0.84	0.68	0.94	0.85	0.79	1.50	0.70	0.58
Munin3	0.94	0.74	0.67	0.96	0.80	0.59	0.95	0.89	0.86	1.19	0.78	0.66
Munin4	0.89	0.85	0.80	0.92	0.86	0.72	0.97	0.90	0.82	0.97	0.83	0.67
Pigs	1.08	0.95	0.82	1.05	0.93	0.78	1.00	0.99	0.91	1.02	0.93	0.74
Water	1.03	0.70	0.50	1.05	0.76	0.46	1.04	0.93	0.85	1.26	0.97	0.51

Table 4.2: Costs of VE using MinF heuristic. Values shown are ratios to T100.

	T_ζ			T_ψ			w_ρ			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.81	0.77	0.59	0.81	0.70	0.56	0.92	0.90	0.79	0.81	0.76	0.67
Barley	1.14	0.63	0.46	1.10	0.75	0.47	0.98	0.97	0.86	1.39	0.95	0.53
Hailfinder	0.86	0.79	0.56	0.90	0.84	0.63	0.99	0.90	0.85	0.98	0.72	0.67
Insurance	0.81	0.77	0.56	0.83	0.70	0.56	0.99	0.92	0.79	0.89	0.81	0.69
Link	0.93	0.76	0.47	0.89	0.72	0.48	0.95	0.92	0.78	1.03	0.86	0.53
Mildew	1.05	0.80	0.47	1.42	0.94	0.45	1.08	0.93	0.86	1.29	0.92	0.46
Munin1	0.82	0.80	0.67	1.79	0.82	0.66	1.08	0.95	0.84	1.42	0.80	0.65
Munin2	1.05	0.76	0.68	1.00	0.87	0.60	0.96	0.78	0.72	1.35	0.74	0.64
Munin3	2.86	0.79	0.56	2.83	0.80	0.54	1.57	0.93	0.79	1.99	0.83	0.56
Munin4	0.85	0.78	0.59	0.81	0.74	0.55	0.92	0.89	0.79	0.87	0.78	0.65
Pigs	1.02	0.85	0.50	1.11	0.79	0.40	0.94	0.91	0.89	1.05	0.78	0.54
Water	1.33	0.77	0.46	0.94	0.71	0.29	1.01	0.93	0.84	1.17	0.74	0.50

Table 4.3: Costs of VE using MinN heuristic. Values shown are ratios to T100.

	T_ζ			T_ψ			w_ρ			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.81	0.63	0.61	0.83	0.77	0.68	0.89	0.86	0.85	0.87	0.81	0.76
Barley	1.71	0.81	0.66	1.68	0.84	0.65	1.07	0.91	0.77	1.32	0.82	0.62
Hailfinder	1.29	0.84	0.50	1.41	0.91	0.51	1.08	0.83	0.78	1.25	0.87	0.63
Insurance	0.88	0.86	0.85	0.81	0.77	0.67	0.89	0.86	0.85	0.83	0.77	0.68
Link	0.90	0.64	0.59	0.83	0.64	0.52	0.97	0.95	0.88	0.94	0.72	0.55
Mildew	1.43	0.85	0.46	1.71	1.18	0.48	1.05	0.94	0.85	1.53	0.97	0.47
Munin1	1.09	0.68	0.61	1.15	0.84	0.65	0.97	0.94	0.86	1.30	0.96	0.76
Munin2	2.04	0.90	0.80	1.25	0.88	0.72	1.06	0.83	0.78	2.10	1.01	0.70
Munin3	1.84	0.83	0.67	1.75	0.85	0.63	1.09	0.90	0.78	2.88	0.82	0.62
Munin4	0.86	0.69	0.65	0.77	0.73	0.69	0.98	0.91	0.83	0.95	0.84	0.72
Pigs	1.11	0.91	0.63	1.09	0.72	0.58	0.97	0.96	0.85	1.20	0.86	0.57
Water	0.95	0.65	0.50	0.87	0.62	0.44	0.99	0.92	0.87	0.94	0.68	0.51

Table 4.4: Costs of VE using MinW heuristic. Values shown are ratios to T100.

	T_ζ			T_ψ			w_ρ			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.82	0.70	0.57	0.85	0.74	0.54	0.75	0.68	0.64	0.71	0.62	0.44
Barley	1.05	0.79	0.49	1.03	0.80	0.47	1.00	0.93	0.82	1.02	0.83	0.52
Hailfinder	1.02	0.96	0.50	0.96	0.76	0.44	1.13	0.99	0.89	1.06	0.81	0.54
Insurance	0.80	0.75	0.43	0.85	0.79	0.53	0.72	0.74	0.53	0.79	0.62	0.55
Link	0.97	0.74	0.50	0.94	0.74	0.46	1.04	0.93	0.80	1.26	0.97	0.50
Mildew	1.42	0.93	0.43	1.42	0.87	0.36	0.96	0.90	0.78	1.47	0.98	0.46
Munin1	0.99	0.81	0.58	1.45	0.92	0.44	1.09	0.94	0.86	1.26	0.92	0.44
Munin2	2.08	0.93	0.49	2.13	0.86	0.46	1.03	0.80	0.68	2.76	0.97	0.48
Munin3	1.90	0.77	0.47	1.41	0.80	0.47	1.03	0.86	0.76	1.68	0.80	0.54
Munin4	0.88	0.69	0.49	0.77	0.67	0.47	0.92	0.72	0.54	1.01	0.67	0.42
Pigs	2.12	0.87	0.49	2.13	0.92	0.26	1.05	0.92	0.70	2.67	0.96	0.47
Water	0.91	0.64	0.40	0.89	0.62	0.28	0.92	0.89	0.56	0.96	0.72	0.35

Table 4.5: Costs of VE using WMinF heuristic. Values shown are ratios to T100.

For the ElimTree algorithm, the expected costs compared were total recursive calls (T_{RC}), total cache size(T_{Cache}), largest cache size(L_{Cache}), and runtime. We chose RC (which is machine independent) as an evaluation metric to be consistent with other research publications regarding recursive conditioning applications (e.g., see [10],[26]). In addition, we chose cache sizes as the measurement of space savings. Tables 4.6 to 4.9 illustrate the results for ElimTree. Like the VE comparison, the costs are shown as a ratio with T100 in all of these tables.

We make several observations regarding our results from Table 4.2 to Table 4.5. First, regarding the thresholding approaches, T0 typically resulted in worse orderings than the standard approach (T100). A particularly striking example is with the Munin3 network in Table 4.3, where a T0 increased the arithmetic operation counts by over 150%. This suggests that removing an edge incorrectly from a network costs more than keeping an edge mistakenly. Another interesting observation is that sometimes the T0 approach executed additional arithmetic operations and needed additional runtime than the standard approach, even though both of them used the same induced width elimination orders. For example,

in Table 4.2 the Link network executed an additional 28% multiplication operations at the cost of 37% increase in runtime though there was no degradation in induced width of elimination orders obtained from the T0 approach in comparison to T100. This suggests that elimination orderings do not affect the sizes of all intermediate factors in a linear manner. The size of the largest factors might have been unaffected or reduced, but the rest of the factors' sizes were increased.

Our second observation, as predicted, is that the T50 approach produced comparatively better elimination orders than the other thresholds in almost all cases. In some cases, almost 40% runtime improvement was realized.

The most significant observation is that for all tested benchmark networks, the WEEA approach consistently outperformed the thresholded and standard approaches. Using elimination orders obtained from WEEA resulted in the lowest cost across all metrics. In some cases, the total multiplication cost was reduced by over 50% and the total summation cost was reduced by over 70%. The induced width was reduced as well, in some cases by over 40%. The overall runtime improved significantly using the weighted approach, as well in some cases over 55%. Therefore, we propose the general usage of WEEA to obtain elimination orders because they reduce the expected cost of VE more substantially than the other three approaches, as well as speeding up the performance of VE more significantly.

	T_{RC}			T_{Cache}			L_{Cache}			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.89	0.87	0.83	1.06	0.89	0.79	1.14	0.77	0.76	0.97	0.84	0.74
Barley	1.03	0.93	0.68	1.34	0.90	0.72	1.66	0.90	0.70	1.10	0.94	0.67
Hailfinder	1.30	0.82	0.54	1.28	0.87	0.53	0.94	0.82	0.28	1.28	0.80	0.49
Insurance	0.90	0.86	0.75	0.89	0.86	0.35	0.31	0.18	0.16	0.98	0.86	0.65
Link	1.08	0.65	0.60	0.99	0.53	0.48	0.97	0.59	0.36	1.03	0.68	0.47
Mildew	1.33	0.88	0.73	1.65	1.06	0.71	1.60	1.40	0.81	1.37	0.87	0.68
Munin1	1.35	0.84	0.73	2.08	0.90	0.72	1.76	0.93	0.76	1.76	0.95	0.71
Munin2	1.15	0.88	0.77	1.52	0.91	0.73	1.33	0.95	0.67	1.29	0.82	0.67
Munin3	1.42	0.88	0.62	2.05	0.90	0.63	1.76	0.91	0.68	1.46	0.84	0.60
Munin4	0.90	0.78	0.76	0.98	0.91	0.77	0.87	0.86	0.71	0.99	0.73	0.69
Pigs	0.99	0.88	0.74	1.24	1.14	0.76	1.67	1.46	0.76	1.11	0.92	0.73
Water	0.64	0.42	0.36	0.62	0.59	0.39	0.67	0.55	0.37	0.79	0.60	0.41

Table 4.6: Costs of ElimTree using MinF heuristic. Values shown are ratios to T100.

	T_{RC}			T_{Cache}			L_{Cache}			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.89	0.87	0.85	0.70	0.58	0.56	0.81	0.76	0.67	0.89	0.87	0.80
Barley	1.36	0.85	0.72	1.12	0.85	0.76	1.20	1.00	0.77	1.30	0.92	0.66
Hailfinder	1.50	0.82	0.63	1.47	0.94	0.59	1.52	0.95	0.51	1.47	0.83	0.56
Insurance	0.96	0.92	0.39	1.45	0.92	0.74	1.69	0.99	0.43	1.26	0.92	0.38
Link	0.86	0.61	0.52	1.18	0.73	0.57	1.03	0.82	0.47	0.79	0.66	0.46
Mildew	1.48	0.73	0.58	1.35	0.93	0.58	1.71	0.92	0.63	1.32	0.83	0.56
Munin1	1.13	0.78	0.72	1.60	0.89	0.68	1.13	0.92	0.70	1.45	0.75	0.68
Munin2	1.92	0.98	0.76	1.89	1.01	0.77	1.77	1.01	0.78	2.06	0.98	0.70
Munin3	2.00	0.89	0.57	2.12	0.95	0.52	2.23	1.00	0.56	2.01	0.87	0.56
Munin4	0.88	0.81	0.67	0.98	0.86	0.65	0.99	0.86	0.69	0.87	0.79	0.65
Pigs	0.89	0.91	0.50	1.06	0.86	0.59	1.37	0.91	0.53	0.99	0.87	0.59
Water	1.10	0.62	0.50	1.07	0.50	0.43	1.03	0.73	0.37	1.05	0.62	0.45

Table 4.7: Costs of ElimTree using MinN heuristic. Values shown are ratios to T100.

	T_{RC}			T_{Cache}			L_{Cache}			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.90	0.89	0.81	0.83	0.77	0.68	0.87	0.81	0.76	0.90	0.84	0.74
Barley	1.39	0.94	0.71	1.33	0.72	0.70	1.29	0.75	0.64	1.14	0.91	0.68
Hailfinder	1.24	0.88	0.61	1.51	0.87	0.48	1.93	0.91	0.35	1.21	0.91	0.52
Insurance	0.91	0.70	0.66	1.80	1.18	0.65	1.71	1.08	0.77	1.05	0.87	0.76
Link	0.83	0.67	0.58	0.86	0.71	0.57	0.88	0.87	0.50	0.78	0.70	0.47
Mildew	1.27	1.20	0.71	2.18	1.28	0.72	3.10	1.42	0.74	1.60	1.44	0.69
Munin1	1.23	0.95	0.75	1.57	0.99	0.75	1.34	0.99	0.74	1.40	0.94	0.69
Munin2	1.74	1.06	0.72	1.61	1.24	0.82	1.30	1.26	0.80	1.95	1.01	0.60
Munin3	1.17	0.76	0.71	1.17	0.91	0.71	1.08	0.86	0.65	1.27	0.69	0.66
Munin4	1.86	0.84	0.72	2.02	0.95	0.71	2.09	0.95	0.72	1.98	0.83	0.67
Pigs	2.13	0.93	0.62	1.32	0.90	0.62	1.70	0.97	0.62	2.03	0.91	0.58
Water	0.85	0.64	0.50	0.84	0.68	0.51	0.95	0.73	0.44	0.95	0.86	0.46

Table 4.8: Costs of ElimTree using MinW heuristic. Values shown are ratios to T100.

	T_{RC}			T_{Cache}			L_{Cache}			Runtime		
Network	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA	T0	T50	WEEA
Alarm	0.85	0.73	0.66	0.88	0.84	0.80	0.83	0.82	0.81	0.78	0.73	0.64
Barley	0.91	0.85	0.56	1.21	0.68	0.57	1.32	0.79	0.55	0.94	0.84	0.54
Hailfinder	1.06	0.69	0.38	1.24	0.90	0.62	1.20	0.73	0.38	1.06	0.77	0.41
Insurance	0.91	0.88	0.45	1.42	0.87	0.74	1.86	0.81	0.40	0.95	0.98	0.48
Link	0.69	0.46	0.37	0.64	0.60	0.44	0.66	0.63	0.42	0.81	0.59	0.33
Mildew	1.77	0.89	0.57	1.76	0.86	0.66	2.08	0.95	0.61	1.79	0.88	0.55
Munin1	0.98	0.73	0.63	0.94	0.78	0.70	0.92	0.83	0.67	0.97	0.71	0.53
Munin2	1.17	0.79	0.65	1.57	0.75	0.68	1.21	0.92	0.71	1.50	0.90	0.41
Munin3	1.58	0.95	0.53	1.44	0.87	0.66	1.29	0.84	0.70	1.26	0.77	0.51
Munin4	0.87	0.82	0.57	0.93	0.85	0.72	0.89	0.82	0.72	0.93	0.77	0.54
Pigs	1.52	0.83	0.60	1.68	0.96	0.71	2.63	0.99	0.73	1.70	0.92	0.57
Water	0.85	0.62	0.51	1.07	0.77	0.56	1.01	0.82	0.48	0.81	0.66	0.46

Table 4.9: Costs of ElimTree using WMinF heuristic. Values shown are ratios to T100.

Tables 4.6 to 4.9 demonstrate similar performance results of the ElimTree algorithm. Firstly, regarding the thresholding approach, using the T0 approach generally resulted in worse orderings than the use of T100 and T50. Using these elimination orders, the expected cost was increased by an enormous amount in most of the cases, across all metrics. As a result, there was a substantial increase in runtime and memory usage. A remarkable example is the Mildew network in Table 4.8, where the T0 approach boosted the number of recursive calls by 27% which resulted in 60% additional time to execute the queries. In addition, it also increased the memory usage by more than 1.15 times in comparison to the standard approach; similarly, the largest cache became more than double in size.

Secondly, a T50 approach performed better than both the T100 and T0 approaches. The cost of computations was trimmed down in almost all cases. In some cases, the number of recursive calls and runtime were decreased by 50% and 40%, respectively.

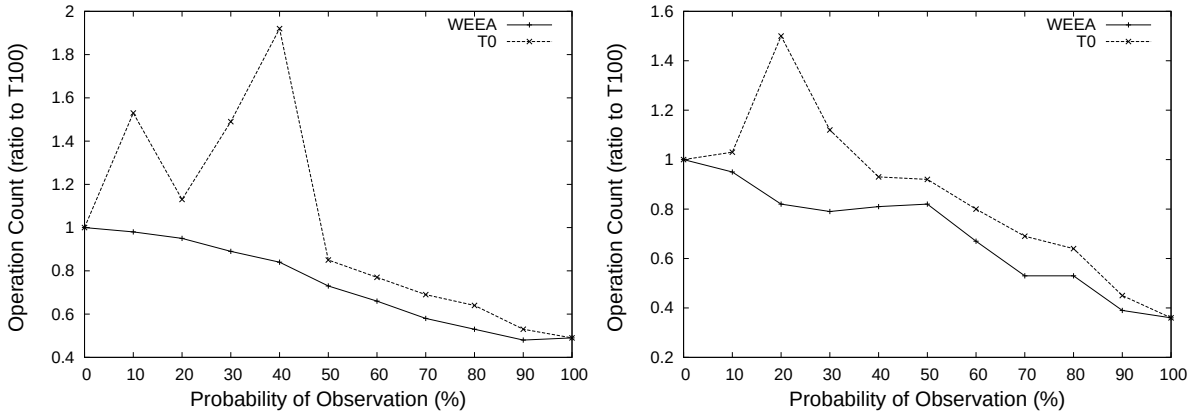
Third, our experimental results demonstrate that the WEEA approach outperformed both the standard and thresholding approaches and gained savings over all evaluation metrics. The cost of the recursive call counts was reduced in all tested cases over all benchmarks, by at least 15%, and up to 64% in comparison to the standard (T100) approach. The overall runtime was reduced as well, in some cases by over 50%.

Fourth, a substantial improvement was gained over the memory usage. Considering T_{Cache} and L_{Cache}, we see that it was always better to use elimination orders obtained from the weighted approach rather than other approaches. The expected total cache size was reduced by at least 10% in all cases, and by over 55% in some cases. The size of the largest cache was decreased as well, from 19% to over 65%.

4.1.2 *Effect of the Probability of Observation on Algorithmic Performance*

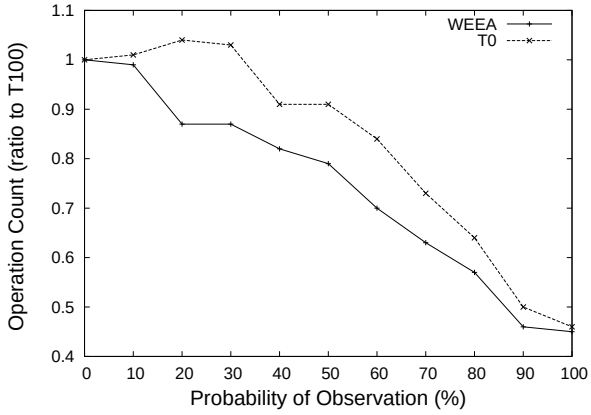
This experiment addresses the question of how the performance of the T0, T100, and WEEA algorithms vary with probability of observation. To do this, we used the same twelve repository benchmarks from the previous experiment. For each network, 20 potential observation sets of 3 to 7 variables were chosen randomly. We varied the probabilities of observations from 0% to 100% over the variables in each observation set. We did not consider the T50 approach in the comparison, because we have used consistent probabilities in our observation set. Therefore, the performance of the T50 approach is equivalent to the T100 when the probability of observation is less than or equal to 50%, and similarly equivalent to the T0 approach when probability of observation is greater than 50%. Using the same testing procedure of the earlier experiments, we recorded the expected total number of arithmetic operations (multiplications and summations) for VE, as well as the total number of recursive calls for ElimTree. The experimental results are plotted in the graphs of Figure 4.1 to Figure 4.14. Since similar results were seen on other networks, their graphs were excluded.

From the shape of these graphs, we make a few observations regarding our results. First, not surprisingly, the standard algorithm (T100) outperforms T0 for low probabilities, while T0 works well for high probabilities. Initially, in all networks, the T0 approach increases the expected cost of VE and ElimTree substantially while the probabilities of observations remain comparatively lower. An example is with the Mildew network in Figure 4.5, where the T0 approach increases the cost of arithmetic operations by over 150% (approximately) due to lower probabilities of observations. As the probabilities of observations are increasing gradually, the T0 approach starts to outperform the standard algorithms; this transition occurs in all networks, although not always starting at 50%.

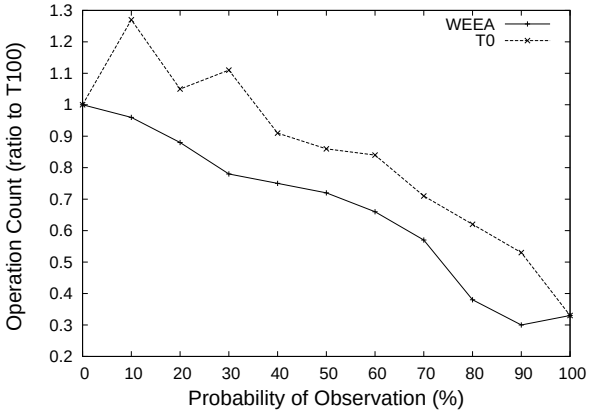


(a) MinF

(b) MinN

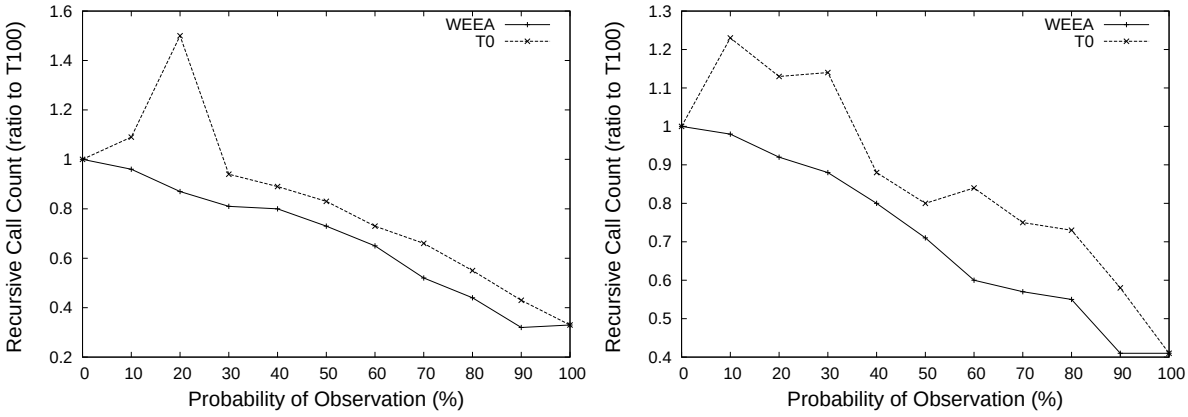


(c) MinW



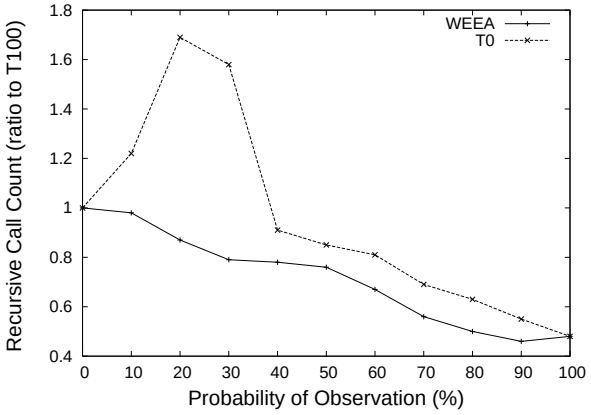
(d) WMinF

Figure 4.1: Effects of increase in probability of observation in VE over the Barley Network.

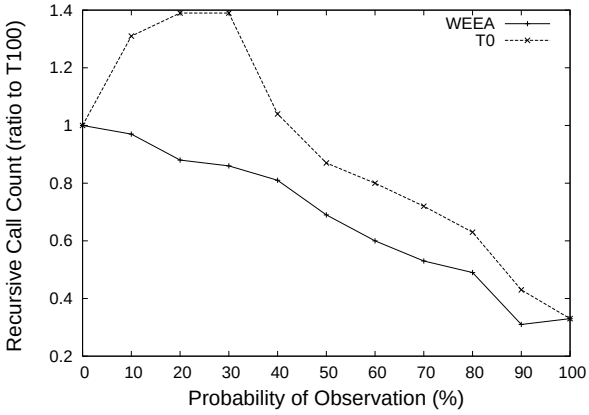


(a) MinF

(b) MinN

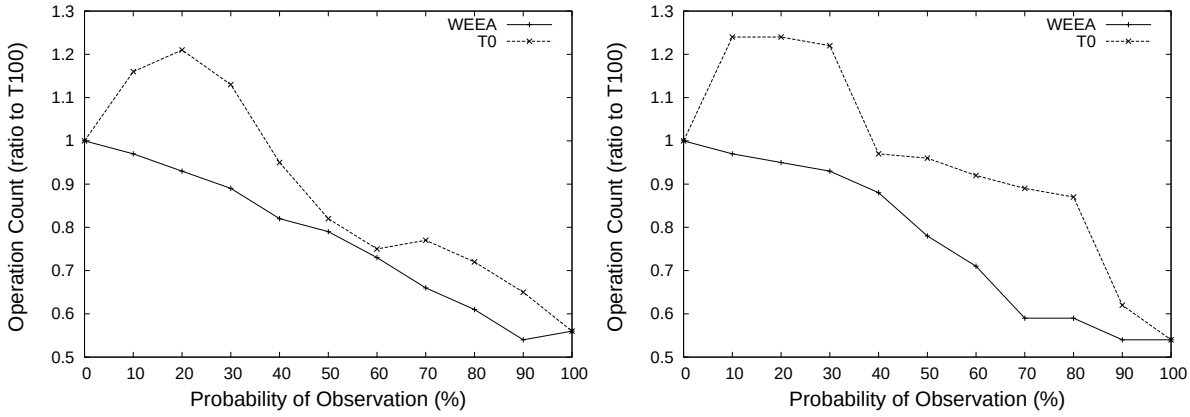


(c) MinW



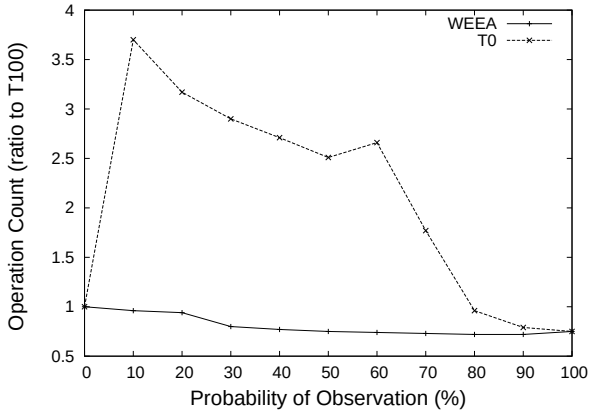
(d) WMinF

Figure 4.2: Effects of increase in probability of observation in ElimTree over the Barley Network.

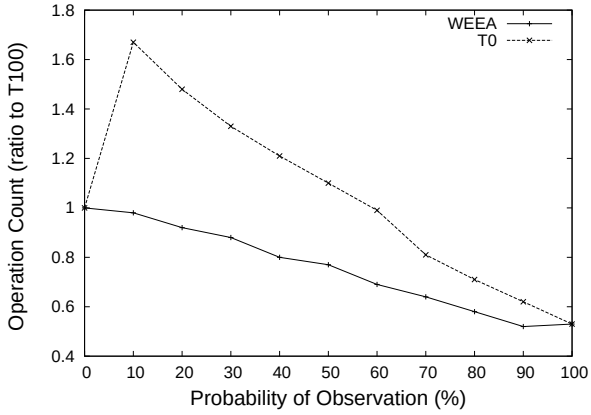


(a) MinF

(b) MinN

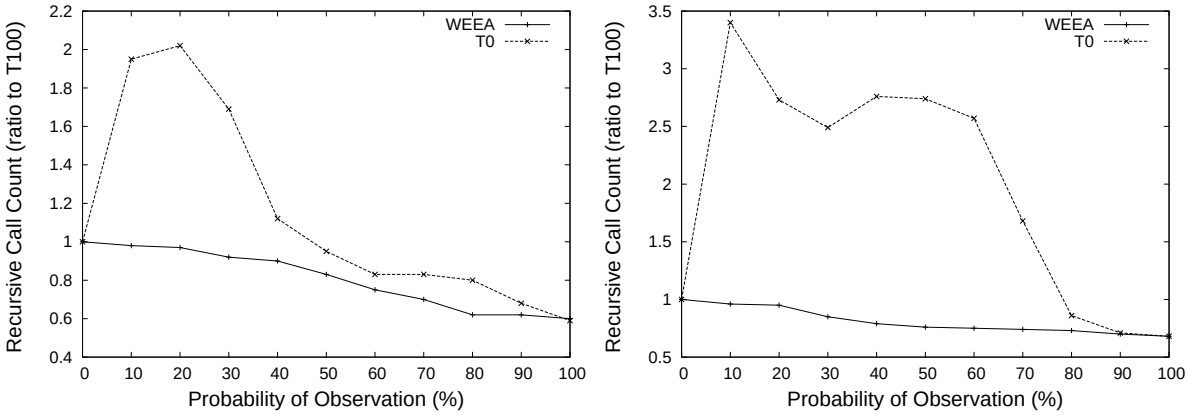


(c) MinW



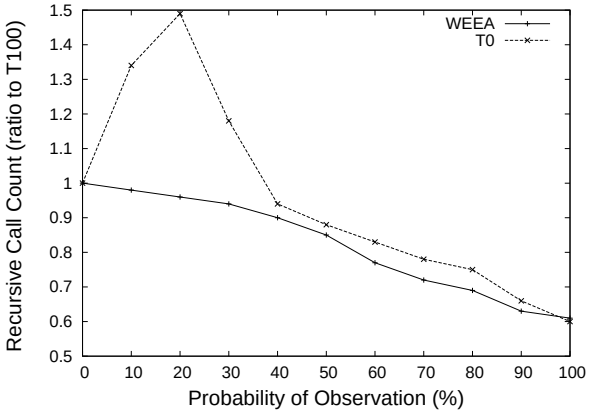
(d) WMinF

Figure 4.3: Effects of increase in probability of observation in VE over the Hailfinder Network.

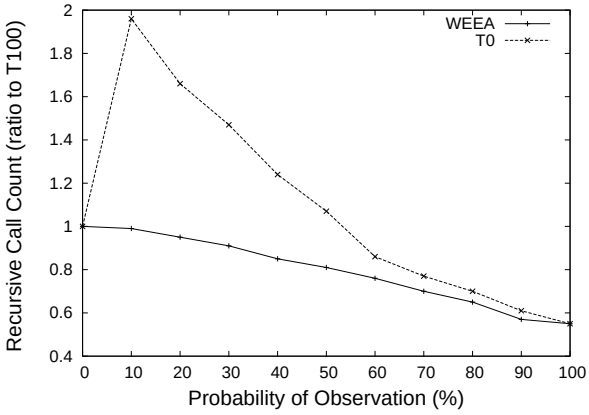


(a) MinF

(b) MinN

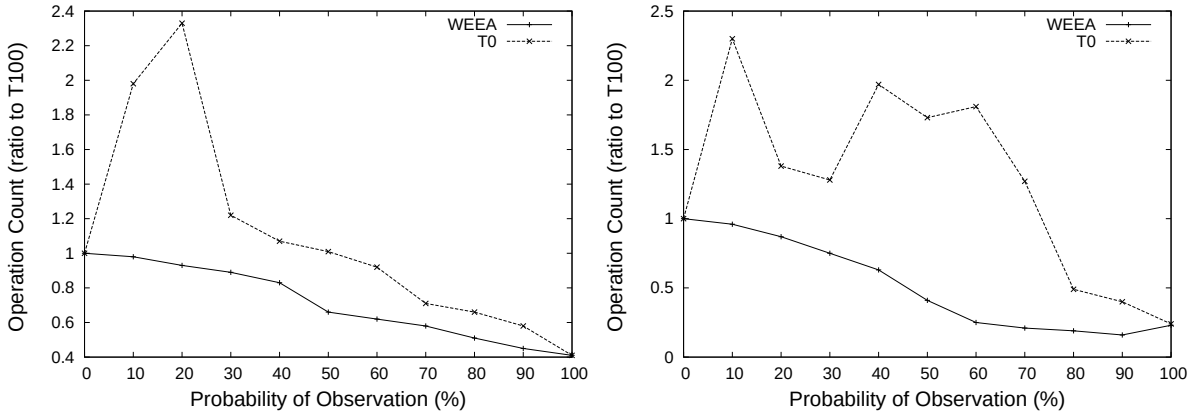


(c) MinW



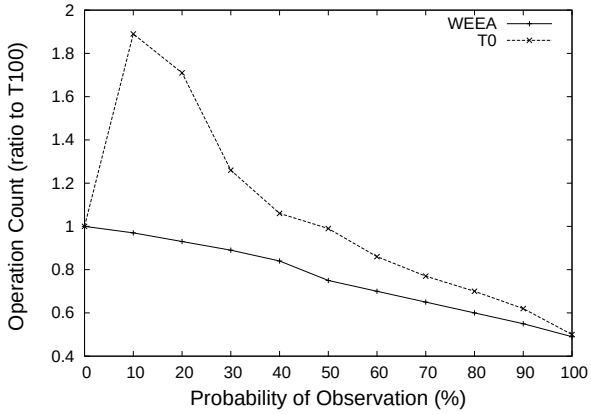
(d) WMinF

Figure 4.4: Effects of increase in probability of observation in ElimTree over the Hailfinder Network.

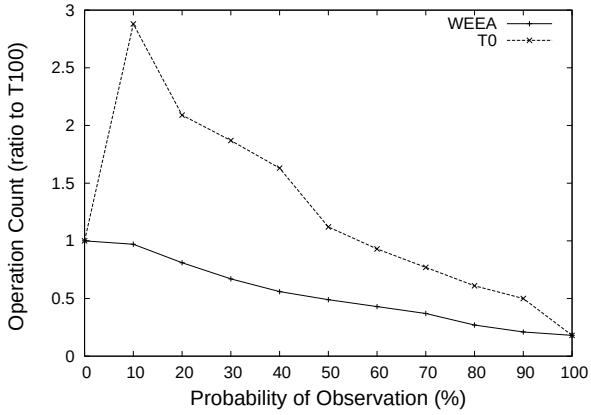


(a) MinF

(b) MinN

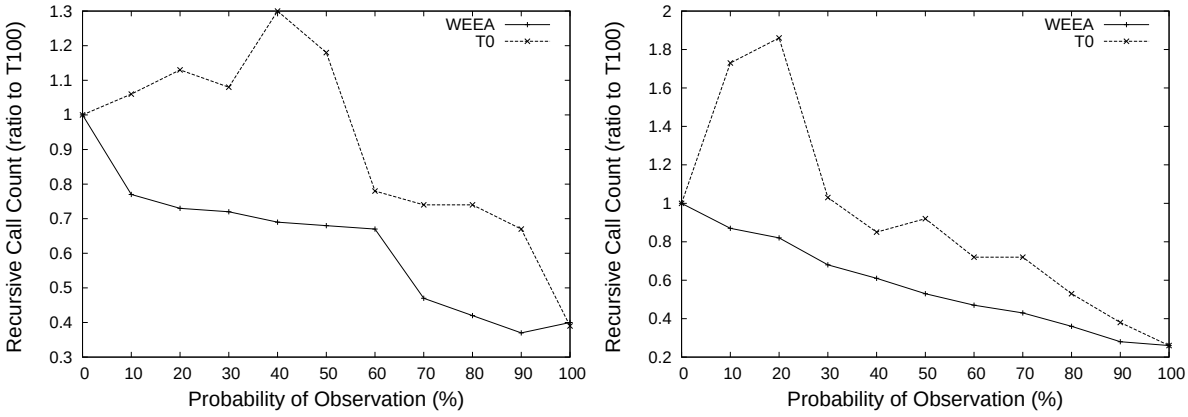


(c) MinW



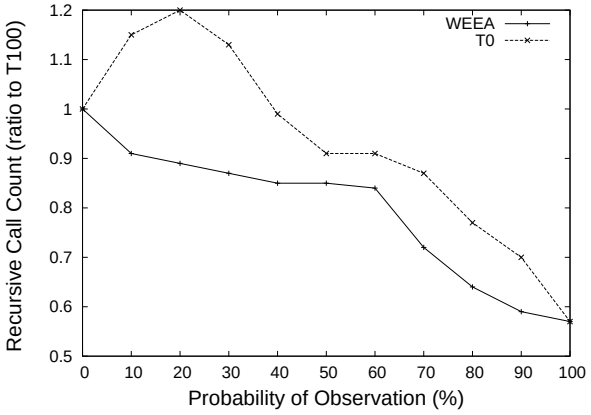
(d) WMinF

Figure 4.5: Effects of increase in probability of observation in VE over the Mildew Network.

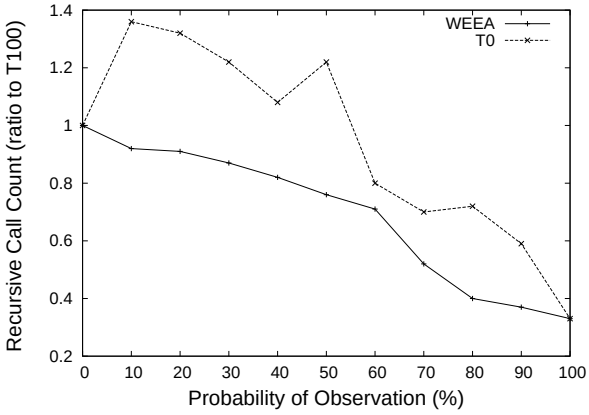


(a) MinF

(b) MinN

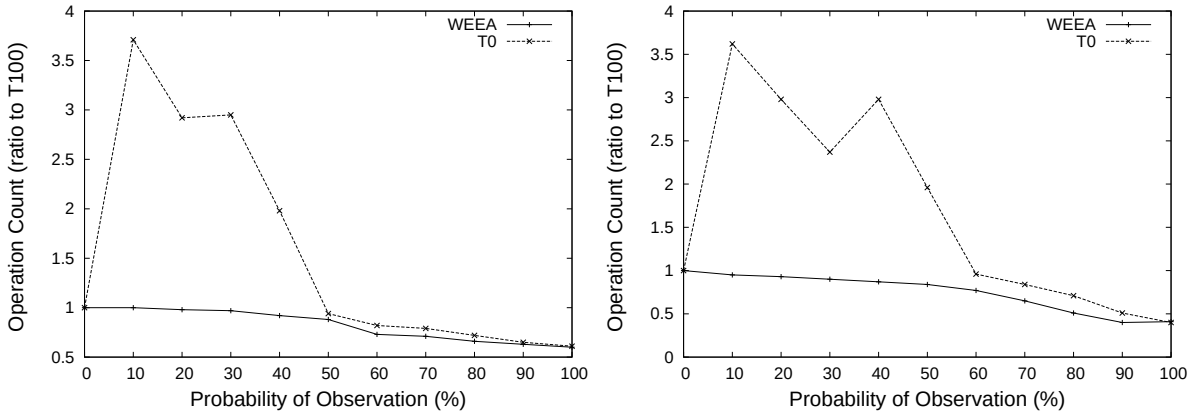


(c) MinW



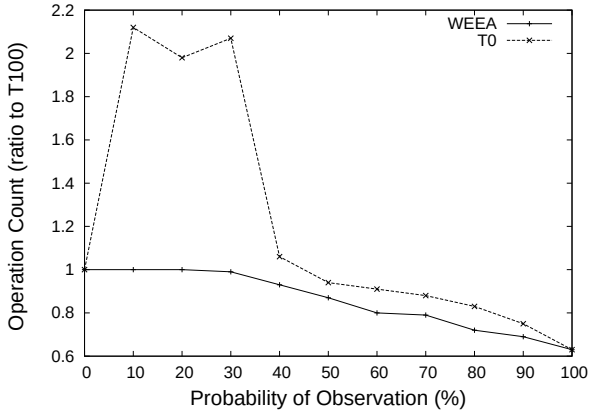
(d) WMinF

Figure 4.6: Effects of increase in probability of observation in ElimTree over the Mildew Network.

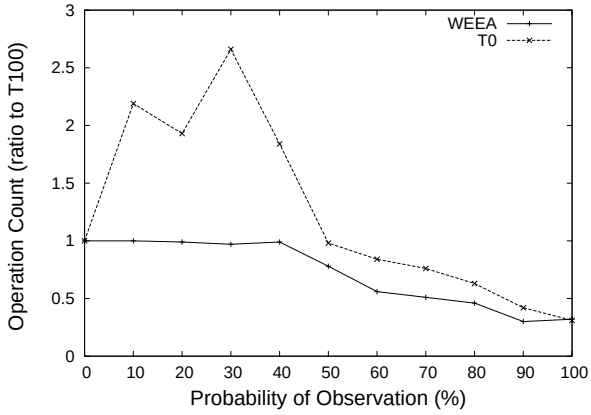


(a) MinF

(b) MinN

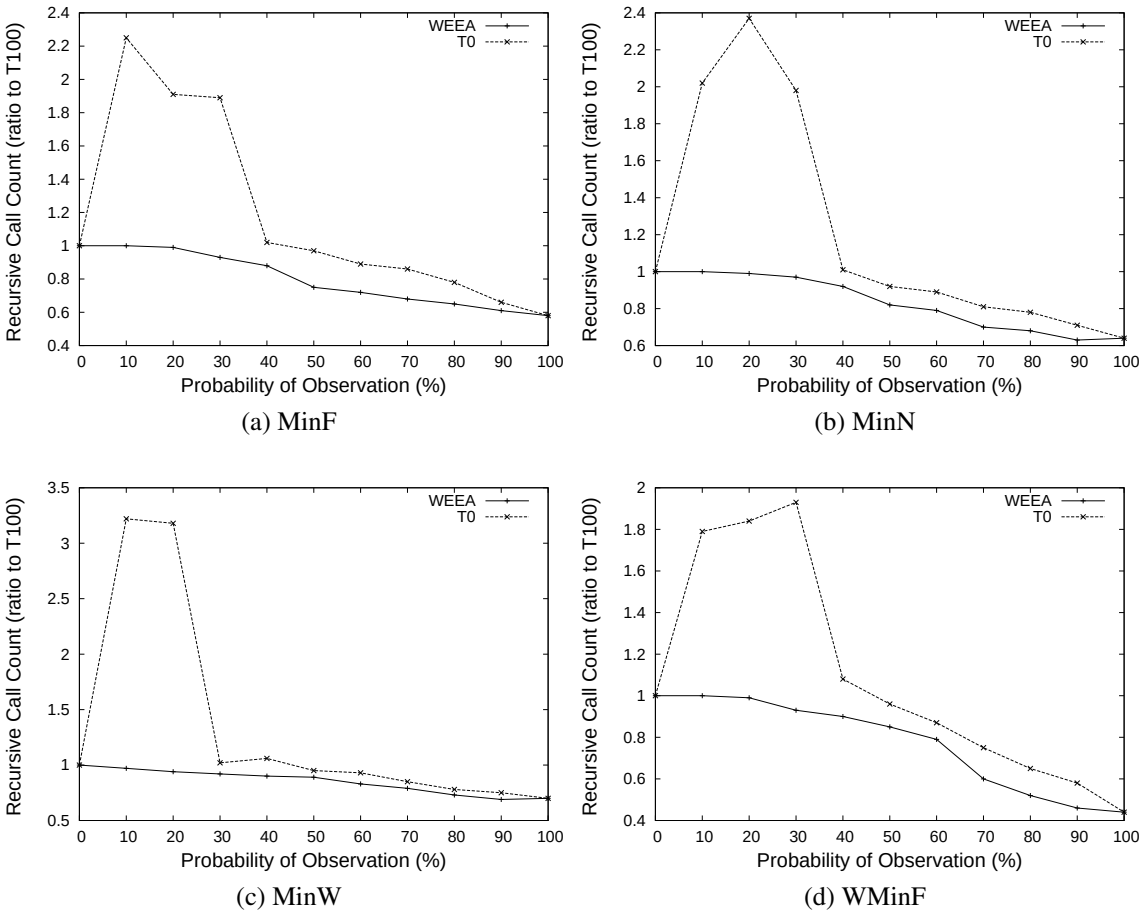


(c) MinW



(d) WMinF

Figure 4.7: Effects of increase in probability of observation in VE over the Munin1 Network.

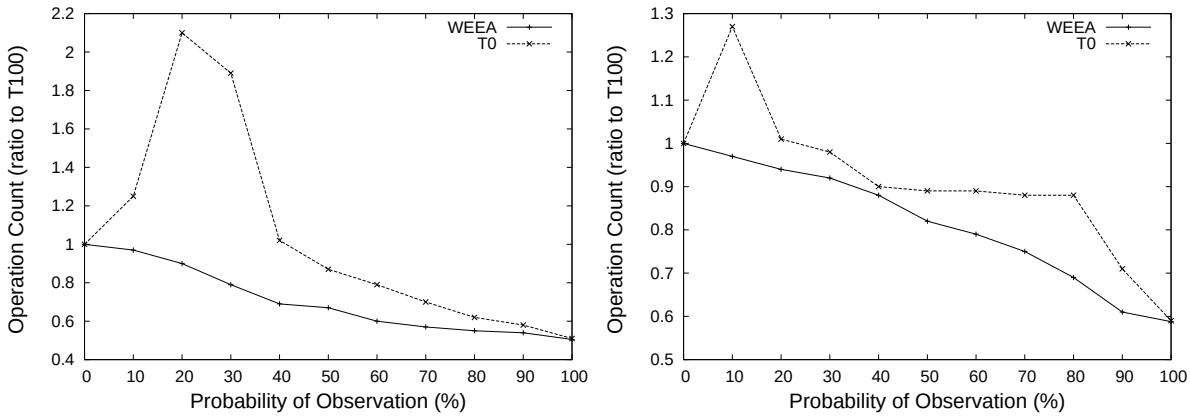


(a) MinF (b) MinN

(c) MinW (d) WMinF

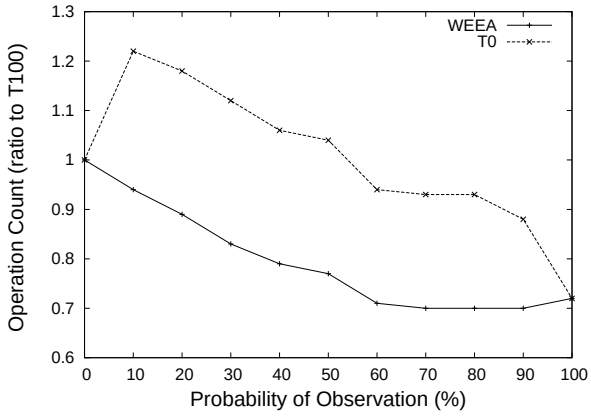
Figure 4.8: Effects of increase in probability of observation in ElimTree over the Munin1 Network.

An interesting feature, however, is that WEEA is at least as good as the *best* thresholding algorithm for any probability, and is typically much better. From these graphs, we observe that as the probabilities of variables increase, there is a gradual decrease over the cost of the arithmetic operations and recursive calls for all tested benchmark networks. Typically, when the probabilities of observations are high, WEEA consistently outperforms the thresholding approaches. On the contrary, while the probabilities are considerably lower, this weighted approach is no longer able to gain much of an advantage. However, it never has any disadvantage, because then it seems more likely that no variables will be observed,

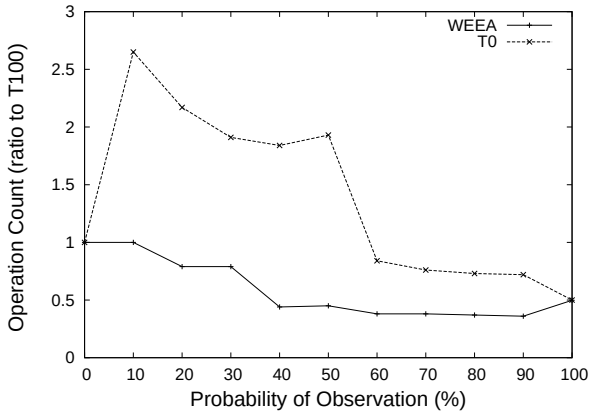


(a) MinF

(b) MinN



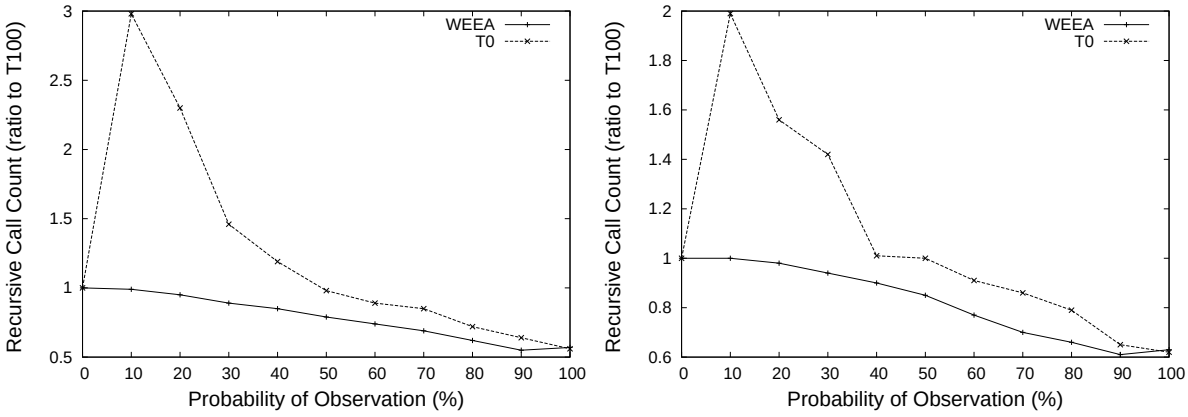
(c) MinW



(d) WMinF

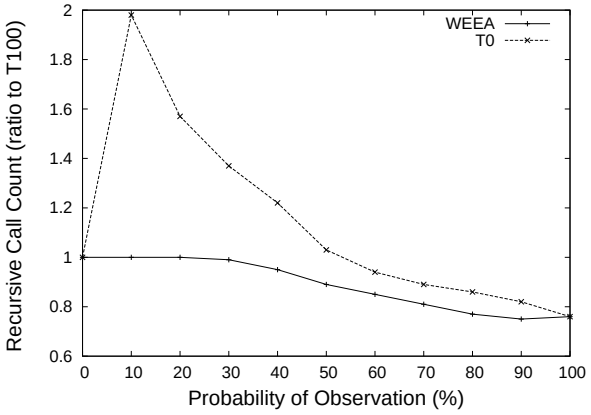
Figure 4.9: Effects of increase in probability of observation in VE over the Munin4 Network.

and this is what the standard algorithms were designed to compute over. Furthermore, low probabilities increase the probabilities on the edges in the triangulated graph. As a result, for relatively low probabilities, the performance of WEEA is almost closer to T100. WEEA starts to enjoy the benefit of inclusion of this application specific knowledge more effectively, as the probability of observation increases steadily and edge weights become more smaller in the triangulated graph. Therefore, we obtain a substantial amount of savings in computational cost of both of the inference methods.

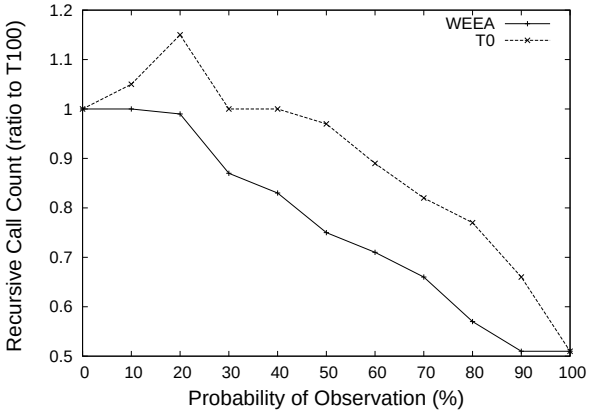


(a) MinF

(b) MinN

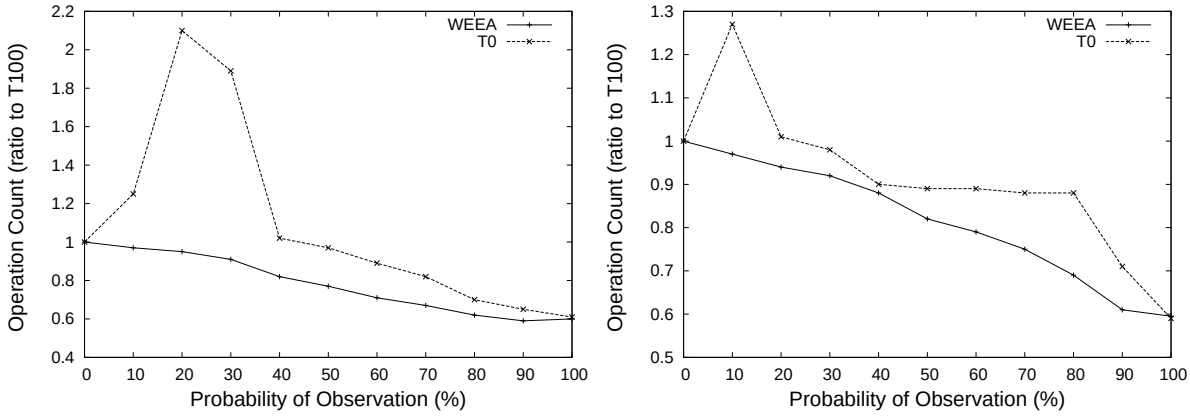


(c) MinW



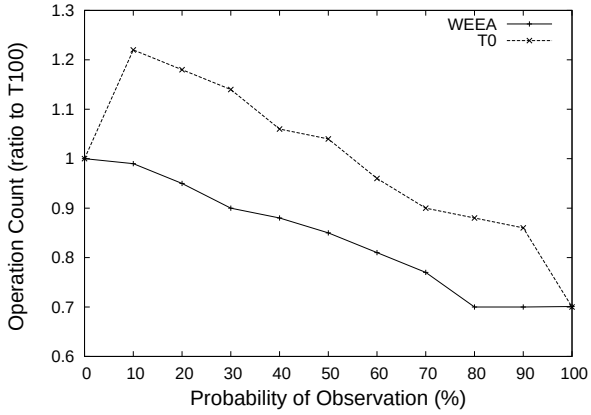
(d) WMinF

Figure 4.10: Effects of increase in probability of observation in ElimTree over the Munin4 Network.

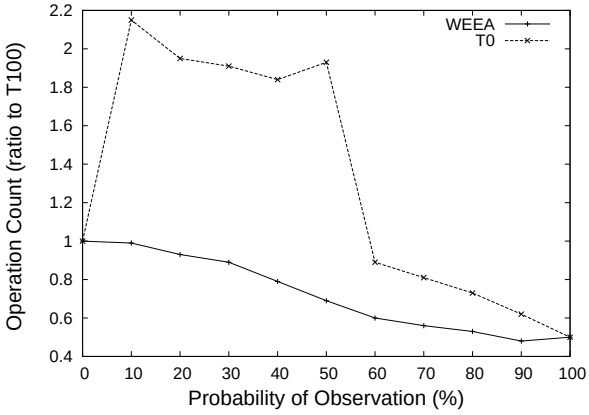


(a) MinF

(b) MinN

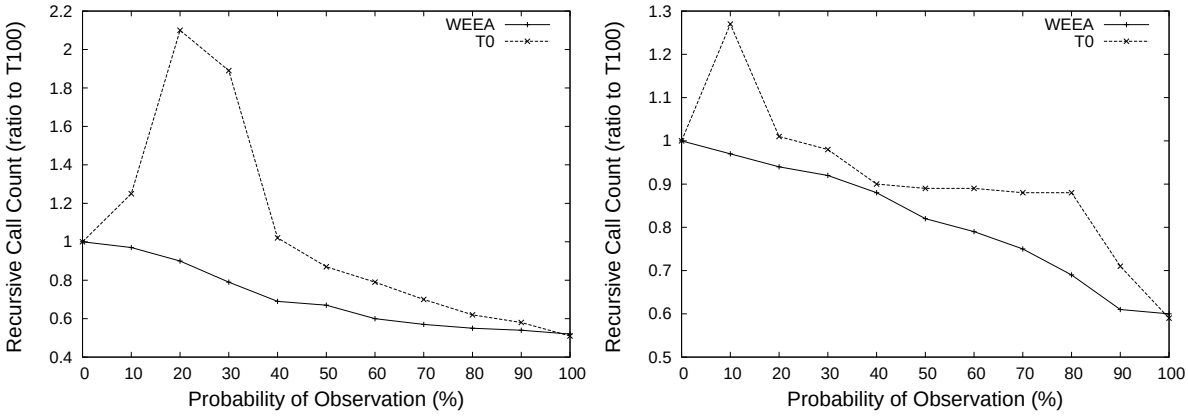


(c) MinW



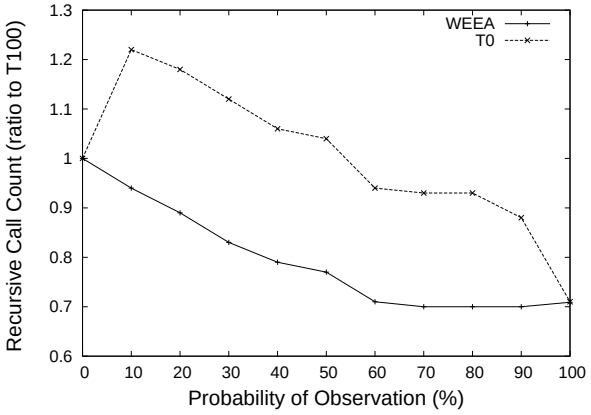
(d) WMinF

Figure 4.11: Effects of increase in probability of observation in VE over the Pigs Network.

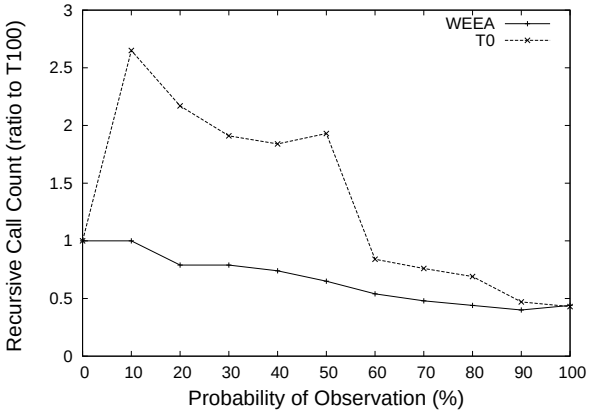


(a) MinF

(b) MinN

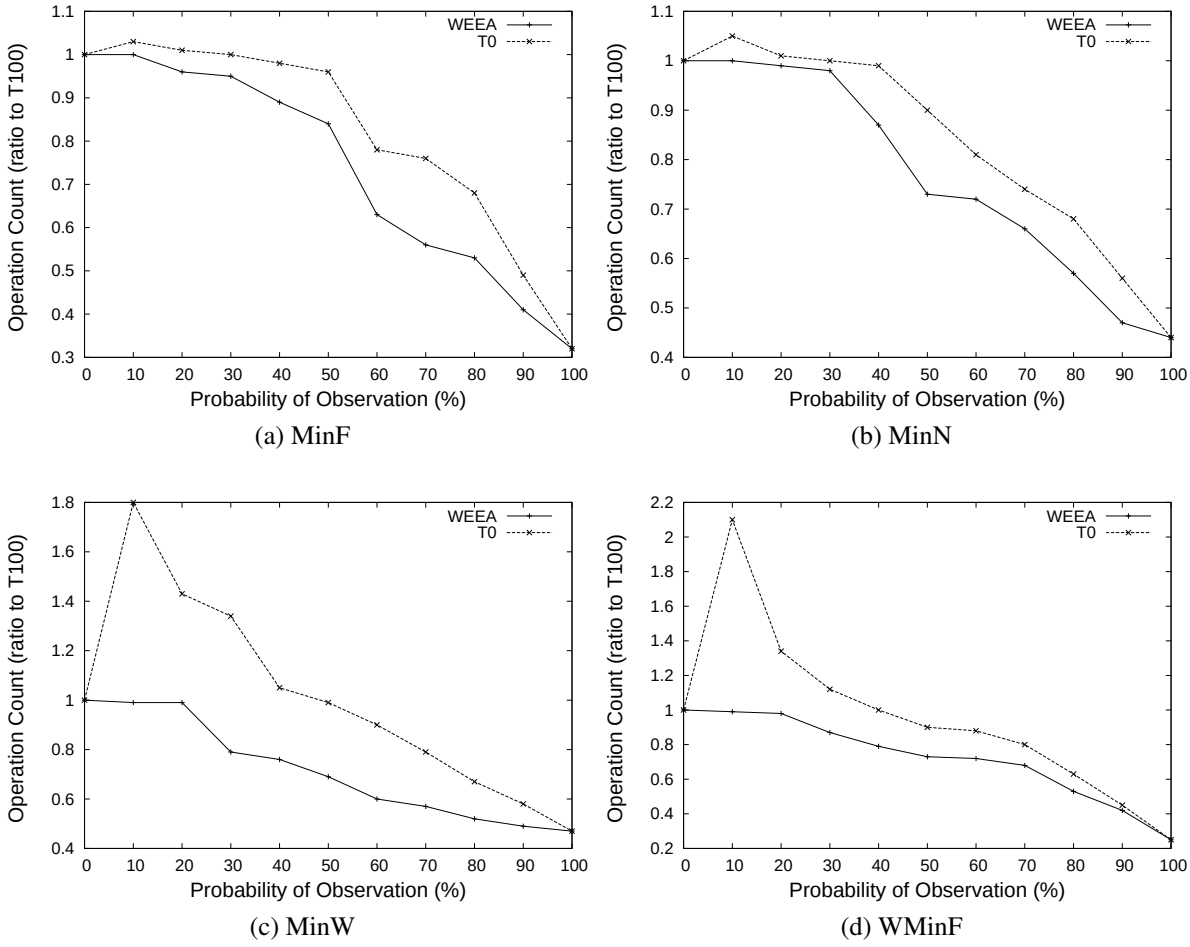


(c) MinW



(d) WMinF

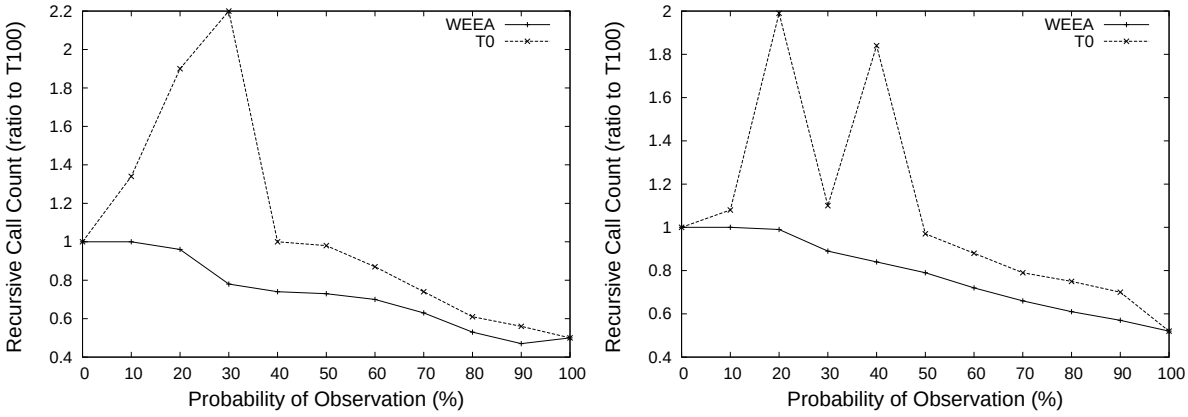
Figure 4.12: Effects of increase in probability of observation in ElimTree over the Pigs Network.



(a) MinF (b) MinN

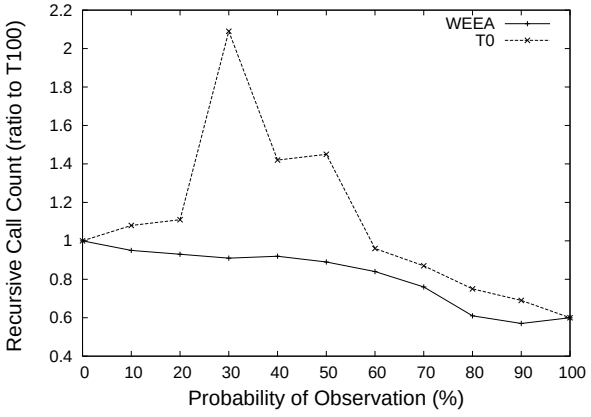
(c) MinW (d) WMinF

Figure 4.13: Effects of increase in probability of observation in VE over the Water Network.

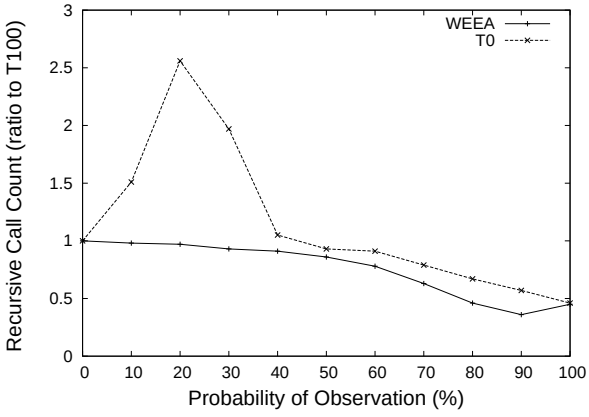


(a) MinF

(b) MinN



(c) MinW



(d) WMinF

Figure 4.14: Effects of increase in probability of observation in ElimTree over the Water Network.

4.2 Summary

In this chapter, we experimentally studied the performance ratio of the elimination orders generated by thresholding and weighted algorithms in comparison to the standard algorithms over the benchmark networks. From the results of the first experiment, we observed that for any benchmark network, the T50 algorithm performs better among three thresholding methods; and, the weighted method consistently outperforms all other algorithms. The second experiment showed how the increase in probability of observation affects the performance of the discussed algorithms. From these experimental results, we observed that T100 performs well with low probabilities, while the performance of T0 is better with high probabilities. We also saw that the WEEA method is superior to others while the magnitude of the probability of observation ranges from moderate to high; even for low probabilities the WEEA method performs at least as well as any other method whose performance is the best among all.

Chapter 5

Conclusion

In this thesis, we consider application-specific information, which can be used to generate better elimination orders for Bayesian network inference if exploited in a proper manner. This application-specific knowledge is the the *probability of observation* over a subset of the variables in the Bayesian network. The probability of observation refers to the likelihood of a variable being observed at runtime during a query.

We present several methods for computing low-cost variable orderings when the probability of observation of a set of variables is known. We consider a simple thresholding technique that generalizes traditional standard algorithms for exploiting the probability of observations. We then demonstrate a more sophisticated approach that uses the observation probabilities to weight the edges of the graph during the triangulation process. We show empirically that these methods reduce the induced width of the variable ordering, as well as improve the average multiplication counts, summation counts, and overall runtime by 50%, 70% and 55%, respectively, when these new orderings are used by the VE algorithm. Our experimental evaluations also illustrate that by employing these new orders, the ElimTree algorithm gains a considerable amount of savings in the runtime and memory usage. These algorithms reduced the recursive calls and runtime by as much as 64% and 50%, respectively. Similarly, the total cache size and the largest cache size are reduced by as much as 55% and 65%, respectively. We point out that these performance enhancements are not restricted to any particular type of inference algorithm, since most of the inference algorithms use elimination orders. Moreover, the process of exploiting the observation probability is a compile time task, and query independent, thus no additional overhead is added at runtime.

There are several avenues of future work. First, we know of no other work that considers exploiting the probability of observation. Choosing variable orderings using this information is only one example, we are considering other areas in Bayesian network inference where such information could be utilized. An example is an *influence diagram*, which is a Bayesian network augmented with decision variables and utility functions specifying the preferences of the decision maker.

In regards to the techniques of this thesis, we made a few assumptions of independence that simplified our procedure. We are considering methods for modeling dependence and thus maintaining exact probabilities on each edge. Since maintaining exact probabilities by complex calculations may cause a substantial amount of overhead at runtime, we used approximations in this thesis. In the future, we are planning to explore methods to implement dependencies, which will not increase the computational cost, but improve the performance of WEEA. In addition, the techniques in this thesis are applicable only to hard evidence; extending them to soft and virtual evidence is left as future work. A final direction that we are considering is the relaxation of the exact nature of the probabilities of observation, as it may be easier in some applications to suggest an approximate probability, or an interval (e.g., a variable is observed with 60-70% probability).

Bibliography

[1] Bayesian network repository. http://www.cs.huji.ac.il/site/labs/compbio/Repository/.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8:277–284, April 1987.

[3] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.

[4] A. Berry, J. R. S. Blair, P. Heggernes, and Barry W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4):287–298, 2004.

[5] H. L. Bodlaender, A. M. C. A. Koster, F. van den Eijkhof, and L. C. van der Gaag. Pre-processing rules for triangulation of probabilistic networks. In *Proceedings of the 17th Conference on Uncertainity in Artifical Intelligence(UAI)*, pages 286–305, 2001.

[6] J. S. Breese and Da. Heckerman. Decision-theoretic case-based reasoning. In *In Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 56–63, 1995.

[7] A. Bretscher, D. Corneil, M. Habib, and C. Paul. A simple linear time LexBFS cograph recognition algorithm. *SIAM J. Discret. Math.*, 22(4):1277–1296, July 2008.

[8] G. F. Cooper. Bayesian belief-network inference using recursive decomposition. Technical Report KSL-90-05, Knowledge Systems, AI Laboratory, USA, 1990.

[9] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

[10] A. Darwiche. Recursive conditioning - any-space conditioning method with treewidth-bounded complexity. *Artificial Intelligence*, 126(1-2):5–41, 2000.

[11] A. Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge University Press, 2009.

[12] A. Darwiche and M. Hopkins. Using recursive decomposition to construct elimination orders, jointrees, and dtrees. In *Trends in Artificial Intelligence, Lecture Notes in AI*, pages 180–191. Springer-Verlag, 2001.

[13] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence*, pages 211–219, 1996.

[14] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

[15] D.Heckerman, J. Breese, and K. Rommelse. Troubleshooting under uncertainty. Technical Report MSR-TR-94-07, 1994.

[16] F. J. Diez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87:1–20, 1996.

[17] P. A. Dow and R. E. Korf. Best-first search for treewidth. In *Proceedings of the 22nd Conference on Artificial Intelligence*. AAAI press,CA, 2007.

[18] P. A. Dow and R. E. Korf. Best-first search with a maximum edge cost function. In *Proceedings of the 10th International Symposium on Artificial Intelligence and Mathematics*. Ft. Lauderdale,FL, 2008.

[19] M. Fishelson, N. Dovgolevsky, and D. Geiger. Maximum likelihood haplotyping for general pedigrees. Technical Report CS-2004-13, Department of Computer Science, Hafia,Israel, 2004.

[20] M. Fishelson and D. Geiger. Optimizing exact genetic linkage computations. In *Proceedings of the International Conference on Research in Computational Molecular Biology(RECOMB)*, pages 114–121, April 2003.

[21] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. In *Proceedings of the fourth annual international conference on Computational molecular biology*, RECOMB '00, pages 127–135, 2000.

[22] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Annual Conference on Uncertainity in Artificial Intelligence*, pages 201–208. AUAI press,Arlinton,VA, 2004.

[23] K. Grant. *Conditioning graphs: Practical structures for inference in Bayesian networks.* PhD thesis, Dept. of Computer Science,University of Saskatchewan, Saskatoon, SK,Canada, 2006.

[24] K. Grant and M. C. Horsch. Conditioning graphs: practical structures for inference in Bayesian networks. In *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, pages 49–59, 2005.

[25] K. Grant and M. C. Horsch. Methods for constructing balanced elimination trees and other recursive decompositions. In *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, pages 812–817, 2006.

[26] K. Grant and K. Scholten. On the structure of elimination trees for Bayesian network inference. In *MICAI (2)*, pages 208–220, 2010.

[27] E. Horvitz, P. Koch, C. M. Kadie, and A. Jacobs. Coordinate: Probabilistic forecasting of presence and availability. In *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence (UAI'02)*, pages 224–233. Morgan Kaufmann Publishers, 2002.

[28] F. V. Jensen. *An introduction to Bayesian networks*. Springer-Verlag, New York,Inc., 1996.

[29] F. V. Jensen, S. L. Lauritzen, and K. G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quaterly*, 4:269–282, 1990.

[30] K.Grant and M. C. Horsch. Efficient caching in elimination trees. In *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*, pages 98–103, 2007.

[31] U. Kjaerulff. Triangulation of graphs—algorithms giving small total state space. Technical Report R90-09, Department of Computer Science, University of Aalborg, Denmark , March 1990.

[32] D. Koller and N. Friedman. *Probabilistic graphical models - Principles and techniques*. MIT Press, 2009.

[33] A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. *Electronic Notes in Discrete Mathematics*, 8:54–75, 2001.

[34] S. Langevin and M. Valtorta. Performance evaluation of algorithms for soft evidential update in Bayesian networks: First results. In *SUM*, pages 284–297, 2008.

[35] S. L. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.

[36] S. Monti and G. F. Cooper. Bounded recursive decomposition: a search-based method for belief-network inference under limited resources. *Int. J. Approx. Reasoning*, 15(1):49–75, 1996.

[37] F. Mousumi and K. Grant. Exploiting the probability of observation for efficient Bayesian network inference. In *Proceedings of the 25th Canadian conference on Advances in Artificial Intelligence*, Canadian AI'12, pages 133–144. Springer-Verlag, 2012.

[38] R. E. Neapolitan. *Probabilistic reasoning in expert systems:Theory and algorithms*. John Wiley and Sons, NY, 1990.

[39] R. E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.

[40] R. Pan, Y. Peng, and Z. Ding. Belief update in Bayesian networks using uncertain evidence. In *18th IEEE International Conference on Tools with Artificial Intelligence*, pages 441–444, 2006.

[41] J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufman, San Mateo, California, 1988.

[42] M. A. Peot and R. D. Shachter. Fusion and propagation with multiple observations in belief networks. *Artif. Intell.*, 48(3):299–318, 1991.

[43] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5(2):266–283, 1976.

[44] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the AAAI/IAAI*, pages 185–190, 1997.

[45] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984.

[46] M. Valtorta, Y. Kim, and J. Vomlel. Soft Evidential Update for Probabilistic Multiagent Systems. *Int. J. Approx. Reasoning*, 29(1):71–106, 2002.

[47] F. van den Eijkhof, H. L. Bodlaender, and A. M. C. A. Koster. Safe reduction rules for weighted treewidth. Technical Report UU-CS-2002-051, Department of Information and Computing Sciences, Utrecht University, 2002.

[48] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.

Appendix-A
Benchmark Bayesian Networks

Network Name	Description
Alarm	Monitoring of emergency care patients
Barley	Model of Barley crops yield.
HailFinder	Predicting Hails in northern Colorado.
Insurance	Evaluating insurance applications
Link	Pedigree for linkage analysis.
Mildew	A model for deciding on the amount of fungicides to be used against attack of mildew in wheat.
Munin	An expert electromyography assistant.
Pigs	Pedigree of breeding pigs.
Water	A model of the biological processes of a water purification plant.

Network Name	Characteristics
Alarm	Number of nodes: 37 Number of edges: 46 Maximum parents / Maximum in-degree: 4 Average degree: 2.49 Maximum values: 4 Number of parameters: 509 Maximum CPT entries: 108 Average Markov blanket size: 3.51 Reference: I. Beinlich and G. Suermondt and R. Chavez and G. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Proceedings of the 2nd European Conference on AI and Medicine*, Springer-Verlag, Berlin, 1989.
Barley	Number of nodes: 48 Number of edges: 84 Maximum parents / Maximum in-degree: 4 Average degree: 3.5 Maximum values: 67 Number of parameters: 114005 Maximum CPT entries: 40320 Average Markov blanket size: 5.25 Reference: Preliminary model for barley developed under the project: Production of beer from Danish malting barley grown without the use of pesticides by K. Kristensen , I. A. Rasmussen, and others.

Network Name	Characteristics
Hailfinder	Number of nodes: 56 Number of edges: 66 Maximum Parents / Maximum in-degree: 4 Average degree: 2.36 Maximum values: 11 Number of parameters: 2656 Maximum CPT entries: 1188 Average Markov blanket size: 3.54 Reference: B. Abramson, J. Brown, W. Edwards, A. Murphy, and R. L. Winkler. Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57-71, 1996.
Insurance	Number of nodes: 27 Number of edges: 52 Maximum parents / Maximum in-degree: 3 Average degree: 3.85 Maximum values: 5 Number of parameters: 984 Maximum CPT entries: 200 Average Markov blanket size: 5.19 Reference: J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. Machine learning, 29(2-3):213-244, 1997.
Link	Number of nodes: 724 Number of edges: 1125 Maximum parents / Maximum in-degree: 3 Average degree: 3.11 Maximum values: 4 Maximum CPT entries: 128 Number of parameters: 14211 Average Markov blanket size: 4.80 Reference: C. S. Jensen and A. Kong. Blocking Gibbs sampling for linkage analysis in large pedigrees with many Loops. The American Journal of Human Genetics, 65(3):885-901, 1999. Also, Research Report R-96-2048, Department of Computer Science, Aalborg University, 1996.
Mildew	Number of nodes: 35 Number of edges: 46 Maximum Parents/ Maximumin-degree: 3 Average degree: 2.63 Maximum values: 100 Number of parameters: 540150 Maximum CPT Entries: 280000 Average Markov blanket size: 4.57 Reference: Bayesian Network Repository. http://www.cs.huji.ac.il/site/labs/compbio/Repository/Datasets/mildew/mildew.htm. The model has been developed by F. V. Jensen, J. Olesen, and U. Kjaerulff.

Network Name	Characteristics
Munin1	Number of nodes: 186 Number of edges: 273 Maximum parents / Maximum in-degree: 3 Average degree: 2.94 Maximum values: 21 Number of parameters: 15622 Maximum CPT entries: 600 Average Markov blanket size: 3.81 Reference: S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjaerulff, M. Woldbye, A. R. Sorensen, A. Rosenfalck, and F. Jensen. MUNIN – an Expert EMG Assistant. In Computer-Aided Electromyography and Expert Systems. Chapter 21, Elsevier (North-Holland), 1989.
Munin2	Number of nodes: 1003 Number of edges: 1244 Maximum parents / Maximum in-degree: 3 Average degree : 2.48 Maximum values: 21 Number of parameters: 69431 Maximum CPT entries: 600 Average Markov blanket size: 3.31 Reference: S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjaerulff, M. Woldbye, A. R. Sorensen, A. Rosenfalck, and F. Jensen. MUNIN – an Expert EMG Assistant. In Computer-Aided Electromyography and Expert Systems. Chapter 21, Elsevier (North-Holland), 1989.
Munin3	Number of Nodes: 1044 Number of edges: 1306 Maximum parents / Maximum in-degree: 3 Average degree: 2.51 Maximum values: 21 Number of parameters: 71059 Maximum CPT entries: 600 Average Markov blanket size: 3.33 Reference: S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjaerulff, M. Woldbye, A. R. Sorensen, A. Rosenfalck, and F. Jensen. MUNIN – an Expert EMG Assistant. In Computer-Aided Electromyography and Expert Systems. Chapter 21, Elsevier (North-Holland), 1989.
Munin4	Number of nodes: 1041 Number of edges: 1397 Maximum parents / Maximum in-degree: 3 Average degree: 2.67 Maximum values: 21 Number of parameters: 80352 Maximum CPT entries: 600 Average Markov blanket size: 3.53 Reference: S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjaerulff, M. Woldbye, A. R. Sorensen, A. Rosenfalck, and F. Jensen. MUNIN – an Expert EMG Assistant. In Computer-Aided Electromyography and Expert Systems. Chapter 21, Elsevier (North-Holland), 1989.

Network Name	Characteristics
Pigs	Number of nodes: 441 Number of edges: 592 Maximum parents / Maximum in-degree: 2 Average degree: 2.68 Maximum values: 3 Number of parameters: 5618 Maximum CPT entries: 27 Average Markov blanket size: 3.66 Reference: Bayesian Network Repository. http://www.cs.huji.ac.il/site/labs/compbio/Repository/Datasets/pigs/pigs.htm.
Water	Number of nodes: 32 Number of edges: 66 Maximum parents / Maximum in-degree: 5 Average degree: 4.12 Maximum values: 4 Number of parameters: 10083 Maximum CPT Entries: 3072 Average Markov blanket size: 7.69 Reference: F. V. Jensen, U. Kjaerulff , K. G. Olesen and J. Pedersen. An Expert System for Control of Waste Water Treatment – A Pilot Project. Technical Report, Judex Data System A/S, Aalborg, Denmark