UNIVERSITY OF CALIFORNIA,
IRVINE

Fixing and Extending the Multiplicative Approximation Scheme

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Information and Computer Science

by

Sidharth Shekhar

Thesis Committee:
Professor Alexander Ihler, Chair
Professor Xiaohui Xie
Professor Deva Ramanan

2009

The thesis of Sidharth Shekhar is approved:

_____

_____

_____

Committee Chair

University of California, Irvine
2009

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I am indebted to all those who have helped me in finishing this thesis. In particular I would like to thank my advisor, Prof. Alexander Ihler for his advice, his guidance, his constructive criticism and his comments on the earlier drafts of my thesis. I would also like to thank my committee members, Prof. Xiaohui Xie and Prof. Deva Ramanan, for taking the time to review and comment on my work.

I would also like to thank my fellow colleague, Drew Frank for some insightful discussions that greatly assisted my understanding of the problem.

I would also like to thank all my friends in our "IKG" group, in particular Shweta Akhila and Vivek Kumar Singh, for ensuring that at least once a week I could take my mind off my work and have some fun.

Last but not least I would like to thank Natasha Flerova for her comments, her support and for always being there to cheer me up whenever I felt any despair.

ABSTRACT OF THE THESIS

Fixing and Extending the

Multiplicative Approximation Scheme

By

Sidharth Shekhar

Master of Science in Information and Computer Science

University of California, Irvine, 2009

Professor Alexander Ihler, Chair

We analyze the Multiplicative Approximation Scheme (MAS) for inference problems, developed by Wexler and Meek in [24]. MAS decomposes the intermediate factors of an inference procedure into factors over smaller sets of variables with a known error and translates the errors into bounds on the results. We analyze the bounds proposed in [24] and show that under certain conditions the bounds are incorrect. We then derive corrected bounds which we show to be theoretically sound. We also derive new bounds for the results of an inference procedure using an alternate error measure based on the L^∞ norm.

We analyze the methods for optimizing the decomposition of a factor. For decompositions containing disjoint sets, Wexler and Meek provide a closed form solution that uses the L^2 norm. We present a method that uses this closed form solution to generate decompositions containing overlapping sets.

Further, we analyze Wexler and Meek's DynaDecomp algorithm, which applies MAS

to bucket elimination [6]. We show that DynaDecomp provides little control over the size of the largest factor generated. We present our own DynaDecompPlus algorithm which uses a different strategy for decomposition. The new strategy allows us to guarantee that we will never generate a factor larger than the specified threshold.

Finally we provide experimental results which show that the L^∞ bounds are tighter than the corrected bounds. We demonstrate the superiority of the DynaDecomp-Plus algorithm by showing how it can perfectly limit the size of the factors whereas DynaDecomp cannot do so.

Chapter 1

Introduction

Graphical models are a widely used representation framework in a variety of fields especially probability theory, statistics and machine learning. Models such as Bayesian networks, Markov networks, constraints networks, etc. are commonly used for reasoning with probabilistic and deterministic information. By using graphs, these models provide an extremely intuitive way to represent the conditional independence structure among random variables.

This thesis focuses on the problem of performing bounded approximate inference in graphical models. In particular, we analyze an approximation scheme called the Multiplicative Approximation Scheme (MAS) developed by Wexler and Meek in [24].

We provide a brief description of the structure of the thesis along with our contributions in the following section.

1.1 Thesis outline

The thesis begins with providing relevant background information in chapter 2. Here we introduce the notation that we will be following throughout the thesis. We then provide a brief description of graphical models and of performing inference in graphical models. We also provide a brief introduction to an exact inference algorithm called bucket elimination [6].

Chapter 3 provides the details of MAS. We present the scheme as originally described by Wexler and Meek. We show that the bounds proposed by them turn out to be incorrect when dealing with marginalization tasks in normalized probability distributions such as those represented by Bayesian networks or Markov Random Fields by providing simple counter examples. We also fix the error bounds by deriving the corrected bounds which we show to be theoretically sound under all conditions.

Chapter 4 presents an alternate metric that can be used to measure the error of a decomposition. The metric we used is based on the L^∞ norm. We show how this metric can be used with MAS and derive new bounds on the accuracy of the results of an inference procedure.

In the preceding chapters, we assumed that the factored decomposition was already given to us. Using this assumption, we measured errors and derived bounds on the results. In chapter 5 we look at the methods for generating the decomposition of a factor. We present the L^2 optimization method developed by Wexler and Meek. It has a closed form solution if the sets in the decomposition are disjoint. We present a method that uses this closed form solution to generate decompositions that consist of overlapping sets.

Chapter 6 looks at the application of MAS to the bucket elimination [6] inference

algorithm. We analyze the DynaDecomp algorithm developed by Wexler and Meek. We present its major limitation, namely that it provides us with little control over the size of the largest factor generated. We modify the algorithm and present our own DynaDecompPlus algorithm. We show how we can easily limit the size of the largest factor that the algorithm generates by using a smarter strategy for decomposition.

Chapter 7 provides the results of the experimental evaluations of the theories presented in the previous chapters. We present scenarios wherein the original MAS bounds fail and wherein the corrected MAS bounds that we derived turn out to be quite loose and impractical. We then compare the corrected MAS bounds and the new L^∞ bounds that we introduced in chapter 4 and show that new bounds are tighter than the corrected bounds. We also compare the performance of the original DynaDecomp algorithm with our DynaDecompPlus algorithm. We demonstrate the superiority of the DynaDecompPlus algorithm by showing how it can perfectly limit the size of the factors whereas DynaDecomp cannot do so. We also look at instances from the UAI'06 inference evaluation competition [1]. These instances clearly highlight the problems with the corrected bounds and the DynaDecomp algorithm. The DynaDecomp algorithm has trouble solving many instances as it ran into memory issues. Therefore, we only present the results of running the DynaDecompPlus algorithm. Comparing the two bounds on these UAI instances, we see that only in some instances are the new bounds tight and useful. The corrected bounds, however, are extremely loose on all the instances we ran.

Chapter 8 concludes the thesis and also looks at some possible directions for future research.

Chapter 2

Background

This chapter provides all of the relevant background information that is required for this thesis. We begin in section 2.1 by introducing the notation we will be following throughout the thesis. We follow this with a brief description of graphical models and inference problems in graphical models in sections 2.2 and 2.3 respectively. Finally we provide a brief introduction to the bucket elimination algorithm in section 2.4.

2.1 Notation

Unless specified otherwise, we follow the conventions specified below

1. We will make no distinction between a variable or a set of variables in this thesis. Uppercase alphabets will be used to denote both variables and sets of variables. For example we can have a variable X_1, or a set of variables $X = \{X_1, X_2, \ldots, X_n\}$.

2. Given a function or probability distribution over a set of variables X, $F(X)$ or

$P(X)$, we denote an approximation to this function as $\tilde{F}(X)$ or $\tilde{P}(X)$.

2.2 Graphical Models

"*Graphical models are a marriage between probability theory and graph theory. They provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering – uncertainty and complexity – and in particular they are playing an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity – a complex system is built by combining simpler parts. Probability theory provides the glue whereby the parts are combined, ensuring that the system as a whole is consistent, and providing ways to interface models to data. The graph theoretic side of graphical models provides both an intuitively appealing interface by which humans can model highly-interacting sets of variables as well as a data structure that lends itself naturally to the design of efficient general-purpose algorithms.*

Many of the classical multivariate probabilistic systems studied in fields such as statistics, systems engineering, information theory, pattern recognition and statistical mechanics are special cases of the general graphical model formalism – examples include mixture models, factor analysis, hidden Markov models, Kalman filters and Ising models. The graphical model framework provides a way to view all of these systems as instances of a common underlying formalism. This view has many advantages – in particular, specialized techniques that have been developed in one field can be transferred between research communities and exploited more widely. Moreover, the graphical model formalism provides a natural framework for the design of new systems."
— Michael Jordan [13].

Probabilistic graphical models are graphs in which the nodes represent random variables, and the connectivity of the graph represents conditional independence assumptions. Two common types of graphical models are Markov networks, which use undirected graphs, and Bayesian Networks, which use directed acyclic graphs. We describe these two models in the following subsections.

2.2.1 Markov Network

A Markov network is a graphical model that encodes a probability distribution using an undirected graph. Formally, we can define a Markov network as follows:

Definition 2.1. *Given an undirected graph $G = (V, E)$ a set of random variables $X = \{X_v; v \in V\}$ form a Markov network with respect to G if the joint density of X can be factored into a product of functions defined as*

$$P(X) = \prod_{C \in cl(G)} \psi_c(X_c)$$

where $cl(G)$ is the set of cliques, or fully connected subgraphs, of G.

Markov networks represent conditional independence in the following manner: two sets of nodes A and B are conditionally independent given a third set C, if all paths between the nodes in A and B are separated by a node in C. For more information on conditional independence in Markov networks see the book, Pearl [20].

One example of a Markov network is the log linear model. The log linear model is given by

$$P(X) = \frac{1}{Z} \exp\left(\sum_k w_k^T \cdot \psi_k(X_k)\right)$$

where Z is called the partition function and is defined as

$$Z = \sum_{X} \exp\left(\sum_{k} w_k^T \cdot \psi_k(X_k)\right)$$

The partition function Z normalizes the distribution.

2.2.2 Bayesian Network

A Bayesian network is a graphical model that encodes a probability distribution using a directed acyclic graph (DAG). Formally, we can define a Bayesian network as follows:

Definition 2.2. *Given a directed acyclic graph $G = (V, E)$ and a set of random variables $X = \{X_v; v \in V\}$, G is a Bayesian network if the joint probability density function can be written as*

$$P(X) = \prod_{v \in V} P(X_v | X_{pa(v)})$$

where $pa(v)$ is the set of parents of v.

Bayesian Networks have a more complicated characterization of conditional independence than Markov networks, since Bayesian Networks take into account the directionality of the arcs. For a detailed description of conditional independence in Bayesian Networks refer to the book, Pearl [20].

An example of a Bayesian network is shown in Figure 2.1. At each node the conditional probability distribution of the child node given the parent node has been specified using a table.

Although Bayesian Networks have a more complicated characterization of conditional

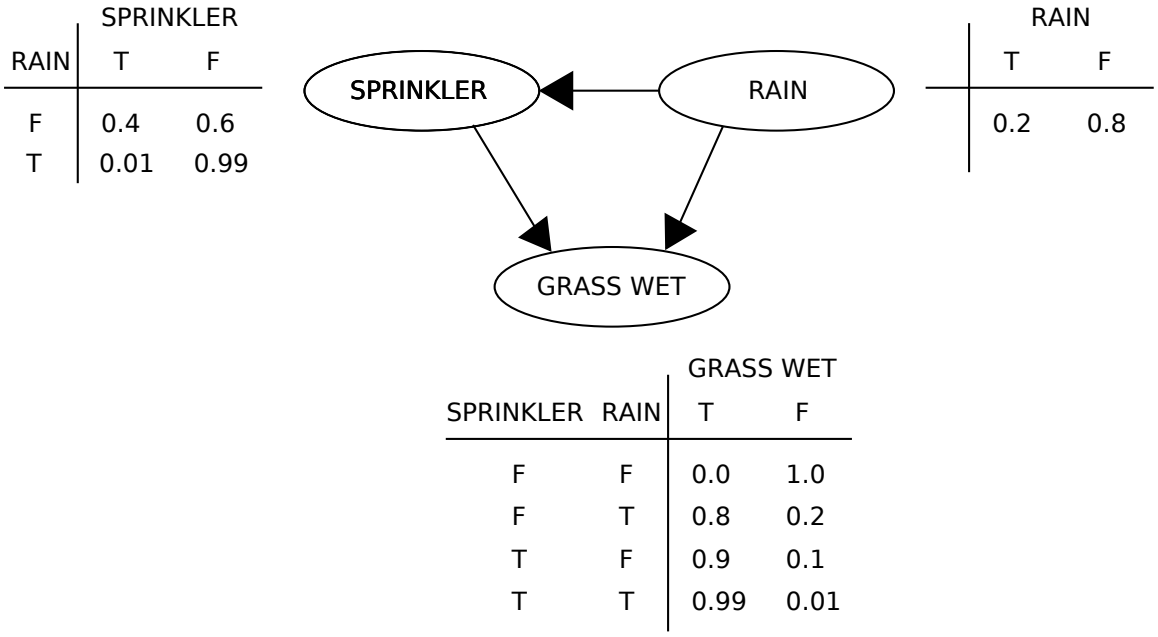

Figure 2.1: A Simple Bayesian Network

independence than Markov networks, they do have some advantages. The biggest advantage is that they can represent causality, i.e., one can regard an arc from A to B as indicating that A "causes" B. For a detailed study of the relationship between directed and undirected graphical models and how to transform one to the other, we direct the reader to the book, Pearl [20].

2.3 Inference in Graphical Models

Given a graphical model on a set of variables X, we can partition this set into disjoint partitions H and E, where H represents hidden variables and E represents our evidence. The evidence consists of the set of variables that have been observed to take on specific values, whereas the hidden variables are the variables for which no observations are available. Hence, if we have no observations then the evidence set $E = \emptyset$, and the set of hidden variables $H = X$. Within this setup, there are four

kinds of queries that are commonly made. They are as follows:

1. Likelihood of Evidence: Given a graphical model and some observations, a question we might have is that how likely are the observations we have under the model we are given. In other words we would like to compute the probability of the evidence we have under the model we have chosen. This is computed as follows:

$$P(E) = \sum_{H} P(H, E)$$

2. Posterior Marginal probability: Given a variable $A \in H$, we would like to evaluate the effect the evidence had on variable A averaging out the effects of all other variables. This is computed as follows:

$$P(A) = \sum_{H \setminus A} P(H, E)$$

3. Most Probable Explanation (MPE): The MPE task is to find an assignment to the set of hidden variables, denoted as H^*, such that the probability $P(H^*)$ is given as follows:

$$P(H^*) = \max_{H} P(H, E)$$

where the assignment H^* is defined as:

$$H^* = \arg\max_{H} P(H, E)$$

4. Maximum aposteriori (MAP) probabilities: Given a set $A \subset H$, the MAP task is to find an assignment to the set of variables in A, denoted as A^*, such that

the probability $P(A^*)$ is given as follows:

$$P(A^*) = \max_A \sum_{H \setminus A} P(H, E)$$

where the assignment A^* is defined as:

$$A^* = \arg\max_A \sum_{H \setminus A} P(H, E)$$

Performing exact inference in a graphical model has been proven to be NP-Hard [4]. A few exact inference methods are available, such as variable elimination [6, 22], clique tree propagation [12, 16, 23], and recursive conditioning [5]. However, all these methods have complexity that is exponential in a graph parameter known as treewidth [21]. In practice the complexities are represented in terms of a related graph parameter called the induced width [8]. We provide more information regarding induced width in section 2.4.

The exponential complexity of performing exact inference has lead to the development of a large number of approximate inference algorithms [15, 25, 14, 9, 10]. With approximation schemes, it is useful to have error bounds on the result, so that we have some guarantees on the exact value of the probabilities we are interested in. Many of the approximation techniques developed do provide such guarantees.

In this thesis we focus on a general scheme called the Multiplicative Approximation Scheme (MAS) [24]. MAS decomposes the intermediate factors of an inference procedure into factors over smaller sets of variables with a known error. It then translates the errors into bounds on the accuracy of the results. To test the accuracy and tightness of the MAS bounds, we apply MAS to the bucket elimination algorithm. In the next section we provide a brief introduction to the bucket elimination algorithm.

2.4 Introduction to Bucket Elimination

Bucket elimination is a framework that provides a unifying view of variable elimination algorithms for a variety of reasoning tasks.

A bucket elimination algorithm accepts as an input an ordered set of variables and a set of dependencies, such as the probability functions given by a graphical model. Each variable is then associated with a bucket constructed as follows: all the functions defined on variable X_i but not on higher index variables are placed into the bucket of X_i. Once the buckets are created the algorithm processes them from last to first. It computes new functions, applying an *elimination operator* to all the functions in the bucket. The new functions summarizing the effect of X_i on the rest of the problem, are placed in the appropriate lower buckets.

The *elimination operator* depends on the task. For example, if we consider the task of computing likelihood of evidence or computing the posterior marginal then the desired *elimination operator* is summation. To eliminate a variable X_i we only need to look at the functions that depend on X_i. Therefore, we look at the bucket of X_i and compute $\sum_{X_i} F$, where F is the product of all functions in the bucket of X_i. Similarly for an MPE task, the *elimination operator* will be maximization. The bucket-elimination algorithm terminates when all buckets are processed, or when some stopping criterion is satisfied.

To demonstrate the bucket elimination approach, we consider an example for computing the posterior marginal. We apply the bucket elimination algorithm to the network in Figure 2.2a, with the query variable A, the ordering $o = (A, E, D, C, B)$, and evidence $E = 0$. The posterior marginal $P(A|E = 0)$ is given by

$$P(A|E = 0) = \frac{P(A, E = 0)}{P(E = 0)}$$

We can compute the joint probability $P(A, E = 0)$ as

$$\begin{aligned} P(A, E = 0) &= \sum_{E=0,D,C,B} P(A, B, D, C, E) \\ &= \sum_{E=0,D,C,B} P(A)P(C|A)P(E|B, C)P(D|A, B)P(B|A) \\ &= \sum_{E=0} \sum_{D} \sum_{C} P(C|A) \sum_{B} P(E|B, C)P(D|A, B)P(B|A) \end{aligned}$$

The bucket elimination algorithm computes this sum from right to left using the buckets, as shown below:

1. bucket B: $h^B(A, D, C, E) = \sum_B P(E|B, C)P(D|A, B)P(B|A)$
2. bucket C: $h^C(A, D, E) = \sum_C P(C|A)h^B(A, D, C, E)$
3. bucket D: $h^D(A, E) = \sum_D h^C(A, D, E)$
4. bucket E: $h^E(A) = h^D(A, E = 0)$
5. bucket A: $P(A, E = 0) = P(A)h^E(A)$,

To compute the posterior marginal $P(A|E = 0)$, we can just normalize the result in bucket A since summing out A is equivalent to calculating $P(E = 0)$. Therefore we can calculate $P(A|E = 0)$ as

$$P(A|E = 0) = \alpha P(A)h^E(A)$$

where α is a normalizing constant. A schematic trace of the algorithm is shown in Figure 2.3.

The performance of bucket elimination algorithms can be predicted using a graph parameter called *induced width* [8]. It is defined as follows

Definition 2.3. *An ordered graph is a pair (G, o) where G is an undirected graph, and $o = (X_1, \ldots, X_n)$ is an ordering of nodes. The width of a node is the number of the node's neighbors that precede it in the ordering. The width of an ordering o is the maximum width over all nodes. The induced width of an ordered graph, denoted by $w^*(o)$, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X_i is processed all its preceding neighbors are connected. The induced width of a graph, denoted by w^*, is the minimal induced width over all its orderings.*

For example, to compute the induced width along any ordering of the graph in Figure 2.2a, we need to to first convert it into an undirected graph. This can be easily done as follows: First we traverse the graph and for every two unconnected nodes that have a common child, we connect them using an undirected edge. In Figure 2.2a this is shown using the undirected dashed edge. Next we convert all the directed edges into undirected edges. This undirected graph is known as the moral graph. Figures 2.2b and 2.2c depict the induced graphs of the moral graph in Figure 2.2a along the orderings $o = (A, E, D, C, B)$ and $o' = (A, B, C, D, E)$, respectively. The dashed line here represents the induced edges, which are the edges that were introduced in the induced ordered graph but were absent in the moral graph. We can see that $w^*(o) = 4$ and $w^*(o') = 2$.

The complexity of bucket elimination algorithms is time and space exponential in $w^*(o)$ [6]. Therefore, ideally we want to find the variable ordering that has smallest induced width. Although this problem has been shown to be NP-Hard [2], there are a few greedy heuristic algorithms that provide good orderings [7].

The greedy heuristic that we use in our experiments is known as the MIN-FILL algorithm. The reason we choose MIN-FILL is because in most cases it has been

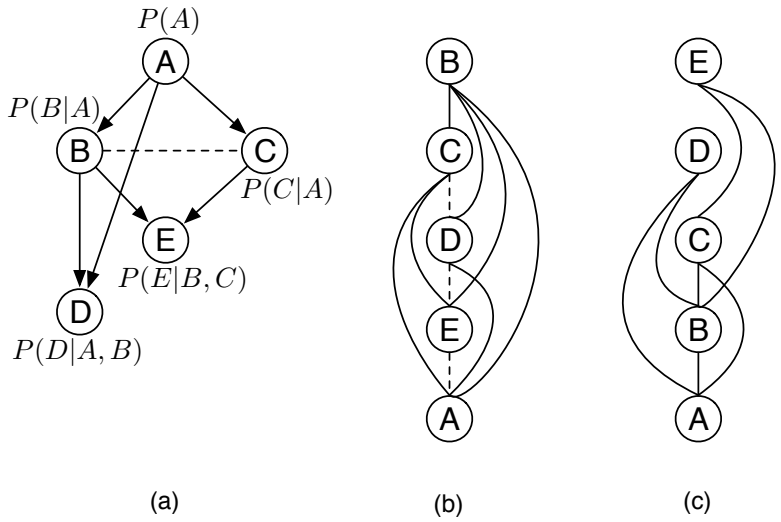

Figure 2.2: (a) A belief network, (b) its induced graph along $o = (A, E, D, C, B)$, and (c) its induced graph along $o = (A, B, C, D, E)$

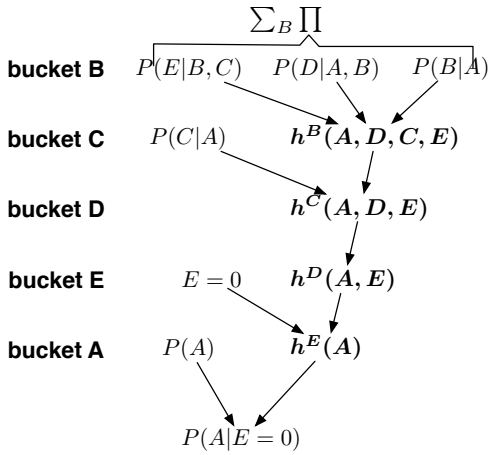

Figure 2.3: A trace of algorithm *elim-bel*

Input: A graph $G = (V, E)$, $V = V_1, \ldots, V_n$
Output: An ordering o of the nodes in V
for $j = n$ *to* 1 *by* -1 **do**
    $R \leftarrow$ a node in V with smallest fill edges for his parents
    Put R in position j: $o[j] \leftarrow R$
    Connect R's neighbors: $E \leftarrow E \cup (V_i, V_j)$, if $(V_i, R) \in E$ and $(V_j, R) \in E$
    Remove R from resulting graph: $V \leftarrow V - \{R\}$
end
Return the ordering o

Figure 2.4: The MIN-FILL algorithm

empirically demonstrated to be better than other greedy algorithms [7]. The MIN-FILL algorithm orders nodes in the order of their *min-fill* value, which is the number of edges that need to be added to the graph so that its parent set is fully connected. The MIN-FILL algorithm is provided in Figure 2.4.

Chapter 3

The Multiplicative Approximation Scheme

Wexler and Meek recently introduced a new approximation scheme called the Multiplicative Approximation Scheme (MAS) [24]. The primary goal of this scheme was to provide good error bounds for computation of likelihood of evidence, marginal probabilities and the maximum a posteriori probabilities in discrete directed and undirected graphical models. In section 3.1 we describe the probability distribution we will be using in our analysis. In section 3.2 we provide the basics of MAS and the methodology by which the MAS error bounds are calculated.

3.1 Preliminaries

From this point in the thesis, unless specified otherwise, the graphical model we will consider will be defined on a set of n variables $X = \{X_1, \ldots, X_n\}$, and will encode

the normalized probability distribution

$$P(X) = \frac{1}{Z} \prod_j \psi_j(D_j)$$

where $D_j \subseteq X$ and $0 < \psi_j(D_j) < 1$.

For convenience we will be using the notation

$$\phi(D_j) = \log \psi_j(D_j)$$

Finally, we will be denoting the set of evidence variables as E and the set of hidden variables as H. Within this setup, the log joint probability can be represented by a sum of factors as follows:

$$\log P(X) = \log P(H, E) = \sum_j \phi_j(D_j)$$

3.2 MAS Basics

The fundamental operations of MAS are local approximations that are known as ϵ-decompositions. Each decomposition has an error associated with it. The errors of all such local decompositions are later combined into an error bound on the result. Let us first define an ϵ-decomposition.

Definition 3.1. *Given a set of variables X, and a function $\phi(X)$ that assigns real values to every instantiation of X, a set of m functions $\tilde{\phi}_l(X_l)$, $l = 1 \ldots m$, where $X_l \subseteq X$, is an ϵ-decomposition if $\bigcup_l X_l = X$, and for all instantiations of X the*

following holds:

$$\frac{1}{1+\epsilon} \leq \frac{\sum_l \tilde{\phi}_l(X_l)}{\phi(X)} \leq 1+\epsilon \tag{3.1}$$

for some $\epsilon \geq 0$.

With this definition in mind, Wexler and Meek make the following claims

Claim 3.1. *The log of the joint probability* $P(H,E)$ *can be approximated within a multiplicative factor of* $1+\epsilon_{max}$ *using a set of* ϵ_j*-decompositions, where* $\epsilon_{max} = \max_j \epsilon_j$.

$$\frac{1}{1+\epsilon_{max}} \log P(H,E) \leq \log \tilde{P}(H,E) \leq (1+\epsilon_{max}) \log P(H,E) \tag{3.2}$$

where $\log \tilde{P}(H,E)$ *is defined as*

$$\log \tilde{P}(H,E) = \sum_{j,l} \tilde{\phi_{jl}}(D_{jl})$$

Claim 3.2. *For a set* $A \subseteq H$*, the expression* $\log \sum_A P(H,E)$ *can be approximated within a multiplicative factor of* $1+\epsilon_{max}$ *using a set of* ϵ_j*-decompositions.*

$$\frac{1}{1+\epsilon_{max}} \log \sum_A P(H,E) \leq \log \sum_A \tilde{P}(H,E) \leq (1+\epsilon_{max}) \log \sum_A P(H,E) \tag{3.3}$$

To prove their claims, Wexler and Meek make the assumption that all functions satisfy the property $\psi_j(D_j) > 1,\ \forall D_j$. However, as this is not always the case, they state that in such scenarios we can first scale or shift in the log domain, every function that does not satisfy the property, i.e., we can generate a function $\psi'_j(D_j) = z_j \cdot \psi_j(D_j)$ such that the property $\psi'_j(D_j) > 1,\ \forall D_j$ is true. Then at the end of the procedure we renormalize by dividing the result by $\prod_j z_j$. Wexler and Meek make the claim that

doing the shift and renormalize procedure does not affect the results of claim 3.1 and claim 3.2. In the next section, we derive the MAS error bounds and verify whether performing the shift and renormalize procedure has no effect on the error bounds.

3.3 MAS Error Bounds

We divide this section into two sub-sections. In the first part, we perform the shift and renormalize procedure and derive the MAS error bounds. In the second part, we make the assumption that all the factors satisfy the property $0 < \psi_j(D_j) < 1$. With this assumption we derive MAS error bounds without performing any shifting.

3.3.1 Shift and Renormalize

Given our normalized probability distribution $P(X)$, we create new functions ψ'_j by multiplying each function ψ_j with a constant z_j such that

$$z_j > \min_{D_j} \psi_j(D_j)$$

Thus, we define

$$\psi'_j(D_j) = z_j \cdot \psi_j(D_j)$$

which has the property that $\psi'_j(D_j) > 1,\ \forall D_j$.

We now define the function $f(X)$ to be the product of the $\psi'_j s$

$$f(X) = \prod_j \psi'_j(D_j) = \prod_j z_j \cdot \psi_j(D_j)$$

and also define the shifted functions $\phi'_j(D_j)$ as

$$\phi'_j(D_j) = \log \psi'_j(D_j) = (\log z_j + \phi_j(D_j))$$

Assume that for each factor ψ'_j we have an ϵ_j-decomposition. With this assumption we proceed to derive bounds for the log of the joint probability $\log P(H, E)$. We first consider the approximation to $\log f(H, E)$ given by

$$\log \tilde{f}(H, E) \equiv \log \prod_{j,l} e^{\tilde{\phi_{jl}}'(D_{jl})} = \sum_{j,l} \tilde{\phi_{jl}}'(D_{jl})$$

Using the definition of an ϵ-decomposition (3.1) and the fact that $\{\psi'_{jl}\}$ is an ϵ-decomposition for ψ'_j, we have

$$\begin{aligned} \log \tilde{f}(H, E) = \sum_{j,l} \tilde{\phi_{jl}}'(D_{jl}) &\leq \sum_j (1 + \epsilon_j)\phi'_j(D_j) \\ &\leq (1 + \epsilon_{max}) \sum_j \phi'_j(D_j) \end{aligned}$$

To perform the renormalization we divide $\tilde{f}(H, E)$ by $\prod_j z_j$, which is equivalent to subtracting $\sum_j \log z_j$ from $\log \tilde{f}(H, E)$. Hence, we get

$$\log \tilde{f}(H, E) - \sum_j \log z_j \leq (1 + \epsilon_{max}) \sum_j \phi'_j(D_j) - \sum_j \log z_j \tag{3.4}$$

We have,

$$P(X) = \prod_j \psi_j(D_j) = \frac{f(X)}{\prod_j z_j}$$

So, we define $\tilde{P}(X)$ as

$$\tilde{P}(X) = \frac{\tilde{f}(X)}{\prod_j z_j}$$

Using the definitions of $\tilde{P}(X)$ and $\phi'_j(D_j)$ in equation (3.4), we get

$$\begin{aligned}
\log \tilde{P}(H,E) &\leq (1+\epsilon_{max}) \sum_j (\phi_j(D_j) + \log z_j) - \sum_j \log z_j \\
&= (1+\epsilon_{max}) \sum_j \phi_j(D_j) + (1+\epsilon_{max}) \sum_j \log z_j - \sum_j \log z_j \\
&= (1+\epsilon_{max}) \log P(H,E) + \epsilon_{max} \sum_j \log z_j \qquad (3.5)
\end{aligned}$$

Equation (3.5) provides us with an upper bound on the log of the joint probability. Next we will derive a similar lower bound on the log of the joint probability.

Again, using the definition of an ϵ-decomposition (3.1), we have

$$\begin{aligned}
\log \tilde{f}(H,E) = \sum_{j,l} \tilde{\phi_{jl}}'(D_{jl}) &\geq \sum_j \frac{1}{1+\epsilon_j} \phi'_j(D_j) \\
&\geq \frac{1}{1+\epsilon_{max}} \sum_j \phi'_j(D_j)
\end{aligned}$$

Subtracting $\sum_j \log z_j$ from both sides and using the definition of $\phi'_j(D_j)$, we have

$$\begin{aligned}
\log \tilde{P}(H,E) &\geq \frac{1}{1+\epsilon_{max}} \sum_j (\phi_j(D_j) + \log z_j) - \sum_j \log z_j \\
&= \frac{1}{1+\epsilon_{max}} \sum_j \phi_j(D_j) + \frac{1}{1+\epsilon_{max}} \sum_j \log z_j - \sum_j \log z_j \\
&= \frac{1}{1+\epsilon_{max}} \log P(H,E) - \frac{\epsilon_{max}}{1+\epsilon_{max}} \sum_j \log z_j \qquad (3.6)
\end{aligned}$$

Equation (3.6) provides us with a lower bound on the log of the joint probability. Combined with the upper bound on the log of the joint probability (3.5), we can state the following theorem:

Theorem 3.1. *If the log of the joint probability $P(H,E)$ is approximated using a set*

of ϵ_j-decompositions on the factors ψ_j of P, then we have the following bounds:

$$\log \tilde{P}(H,E) \leq (1+\epsilon_{max}) \log P(H,E) + \epsilon_{max} \sum_j \log z_j$$

$$\log \tilde{P}(H,E) \geq \frac{1}{1+\epsilon_{max}} \log P(H,E) - \frac{\epsilon_{max}}{1+\epsilon_{max}} \sum_j \log z_j$$

where $\epsilon_{max} = \max_j \epsilon_j$.

Comparing theorem 3.1 with claim 3.1, we see that the bounds differ in that the bounds in theorem 3.1 have extra terms that are multiples of $\sum_j \log z_j$.

Having derived bounds for the joint probability, we proceed towards deriving bounds for marginal probabilities, i.e., we would like bounds for the expression $\log \sum_A P(H,E)$ where $A \subseteq H$.

From theorem 3.1, we have

$$\log \tilde{P}(H,E) \leq \log \Big[P(H,E) \Big]^{(1+\epsilon_{max})} + \log \Big(\prod_j z_j \Big)^{\epsilon_{max}}$$

We would like to get a relation between $\sum_A P(H,E)$ and $\sum_A \tilde{P}(H,E)$. Therefore, we rearrange the terms on the right hand side of the above equation as follows

$$\begin{aligned} \log \tilde{P}(H,E) &\leq \log \Big[(P(H,E))^{(1+\epsilon_{max})} \cdot \Big(\prod_j z_j \Big)^{\epsilon_{max}} \Big] \\ &= \log \Big[P(H,E) \cdot \Big(\prod_j z_j \Big)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}} \Big]^{1+\epsilon_{max}} \end{aligned}$$

Marginalizing out the set A from $\tilde{P}(H,E)$, we get

$$\log \sum_A \tilde{P}(H,E) \leq \log \sum_A \Big[P(H,E) \cdot \Big(\prod_j z_j \Big)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}} \Big]^{1+\epsilon_{max}}$$

To get the relation between $\sum_A P(H,E)$ and $\sum_A \tilde{P}(H,E)$ we need to move the summation on the right hand side inside the exponentiated part. For this we use the following lemma

Lemma 3.1. *If $C_j \geq 0$ and $r \geq 1$, then*

$$\sum_j (C_j)^r \leq \left(\sum_j C_j\right)^r$$

If $C_j \geq 0$ and $0 < r \leq 1$, then

$$\sum_j (C_j)^r \geq \left(\sum_j C_j\right)^r$$

Therefore, we have

$$\begin{aligned}
\log \sum_A \tilde{P}(H,E) &\leq \log\left[\sum_A P(H,E) \cdot \left(\prod_j z_j\right)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}}\right]^{1+\epsilon_{max}} \\
&= \log\left[\left(\prod_j z_j\right)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}} \sum_A P(H,E)\right]^{1+\epsilon_{max}} \\
&= (1+\epsilon_{max}) \log \sum_A P(H,E) + \epsilon_{max} \sum_j \log z_j \qquad (3.7)
\end{aligned}$$

Equation (3.7) provides us with an upper bound on the log of the marginal probability. We now perform a very similar analysis to obtain a lower bound.

Again, from theorem 3.1, we have

$$\log \tilde{P}(H,E) \geq \log\left[P(H,E)\right]^{\frac{1}{1+\epsilon_{max}}} - \log\left(\prod_j z_j\right)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}}$$

Rearranging the right hand side of the above equation, we have

$$\log \tilde{P}(H,E) \geq \log \left[\frac{(P(H,E))^{\frac{1}{1+\epsilon_{max}}}}{(\prod_j z_j)^{\frac{\epsilon_{max}}{1+\epsilon_{max}}}} \right]$$
$$= \log \left[\frac{P(H,E)}{(\prod_j z_j)^{\epsilon_{max}}} \right]^{\frac{1}{1+\epsilon_{max}}}$$

Summing out a set $A \subseteq H$, we get

$$\log \sum_A \tilde{P}(H,E) \geq \log \sum_A \left[\frac{P(H,E)}{(\prod_j z_j)^{\epsilon_{max}}} \right]^{\frac{1}{1+\epsilon_{max}}}$$

Again we would like to move the summation inside the exponentiated part. Using lemma 3.1 we have

$$\log \sum_A \tilde{P}(H,E) \geq \log \left[\sum_A \frac{P(H,E)}{(\prod_j z_j)^{\epsilon_{max}}} \right]^{\frac{1}{1+\epsilon_{max}}}$$
$$= \log \left[\frac{1}{(\prod_j z_j)^{\epsilon_{max}}} \sum_A P(H,E) \right]^{\frac{1}{1+\epsilon_{max}}}$$
$$= \frac{1}{1+\epsilon_{max}} \log \sum_A P(H,E) - \frac{\epsilon_{max}}{1+\epsilon_{max}} \sum_j \log z_j \tag{3.8}$$

Equation (3.8) provides us with a lower bound on the log of the marginal probability. Combined with the upper bound on the log of the marginal probability (3.7), we can state the following theorem

Theorem 3.2. *For a set $A \subseteq H$, if the expression $\log \sum_A P(H,E)$ is approximated using a set of ϵ_j-decompositions on the factors ψ_j of P, then we have the following bounds:*

$$\log \sum_A \tilde{P}(H,E) \leq (1+\epsilon_{max}) \log \sum_A P(H,E) + \epsilon_{max} \sum_j \log z_j$$
$$\log \sum_A \tilde{P}(H,E) \geq \frac{1}{1+\epsilon_{max}} \log \sum_A P(H,E) - \frac{\epsilon_{max}}{1+\epsilon_{max}} \sum_j \log z_j$$

where $\epsilon_{max} = \max_j \epsilon_j$.

Theorem 3.2 differs from claim 3.2 in the fact that it has extra terms that are multiples of $\sum_j \log z_j$.

Theorems 3.1 and 3.2 do not disprove claims 3.1 and 3.2. We disprove these claims in section 3.4 where we provide simple counter examples that violate claims 3.1 and 3.2 but do not violate theorems 3.1 and 3.2.

3.3.2 No Shifting of Functions

Now let us consider the scenario when all our functions ψ_j have values that lie between zero and one. An example of such a distribution is a Bayesian Network with no zero probabilities. In this situation, we have the following theorem

Theorem 3.3. *If* $0 < \psi_j(D_j) < 1 \quad \forall D_j \quad \forall j$ *and the log of the joint probability* $P(H, E)$ *is approximated using a set of* ϵ_j*-decompositions on the factors* ψ_j *of* P*, then we have the following bounds:*

$$\frac{1}{1+\epsilon_{max}} \log P(H,E) \geq \log \tilde{P}(H,E) \geq (1+\epsilon_{max}) \log P(H,E) \tag{3.9}$$

where $\epsilon_{max} = \max_j \epsilon_j$.

Proof:

$$\log \tilde{P}(H,E) \equiv \log \prod_{j,l} e^{\tilde{\phi_{jl}}(D_{jl})} = \sum_{j,l} \tilde{\phi_{jl}}(D_{jl})$$

Now we know that

$$0 < \psi_j(D_j) < 1 \Longrightarrow \phi_j(D_j) < 0$$

Using the above fact and the definition of an ϵ-decomposition (3.1), we have

$$\begin{aligned}\log \tilde{P}(H,E) = \sum_{j,l} \tilde{\phi_{jl}}(D_{jl}) &\geq \sum_{j} (1+\epsilon_j)\phi_j(D_j) \\ &\geq (1+\epsilon_{max}) \sum_{j} \phi_j(D_j) \\ &= (1+\epsilon_{max}) \log P(H,E)\end{aligned}$$

and

$$\begin{aligned}\log \tilde{P}(H,E) = \sum_{j,l} \tilde{\phi_{jl}}(D_{jl}) &\leq \sum_{j} \frac{1}{1+\epsilon_j}\phi_j(D_j) \\ &\leq \frac{1}{1+\epsilon_{max}} \sum_{j} \phi_j(D_j) \\ &= \frac{1}{1+\epsilon_{max}} \log P(H,E)\end{aligned}$$

□

Theorem 3.3 allows us to bound the probability of an assignment. Now we look to bound the marginal probability. From theorem 3.3 we have

$$\log \tilde{P}(H,E) \geq \log \Big[P(H,E) \Big]^{(1+\epsilon_{max})}$$

Summing out a set $A \subseteq H$, we have

$$\log \sum_{A} \tilde{P}(H,E) \geq \log \sum_{A} \Big[P(H,E) \Big]^{1+\epsilon_{max}}$$

To get a relation between $\sum_A P(H,E)$ and $\sum_A \tilde{P}(H,E)$ we would like to move the summation sign inside the exponentiated part. Unfortunately since $(1+\epsilon_{max}) > 1$, lemma 3.1 can only provide an upper bound on the sum in the right hand side.

Similarly, looking at the upper bound from theorem 3.3, we have

$$\log \tilde{P}(H, E) \leq \log \Big[P(H, E) \Big]^{\frac{1}{1+\epsilon_{max}}}$$

Summing out a set $A \subseteq H$, we have

$$\log \sum_{A} \tilde{P}(H, E) \leq \log \sum_{A} [P(H, E)]^{\frac{1}{1+\epsilon_{max}}}$$

Again we try to move the summation inside the exponentiation. This is again not possible because lemma 3.1 can only provide a lower bound on the sum in the right hand side since $0 < \frac{1}{1+\epsilon_{max}} \leq 1$.

Hence we are unable to get any bounds on the log of the marginal probability $\sum_A P(H, E)$.

Until this point we have been unable to prove the claims made by Wexler and Meek. In the next section we will prove that the claims are actually false by providing simple counterexamples.

3.4 Example

In our example, we consider a simple model wherein we have two binary valued random variables A and B, and their joint probability is given by

$P(A, B) =$

	$B = 0$	$B = 1$
$A = 0$	0.3	0.25
$A = 1$	0.25	0.2

Let us assume that we find an approximation $\tilde{P}(A, B)$ to $P(A, B)$ as follows

$$\tilde{P}(A,B) = \begin{array}{c|cc} & B=0 & B=1 \\ \hline A=0 & 0.285 & 0.2525 \\ A=1 & 0.25 & 0.2125 \end{array}$$

Using these two equations and the definition of an ϵ-decomposition (3.1), we have $\epsilon_{max} = 0.0426$.

With this value of ϵ_{max}, we look at the bounds for the joint probability $P(A, B = 1)$. We can see that the equation specified by claim 3.1

$$\frac{1}{1+\epsilon_{max}} \log P(A, B=1) \leq \log \tilde{P}(A, B=1) \leq (1+\epsilon_{max}) \log P(A, B=1)$$

does not hold because we have

$$\begin{array}{c|ccccc} A=0 & -1.329 & & -1.376 & & -1.445 \\ & & \geq & & \geq & \\ A=1 & -1.544 & & -1.549 & & -1.678 \end{array}$$

whereas the equation specified by theorem 3.3

$$\frac{1}{1+\epsilon_{max}} \log P(A, B=1) \geq \log \tilde{P}(A, B=1) \geq (1+\epsilon_{max}) \log P(A, B=1)$$

does hold.

Similarly consider the procedure outlined in sub-section 3.3.1, which involves shifting and renormalizing the function. To shift $P(A, B)$ so that all its values are greater than one, we multiply it with the constant $z = 5.1$.

Therefore, we have

$$F(A,B) = z \cdot P(A,B) = \begin{array}{c|cc} & B=0 & B=1 \\ \hline A=0 & 1.53 & 1.275 \\ A=1 & 1.275 & 1.02 \end{array}$$

And we have $\tilde{F}(A,B) = z \cdot \tilde{P}(A,B)$ as

$$\tilde{F}(A,B) = \begin{array}{c|cc} & B=0 & B=1 \\ \hline A=0 & 1.4535 & 1.28775 \\ A=1 & 1.275 & 1.08375 \end{array}$$

Using these two equations and the definition of an ϵ-decomposition (3.1), we have $\epsilon_{max} = 3.0614$. Again, we look at the joint probability $P(A, B=1)$. Once again we see that the bounds given by claim 3.1

$$\frac{1}{1+\epsilon_{max}} \log P(A,B=1) \leq \log \tilde{P}(A,B=1) \leq (1+\epsilon_{max}) \log P(A,B=1)$$

do not hold because

$$\begin{array}{c|ccccc} A=0 & -0.341 & \not\leq & -1.376 & \not\leq & -5.630 \\ A=1 & -0.396 & & -1.549 & & -6.537 \end{array}$$

whereas the bounds given by theorem 3.1

$$\log \tilde{P}(A,B=1) \geq \frac{1}{1+\epsilon_{max}} \log P(A,B=1) - \frac{\epsilon_{max}}{1+\epsilon_{max}}$$
$$\log \tilde{P}(A,B=1) \leq (1+\epsilon_{max}) \log P(A,B=1) + \epsilon_{max} \log z$$

do hold, since

$$\begin{array}{c|ccccc} A=0 & -1.569 & & -1.376 & & -0.642 \\ & & \leq & & \leq & \\ A=1 & -1.624 & & -1.549 & & -1.548 \end{array}$$

Hence, we can state that claim 3.1 does not hold in general.

Having focussed on joint probabilities, we now look to perform marginalization to obtain bounds on the marginal probability. Referring to our original definition of $P(A, B)$, we can calculate $P(B)$ by marginalizing out A. Therefore, we have

$$P(B) = \begin{array}{c|c} 0.55 & B=0 \\ 0.45 & B=1 \end{array}$$

From $\tilde{P}(A, B)$, we have

$$\tilde{P}(B) = \begin{array}{c|c} 0.535 & B=0 \\ 0.465 & B=1 \end{array}$$

First we consider the case when $\epsilon_{max} = 0.0426$ was calculated directly using $P(A, B)$ and $\tilde{P}(A, B)$. We see that the equation specified by claim 3.2

$$\frac{1}{1+\epsilon_{max}} \log P(B) \leq \log \tilde{P}(B) \leq (1+\epsilon_{max}) \log P(B)$$

does not hold because

$$\begin{array}{c|ccccc} B=0 & -0.5734 & > & -0.6255 & < & -0.6233 \\ B=1 & -0.7659 & < & -0.7657 & > & -0.8325 \end{array}$$

Next we consider the case when $\epsilon_{max} = 3.0614$ was calculated using the shift and renormalize method. We see that the equation given by claim 3.2 still does not hold

because

$$\begin{array}{c|ccc} B=0 & -0.1472 & \not\leq & -0.6255 & \not\leq & -2.4281 \\ B=1 & -0.1966 & & -0.7657 & & -3.2431 \end{array}$$

whereas the bounds given by theorem 3.2

$$\frac{1}{1+\epsilon_m}\log P(B) - \frac{\epsilon_m}{1+\epsilon_m}\log z \leq \log \tilde{P}(B) \leq (1+\epsilon_m)\log P(B) + \epsilon_m \log z$$

where $\epsilon_m = \epsilon_{max}$, do hold because

$$\begin{array}{c|ccc} B=0 & -1.3753 & \leq & -0.6255 & \leq & 2.5597 \\ B=1 & -1.4247 & & -0.7657 & & 1.7447 \end{array}$$

Hence, we can state that claim 3.2 does not hold in general.

3.5 Tracking ϵ in an Inference Procedure

In the course of an inference procedure, we come across three major operations

1. Sum out a variable: We have a function $f(X)$ and we want to marginalize a subset $A \subseteq X$. In section 3.3, we considered this scenario and presented the effects it has on the error ϵ.

2. Multiply two functions: Assume that we have two functions f and g. Let $\tilde{f}$ be an approximation to f with error ϵ_1 and let $\tilde{g}$ be an approximation to g with error ϵ_2. We would like the error of $\tilde{f} \cdot \tilde{g}$ with respect to $f \cdot g$.

3. Compounded Approximation: Let us say that we have an approximation $\tilde{f}$ to a function f with error ϵ_1. Now assume that we get an approximation $\hat{f}$ to $\tilde{f}$

with error ϵ_2. We would like the error of $\hat{f}$ with respect to the original function f.

Let us first consider the case of multiplying two functions. From the definition of an ϵ-decomposition (3.1), we have

$$\frac{1}{1+\epsilon_1} \leq \frac{\log \tilde{f}}{\log f} \leq 1+\epsilon_1$$

and

$$\frac{1}{1+\epsilon_2} \leq \frac{\log \tilde{g}}{\log g} \leq 1+\epsilon_2$$

From these two equations we can say

$$\frac{1}{1+\epsilon_1}\log f + \frac{1}{1+\epsilon_2}\log g \leq \log \tilde{f} + \log \tilde{g} \leq (1+\epsilon_1)\log f + (1+\epsilon_2)\log g$$

Hence, we have

$$\frac{1}{1+\max\{\epsilon_1,\epsilon_2\}}(\log f + \log g) \leq \log \tilde{f} + \log \tilde{g} \leq (1+\max\{\epsilon_1,\epsilon_2\})(\log f + \log g)$$

We can rewrite the above equation as

$$\frac{1}{1+\max\{\epsilon_1,\epsilon_2\}} \leq \frac{\log \tilde{f}\tilde{g}}{\log fg} \leq (1+\max\{\epsilon_1,\epsilon_2\})$$

Therefore the error of $\tilde{f}\cdot\tilde{g}$ with respect to $f\cdot g$ is given by $\epsilon = \max\{\epsilon_1,\epsilon_2\}$.

Now we consider the case of performing a compounded approximation. Since $\tilde{f}$ is an approximation to f with error ϵ_1, we have

$$\frac{1}{1+\epsilon_1} \leq \frac{\log \tilde{f}}{\log f} \leq 1+\epsilon_1$$

Also, since $\hat{f}$ is an approximation to $\tilde{f}$ with error ϵ_2, we have

$$\frac{1}{1+\epsilon_2} \leq \frac{\log \hat{f}}{\log \tilde{f}} \leq 1+\epsilon_2$$

Multiplying these two equations, we have

$$\frac{1}{1+\epsilon_1} \cdot \frac{1}{1+\epsilon_2} \leq \frac{\log \tilde{f}}{\log f} \cdot \frac{\log \hat{f}}{\log \tilde{f}} \leq (1+\epsilon_1) \cdot (1+\epsilon_2)$$

Simplifying the above equation we get

$$\frac{1}{1+(\epsilon_1+\epsilon_2+\epsilon_1\epsilon_2)} \leq \frac{\log \hat{f}}{\log f} \leq 1+(\epsilon_1+\epsilon_2+\epsilon_1\epsilon_2)$$

Therefore the error of $\hat{f}$ with respect to f is given by $\epsilon = (\epsilon_1+\epsilon_2+\epsilon_1\epsilon_2)$.

To understand how the error can be tracked during inference, we provide a schematic description of the error tracking procedure in Figure 3.1. We start with four functions (f_a, f_b, f_c, f_d) each associated with zero error, $r_i = 1$. Every multiplication operation is denoted by edges directed from the nodes S, representing the multiplied functions, to a node t representing the resulting function with error $r_t = \max_{s \in S} r_s$. An ϵ-decomposition on the other hand has a single source node s with an associated error r_s, representing the decomposed function, and several target nodes T, with an error $r_t = (1+\epsilon)r_s$ for every $t \in T$. Therefore, the error associated with the result of the entire procedure is the error associated with the final node f_o in the graph. In our example we make the assumption that $\epsilon_1 > \epsilon_2$ and that $1+\epsilon_1 < (1+\epsilon_1)(1+\epsilon_2)$.

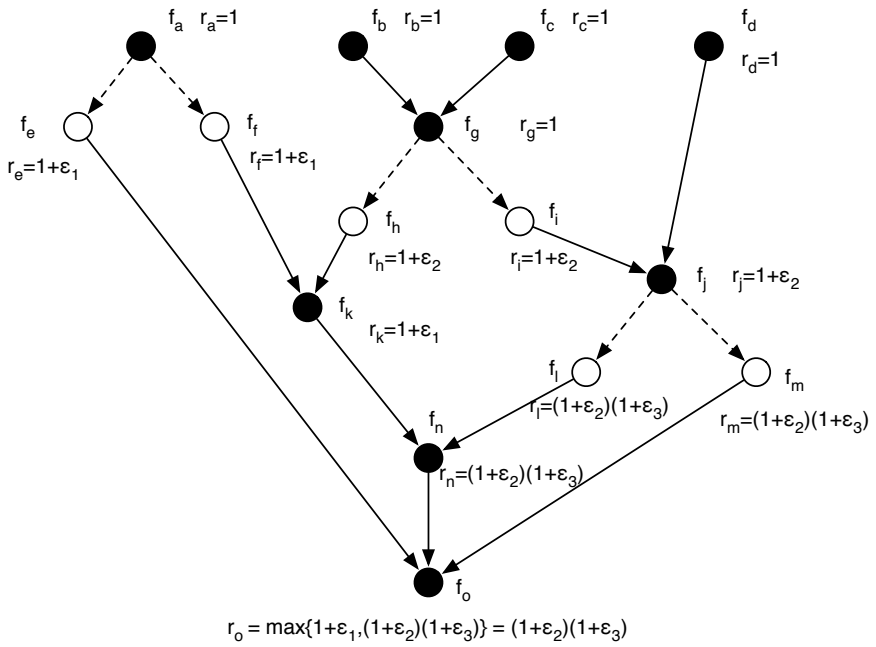

Figure 3.1: Schematic description of error tracking in an inference procedure using ϵ-decompositions

3.6 Need for new bounds

The corrected bounds of theorems 3.1 and 3.2 all contain the term $\sum_j \log z_j$ in some form. The value of this term depends on the functions in our model that have $\psi_j(D_j) < 1$. As a result it can become arbitrarily large when dealing with small probabilities. For example if we have a probability value of 0.01 we will have to multiply that factor with a value that is greater than 100. This value will then dominate the error in our bounds and hence will lead to very loose bounds.

We saw an example of this in section 3.4. The corrected upper bound on the log marginal probability was positive, which states that the marginal probability has an upper bound of one. Although not seen in our examples, in our experiments we will show instances wherein the corrected lower bound for a probability is practically zero.

Since the corrected bounds resulting from the multiplicative error measure 3.1 are

often loose, we are motivated to investigate alternate error measures. In chapter 4 we propose one such measure, based on the L^∞ norm. We show how this measure can be used to compute the error of a local decomposition and then aggregated into a bound on the result.

Chapter 4

MAS Error Bounds using the L^∞ norm

In chapter 3, section 3.6, we stated the need to investigate alternate error measures since the corrected bounds resulting from the multiplicative error measure are often quite loose. Here we use a measure that is based on the L^∞ norm. Using the L^∞ error measure, we derive new bounds on the accuracy of the results of an inference procedure.

In section 4.1 we introduce the L^∞ norm and show how it can be used to compute the error of a local decomposition. Then, in section 4.2 we show how to track the new error measure in an inference procedure. Finally, in section 4.3, we derive new bounds for the log joint probability and log marginal probability under MAS.

4.1 The L^∞ norm

The L^∞ norm can be used to measure the difference between two functions. Suppose we have a function $f(X)$ and an approximation $\tilde{f}(X)$ to $f(X)$. The L^∞ norm of the difference of logs is defined as

$$\left\|\log f(X) - \log \tilde{f}(X)\right\|_\infty = \sup_X \left|\log f(X) - \log \tilde{f}(X)\right|$$

One of the problems with using the L^∞ norm to measure error is that optimizing the L^∞ norm to get the best $\tilde{f}(X)$ is hard. So, the strategy we use is that assuming we have some approximation $\hat{f}(X)$ to $f(X)$ we will perform a scalar shift by α on $\log \hat{f}(X)$ so that we minimize the L^∞ norm. This means that we replace our approximation $\hat{f}(X)$ with a new approximation $\tilde{f}(X)$ given by

$$\tilde{f}(X) = \frac{\hat{f}(X)}{\alpha}$$

We define our error measure δ as

$$\delta = \sup_X \left|\log f(X) - \log \tilde{f}(X)\right|$$

which is equivalent to

$$\delta = \inf_\alpha \sup_X \left|\log f(X) - \log \frac{\hat{f}(X)}{\alpha}\right|$$

The error measure δ we defined is equal to the log of the dynamic range which is a metric introduced by Ihler et al in [11]. An illustration of how δ can be calculated is shown in Figure 4.1. It shows a function $f(X)$ with its approximation $\hat{f}(X)$, and depicts the log-error $\log f/\hat{f}$, error measure δ and the minimizing α.

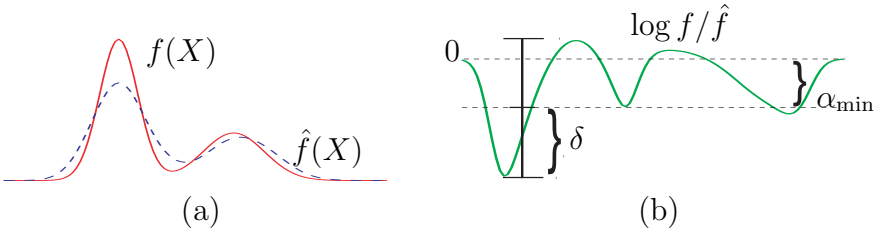

Figure 4.1: (a) A function $f(X)$ and an example approximation $\hat{f}(X)$; (b) their log-ratio $\log f(X)/\hat{f}(X)$, and the error measure δ.

With this error measure, we now define a δ-decomposition as follows

Definition 4.1. *Given a set of variables X, and a function $\phi(X)$ that assigns real values to every instantiation of X, a set of m functions $\hat{\phi}_l(X_l)$, $l = 1 \ldots m$, where $X_l \subseteq X$, is an δ-decomposition if $\bigcup_l X_l = X$, and for all instantiations of X the following holds:*

$$\delta = \inf_{\alpha} \sup_{X} \left| \log \alpha + \phi(X) - \sum_{l} \hat{\phi}_l(X_l) \right| \tag{4.1}$$

Having defined the new error measure δ, in the next section we analyze how the different operations of an inference procedure affect δ.

4.2 Tracking δ in an Inference Procedure

As in chapter 3, we consider how the three major operations of an inference procedure, summing out a set of variables, multiplying two functions and compounded approximations, affect the new error measure δ.

First we consider the operation of summing out a set of variables. The problem is stated as follows: Given a function $f(X)$ and approximation $\tilde{f}(X)$ with error δ we want the error of $\sum_A \tilde{f}(X)$ with respect to $\sum_A f(X)$ where $A \subseteq X$.

We have

$$\left|\log f(X) - \log \tilde{f}(X)\right| \leq \delta \quad \forall X$$

From the above equation we have

$$\frac{1}{e^{\delta}} \leq \frac{f(X)}{\tilde{f}(X)} \leq e^{\delta} \quad \forall X$$

Therefore, we have

$$\frac{\tilde{f}(X)}{e^{\delta}} \leq f(X) \qquad\qquad f(X) \leq e^{\delta}\tilde{f}(X)$$

Summing out the set A, we get

$$\sum_{A} \frac{\tilde{f}(X)}{e^{\delta}} \leq \sum_{A} f(X) \qquad\qquad \sum_{A} f(X) \leq \sum_{A} e^{\delta}\tilde{f}(X)$$

Since δ is a constant we have

$$\frac{1}{e^{\delta}} \sum_{A} \tilde{f}(X) \leq \sum_{A} f(X) \qquad\qquad \sum_{A} f(X) \leq e^{\delta} \sum_{A} \tilde{f}(X)$$

Therefore we have

$$\frac{1}{e^{\delta}} \leq \frac{\sum_A f(X)}{\sum_A \tilde{f}(X)} \leq e^{\delta}$$

which is equivalent to

$$\left|\log \sum_{A} f(X) - \log \sum_{A} \tilde{f}(X)\right| \leq \delta$$

Therefore, summing out a set of variables $A \subseteq X$ has an error of at most δ for the approximation.

Next, consider multiplying two functions. We have a function f with approximation $\tilde{f}$ and error δ_1. We have another function g with approximation $\tilde{g}$ and error δ_2. We want the error of $\tilde{f} \cdot \tilde{g}$ with respect to $f \cdot g$. We have

$$\left|\log f - \log \tilde{f}\right| \leq \delta_1$$
$$\left|\log g - \log \tilde{g}\right| \leq \delta_2$$

We are interested in

$$\left|\log fg - \log \tilde{f}\tilde{g}\right| = \left|\log f - \log \tilde{f} + \log g - \log \tilde{g}\right|$$

By the triangle inequality we have

$$\begin{aligned}\left|\log fg - \log \tilde{f}\tilde{g}\right| &\leq \left|\log f - \log \tilde{f}\right| + \left|\log g - \log \tilde{g}\right| \\ &\leq \delta_1 + \delta_2\end{aligned}$$

Therefore the error of $\tilde{f} \cdot \tilde{g}$ with respect to $f \cdot g$ is at most $\delta = \delta_1 + \delta_2$.

Finally consider the case of a compounded approximation. Assume we have an approximation $\tilde{f}$ to f with error δ_1. Now assume that we get an approximation $\hat{f}$ to $\tilde{f}$ with error δ_2. We want the error of $\hat{f}$ with respect to f. We have

$$\left|\log f - \log \tilde{f}\right| \leq \delta_1$$
$$\left|\log \tilde{f} - \log \hat{f}\right| \leq \delta_2$$

We are interested in

$$\left|\log f - \log \hat{f}\right| = \left|\log f - \log \tilde{f} + \log \tilde{f} - \log \hat{f}\right|$$

By the triangle inequality we have

$$\begin{aligned}\left|\log f - \log \hat{f}\right| &\leq \left|\log f - \log \tilde{f}\right| + \left|\log \tilde{f} - \log \hat{f}\right| \\ &\leq \delta_1 + \delta_2\end{aligned}$$

Therefore the error of $\hat{f}$ with respect to f is at most $\delta = \delta_1 + \delta_2$.

To illustrate the error tracking for δ-decompositions in an inference procedure, we provide a schematic trace in Figure 4.2. Since the error measure δ has been shown to be at most additive, we only need to keep track of the error in each decomposition and add all the errors at the end. In Figure 4.2 there are three decompositions that take place: f_a is decomposed with an error δ_1, f_g is decomposed with an error δ_2 and f_j is decomposed with an error δ_3. Therefore, the error on the result of the inference procedure is at most $\delta_1 + \delta_2 + \delta_3$.

In the next section section we use the definition of a δ-decomposition to derive new bounds for the log joint probability and the log marginal probability.

4.3 New Bounds for MAS

There are two scenarios for us to consider

1. $P(X)$ is a normalized probability distribution, i.e.,

$$P(X) = \frac{1}{Z}\prod_j \psi_j(D_j), \quad D_j \subseteq X$$

   and we know the value of the normalization constant Z.

2. $P(X)$ is not normalized and we need to compute the normalizing constant Z.

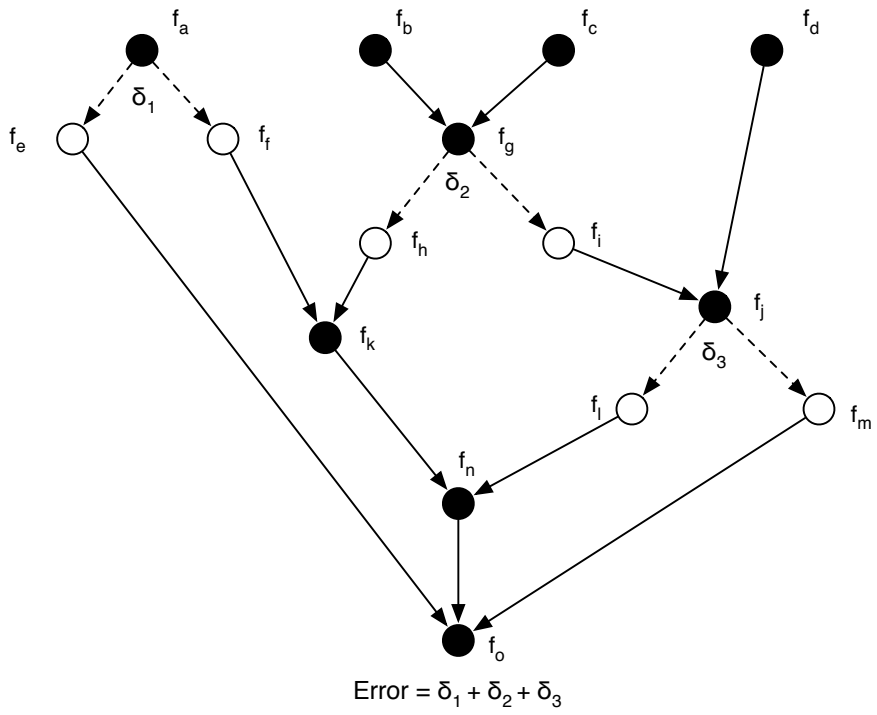

Figure 4.2: Schematic description of error tracking in an inference procedure using δ-decompositions

First we consider the case when $P(X)$ is a normalized probability distribution. We start with deriving bounds for the log of the joint probability $P(H, E)$. Assume that for each factor ψ_j we have a δ-decomposition. Then, we have

$$\left| \phi_j(D_j) - \sum_l \tilde{\phi}_{jl}(D_{jl}) \right| \leq \delta_j \quad \forall j$$

We have shown in section 4.2 that multiplication of functions causes the errors to at most add. Since multiplying the original factors is equivalent to summing the log of the factors, we have

$$\left| \sum_j \phi_j(D_j) - \sum_{j,l} \tilde{\phi}_{jl}(D_{jl}) \right| \leq \sum_j \delta_j$$

Since $P(X)$ is normalized we have

$$\left|\left(\sum_j \phi_j(D_j) - \log Z\right) - \left(\sum_{j,l} \tilde{\phi}_{jl}(D_{jl}) - \log Z\right)\right| \leq \sum_j \delta_j$$

We define $\tilde{P}(H, E)$ as

$$\tilde{P}(H, E) = \frac{1}{Z} \prod_{j,l} e^{\tilde{\phi_{jl}}(D_{jl})}$$

Therefore, we have

$$\left|\log P(H, E) - \log \tilde{P}(H, E)\right| \leq \sum_j \delta_j$$

From the above equation we can state the following theorem

Theorem 4.1. *If the log of the joint probability $P(H, E)$ is approximated using a set of δ_j-decompositions on the factors ψ_j of P and P is a normalized distribution, then*

$$\log P(H, E) - \sum_j \delta_j \leq \log \tilde{P}(H, E) \leq \log P(H, E) + \sum_j \delta_j \tag{4.2}$$

Having derived bounds for the joint probability, we proceed towards deriving bounds for marginal probabilities. From theorem 4.1 we have

$$\left|\log P(H, E) - \log \tilde{P}(H, E)\right| \leq \sum_j \delta_j$$

In section 4.2 we showed that for a set $A \subseteq X$ we have

$$\left|\log \sum_A f(X) - \log \sum_A \tilde{f}(X)\right| \leq \delta$$

Therefore, summing out a set $A \subseteq H$ from $P(H, E)$, we get

$$\left| \log \sum_A P(H, E) - \log \sum_A \tilde{P}(H, E) \right| \leq \sum_j \delta_j$$

From the above equation we have the following theorem

Theorem 4.2. *For a set $A \subseteq H$, if the expression $\log \sum_A P(H, E)$ is approximated using a set of δ_j-decompositions on the factors ψ_j of P and P is a normalized distribution, then*

$$\log \sum_A P(H, E) - \sum_j \delta_j \leq \log \sum_A \tilde{P}(H, E) \leq \log \sum_A P(H, E) + \sum_j \delta_j \qquad (4.3)$$

We now consider the case when $P(X)$ is not a normalized distribution. Assume we have a function $f(X)$ and an approximation $\tilde{f}(X)$ to $f(X)$ with error δ. We have

$$\left| \log f(X) - \log \tilde{f}(X) \right| \leq \delta$$

Assume that we cannot calculate the real normalization constant $Z = \sum_X f(X)$. We can calculate an approximation to the normalization constant as $\tilde{Z} = \sum_X \tilde{f}(X)$. We want a relation between our normalized approximation $\tilde{f}(X)/\tilde{Z}$ and the real normalized function $f(X)/Z$, i.e., we want to find bounds for

$$\left| \log \frac{f(X)}{Z} - \log \frac{\tilde{f}(X)}{\tilde{Z}} \right| \qquad (4.4)$$

In section 4.2 we showed that for a set $A \subseteq X$ we have

$$\left| \log \sum_A f(X) - \log \sum_A \tilde{f}(X) \right| \leq \delta$$

Since $X \subseteq X$, we have

$$\left| \log \sum_{X} f(X) - \log \sum_{X} \tilde{f}(X) \right| \leq \delta$$

Substituting $Z = \sum_X f(X)$ and $\tilde{Z} = \sum_X \tilde{f}(X)$ we have

$$\left| \log Z - \log \tilde{Z} \right| \leq \delta$$

We can rewrite the normalized expression of equation 4.4 as

$$\left| \log \frac{f(X)}{Z} - \log \frac{\tilde{f}(X)}{\tilde{Z}} \right| = \left| \log f(X) - \log \tilde{f}(X) + \log \tilde{Z} - \log Z \right|$$

By triangle inequality we have

$$\begin{aligned} \left| \log \frac{f(X)}{Z} - \log \frac{\tilde{f}(X)}{\tilde{Z}} \right| &\leq \left| \log f(X) - \log \tilde{f}(X) \right| + \left| \log \tilde{Z} - \log Z \right| \\ &\leq 2\delta \end{aligned}$$

Hence, doing the normalization increases the error by a factor of at most two.

Now assuming that for each factor ψ_j of P we have a δ-decomposition, we have

$$\left| \sum_{j} \phi_j(D_j) - \sum_{j,l} \tilde{\phi}_{jl}(D_{jl}) \right| \leq \sum_{j} \delta_j$$

Since we do not know the real normalization constant Z we normalize the result to get

$$\left| \left(\sum_{j} \phi_j(D_j) - \log Z \right) - \left(\sum_{j,l} \tilde{\phi}_{jl}(D_{jl}) - \log \tilde{Z} \right) \right| \leq 2 \sum_{j} \delta_j$$

In this scenario we define $\tilde{P}(H, E)$ as

$$\tilde{P}(H, E) = \frac{1}{\tilde{Z}} \prod_{j,l} e^{\tilde{\phi_{jl}}(D_{jl})}$$

Therefore, we have

$$\left| \log P(H, E) - \log \tilde{P}(H, E) \right| \leq 2 \sum_{j} \delta_j$$

From the above equation we have the following theorem:

Theorem 4.3. *If the log of the joint probability $P(H, E)$ is approximated using a set of δ_j-decompositions on the factors ψ_j of P and P is not a normalized distribution, then*

$$\log P(H, E) - 2 \sum_{j} \delta_j \leq \log \tilde{P}(H, E) \leq \log P(H, E) + 2 \sum_{j} \delta_j \tag{4.5}$$

Since we have shown that summing out a set of variables from a function does not affect the error of the approximation, we can state the theorem for the log of the marginal probabilities as follows:

Theorem 4.4. *For a set $A \subseteq H$, if the expression $\log \sum_A P(H, E)$ is approximated using a set of δ_j-decompositions on the factors ψ_j of P and P is not a normalized distribution, then*

$$\log \sum_{A} P(H, E) - 2 \sum_{j} \delta_j \leq \log \sum_{A} \tilde{P}(H, E) \leq \log \sum_{A} P(H, E) + 2 \sum_{j} \delta_j \tag{4.6}$$

Chapter 5

Optimizing a Decomposition

In chapters 3 and 4, we provided methods to calculate error bounds given a particular approximation. In this chapter we present methods for finding a decomposition of a function.

Ideally we would like to find the decomposition of a function that has the smallest value of the error measure ϵ or δ. There are two problems we have to consider namely, how can we select good subsets $\{X_1, \ldots, X_m\}$ for the decomposition, and given the decomposition $\{X_1, \ldots, X_m\}$, how can we select the functions $\{\tilde{\phi_1}(X_1), \ldots, \tilde{\phi_m}(X_m)\}$.

Choosing good subsets is an interesting but difficult problem. In this thesis, we take the option of choosing subsets $\{X_1, \ldots, X_m\}$ at random and then find good values for the functions $\{\tilde{\phi_1}(X_1), \ldots, \tilde{\phi_m}(X_m)\}$.

Given a decomposition $\{X_1, \ldots, X_m\}$, the ideal way to choose $\{\tilde{\phi_1}(X_1), \ldots, \tilde{\phi_m}(X_m)\}$ would be to choose the functions that minimize our error measure. Considering the

error measure ϵ defined in chapter 3 the problem we would like to solve is

$$\min_{(\tilde{\phi_1},\ldots,\tilde{\phi_m})} \max_X \left\{ \frac{\sum_i \tilde{\phi}_i(X_i)}{\phi(X)}, \frac{\phi(X)}{\sum_i \tilde{\phi}_i(X_i)} \right\} \tag{5.1}$$

If we use the error measure δ defined in chapter 4, then the problem would be

$$\min_{(\tilde{\phi_1},\ldots,\tilde{\phi_m})} \sup_X \left| \phi(X) - \sum_i \tilde{\phi}_i(X_i) \right| \tag{5.2}$$

Since optimizing either of these two objective functions is not easy, we look to optimize some similar measure in the hope that we will get small value of our error measure. Wexler and Meek suggest the L^2 norm because for a vector X, $\|X\|_2 \geq \|X\|_\infty$. The L^2 norm for a decomposition is defined as follows:

$$\min_{(\tilde{\phi_1},\ldots,\tilde{\phi_m})} \sqrt{\sum_X \left[\left(\sum_i \tilde{\phi}_i(X_i) \right) - \phi(X) \right]^2} \tag{5.3}$$

We can choose disjoint or overlapping subsets for the decomposition. In section 5.1 we consider the case wherein we are restricted to choosing disjoint subsets, i.e., each $\tilde{\phi}_i$ is defined on a non-overlapping subset of the variables. More formally we choose sets $\{X_1, \ldots, X_m\}$ such that

$$\forall i, j = 1, \ldots, m \quad X_i \cap X_j = \emptyset \Leftrightarrow i \neq j$$

Then in section 5.2 we remove the restriction of disjoint subsets and consider the more general case of allowing sets that overlap with each other.

5.1 Optimizing a decomposition with disjoint subsets

The L_2 norm is computationally convenient for disjoint subsets. The reason for this is that there exists a closed form solution to equation (5.3) when the sets are disjoint. Wexler and Meek obtained the closed form solution as follows. First they remove the square root since the square root function is monotonic for positive values. Hence the objective function becomes

$$\min_{(\tilde{\phi_1},\ldots,\tilde{\phi_m})} \sum_X \left[\left(\sum_i \tilde{\phi}_i(X_i) \right) - \phi(X) \right]^2$$

Next Wexler and Meek differentiate the equation with respect to each $\tilde{\phi}_k(X_k)$ and set it to zero. The resulting set of equations is under constrained so they use the arbitrary constraint

$$\sum_X \tilde{\phi}_i(X_i) = \frac{\sum_X \phi(X)}{m}$$

to make the system well posed. Solving the resulting set of equations they get the closed form solution as

$$\tilde{\phi}_k(X_k) = \frac{\sum_{X \setminus X_k} \phi(X)}{\prod_{i \neq k} |X_i|} - \frac{(m-1) \sum_X \phi(X)}{m\,|X|} \tag{5.4}$$

5.2 Optimizing a decomposition with overlapping subsets

When we remove the restriction of disjoint subsets, we cannot find a closed form solution for minimizing the L^2 norm. So, we present a method that can generate overlapping decompositions from disjoint decompositions.

Given a function $\phi(X)$ we would like to generate functions defined over overlapping sets of variables. We create overlapping decompositions by first generating functions over two or more different sets of disjoint decompositions and then combining them together. We explain the procedure by considering two sets of disjoint decompositions $\{X_{11}, \ldots, X_{1p}\}$ and $\{X_{21}, \ldots, X_{2q}\}$.

First we generate a disjoint decomposition over $\{X_{11}, \ldots, X_{1p}\}$ for $\phi(X)$. To calculate the values for $\{\tilde{\phi}_{11}(X_{11}), \ldots, \tilde{\phi}_{1p}(X_{1p})\}$ we can use the closed form solution from equation (5.4). We now compute the residual $R(X)$ as

$$R(X) = \phi(X) - \sum_{i=1}^{p} \tilde{\phi}_{1i}(X_{1i})$$

Next we generate a disjoint decomposition over $\{X_{21}, \ldots, X_{2q}\}$ for $R(X)$ where the sets $\{X_{21}, \ldots, X_{2q}\}$ are different from the sets $\{X_{11}, \ldots, X_{1p}\}$. To calculate the values of $\{\tilde{R}_{21}(X_{21}), \ldots, \tilde{R}_{2q}(X_{2q})\}$ we again use the closed form solution from equation (5.4).

We now union the sets $\{\tilde{\phi}_{11}(X_{11}), \ldots, \tilde{\phi}_{1p}(X_{1p})\}$ and $\{\tilde{R}_{21}(X_{21}), \ldots, \tilde{R}_{2q}(X_{2q})\}$ to generate the combined set of functions $\{\tilde{\phi}_{11}(X_{11}), \ldots, \tilde{\phi}_{1p}(X_{1p}), \tilde{R}_{21}(X_{21}), \ldots, \tilde{R}_{2q}(X_{2q})\}$. In this combined set if there exists a function over a set X_j and another function over a set X_k such that $X_j \subseteq X_k$, then we multiply the two functions together and generate a single new function. After performing this multiplication operation on the

combined set we get an overlapping decomposition $\{\hat{\phi_1}(X_1), \ldots, \hat{\phi_m}(X_m)\}$.

To make the procedure of generating overlapping decompositions clearer, we provide the schematic trace of an example in Figure 5.1. We have a function ϕ that is defined over three variables X_1, X_2, X_3 for which we would like to generate overlapping decompositions. We look at all possible decompositions that can be made except for the trivial decomposition of three functions over single variables. At the first stage, we decompose ϕ which can be done in three different ways. At the second stage we decompose the residual. For each choice we made in the first stage we have two possible choices at this stage. Finally we combine all the functions together to generate the required overlapping decomposition. For each path down the tree from root to leaf, we compute the values of ϵ and δ for the resulting decomposition.

Having provided a theoretical analysis of MAS including new ways to calculate error bounds and new ways to optimize ϵ-decompositions, we now shift our focus to the application of MAS to inference algorithms. In the next chapter, we look at the application of MAS to bucket elimination [6]. We analyze the algorithm proposed by Wexler and Meek called DynaDecomp and discuss some of its limitations. We then present our own DynaDecompPlus algorithm which modifies the DynaDecomp algorithm so as to overcome its limitations.

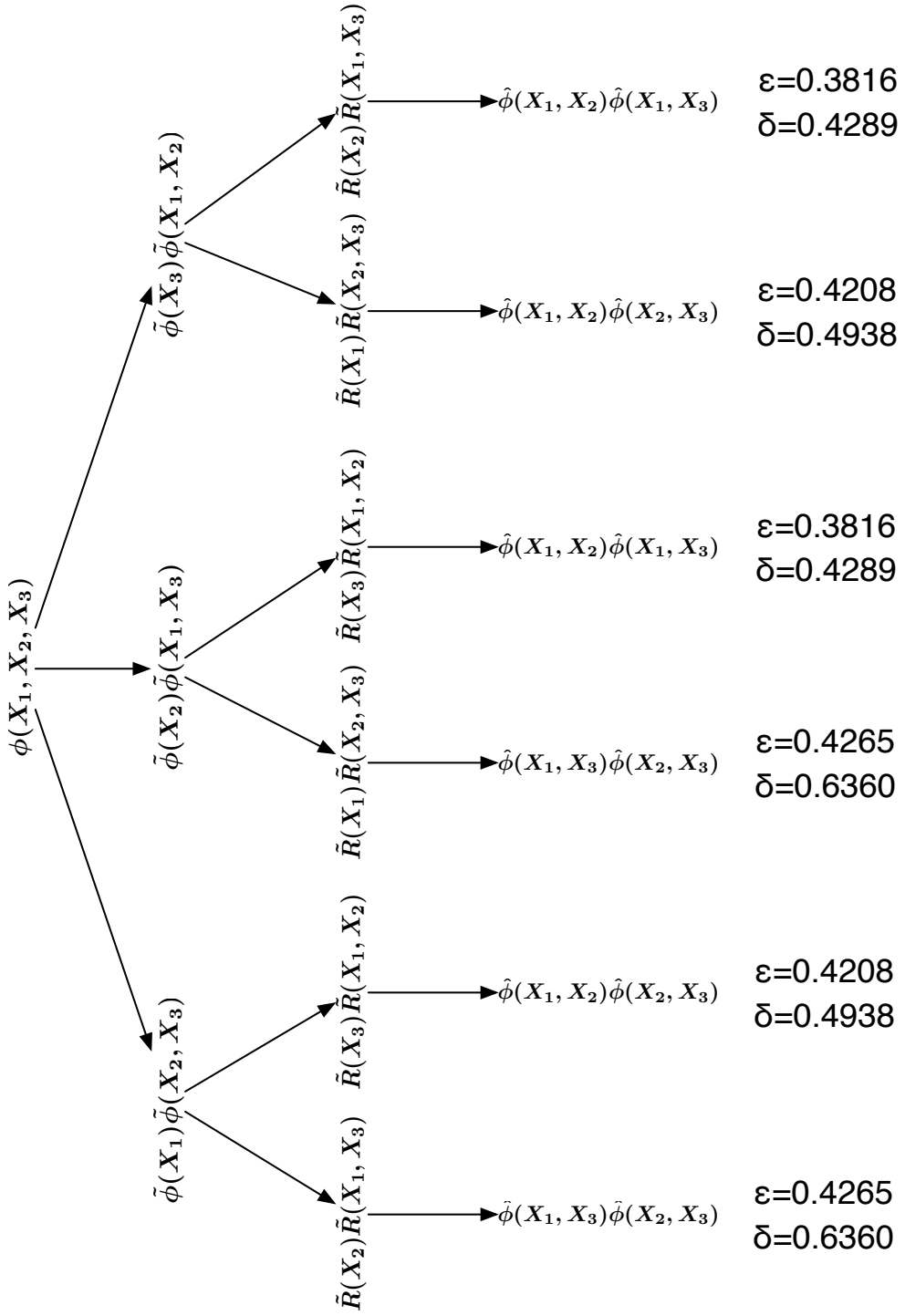

Figure 5.1: Schematic trace of generating overlapping decompositions

Chapter 6

Bucket Elimination with MAS

Many existing inference algorithms for graphical models compute and utilize multiplicative factors during the course of the inference procedure. As a result the multiplicative approximation scheme can be applied to these algorithms. In this chapter we look at one such application, using MAS with bucket elimination [6].

In chapter 2 (section 2.4) we provided an introduction to the bucket elimination framework. In section 6.1 we introduce and analyze the algorithm developed by Wexler and Meek called DynaDecomp and present its major limitation. Then, in section 6.2, we introduce our own algorithm called DynaDecomPlus which improves on the DynaDecomp algorithm by providing additional control over the computational complexity.

6.1 The DynaDecomp Algorithm

DynaDecomp is an algorithm developed by Wexler and Meek and presented in [24]. It performs variable elimination using MAS. The strategy that DynaDecomp uses for

Input: A model for n variables $X = \{X_1, \ldots, X_n\}$ and functions $\psi_j(D_j \subseteq X)$, that encodes $P(X) = \prod_j \psi_j(D_j)$; A set $E = X \backslash H$ of observed variables; An elimination order R over the variables in H; threshold M.
Output: The log-likelihood $\log P(E)$; an error ϵ.
Initialize: $\epsilon = 0$; $F \leftarrow \{\psi_j(D_j)\}$
for $i = 1$ *to* n **do**
  $k \leftarrow R[i]$
  $T \leftarrow$ All functions $f \in F$ that contain variable X_k
  $F \leftarrow F \backslash T$
  Eliminate X_k from functions in bucket T: $f' \leftarrow \sum_{X_k} \prod_{f_j \in T} f_j$
  if $|f'| > M$ **then**
    Decompose f' into a set $\tilde{F}$ and an error $\epsilon_{f'}$
    $F \leftarrow F \cup \tilde{F}$
    $\epsilon = \max\{\epsilon, \epsilon_{f'}\}$
  else
    $F \leftarrow F \cup f'$
  end
end
multiply all constant functions in F and put in p
return $\log p$, ϵ

Figure 6.1: The DynaDecomp Algorithm

decomposition is a simple one: If the size of a function exceeds a predefined threshold, decompose the function. The pseudocode for DynaDecomp is presented in Figure 6.1.

DynaDecomp begins exactly like bucket elimination. Given an elimination order we first assign the functions to their respective buckets and then start processing the buckets one after the other. The difference is that while we are at bucket i, if the function generated after eliminating variable X_i, h^{X_i}, has a size greater than a predefined threshold M, we decompose this function h^{X_i} into a set of smaller functions. A schematic trace of DynaDecomp is presented in Figure 6.2.

In our example we set the threshold $M = 3$, i.e., we will decompose any function that is defined over more than three variables. In figure 6.2 we see that the function $h^B(A, D, C, E)$ generated after eliminating variable B is defined over four variables. Hence, it is decomposed into two smaller functions $\tilde{h}_1^B(D, C)$ and $\tilde{h}_2^B(A, E)$. The

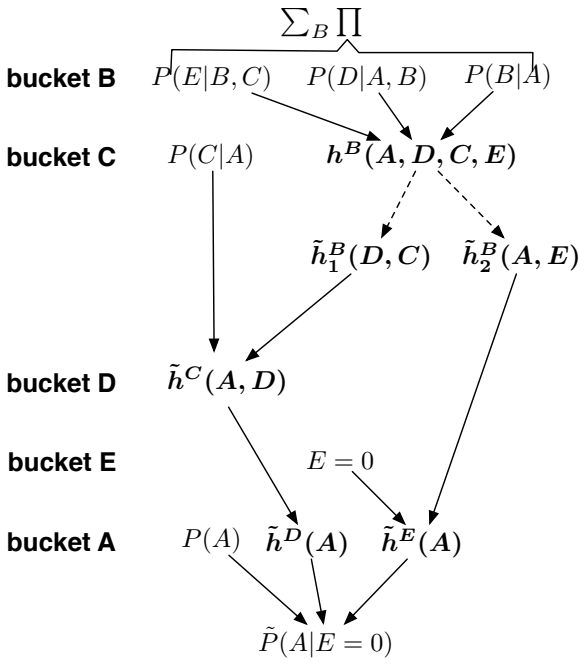

Figure 6.2: A trace of algorithm DynaDecomp

algorithm then proceeds as in standard bucket elimination and in the end we get an approximation $\tilde{P}(A|E=0)$ to $P(A|E=0)$.

One thing to notice is that even though we wanted to restrict ourselves to functions over three variables, the algorithm generated a function over 5 variables when we multiplied all the functions in bucket B. Hence, even though the DynaDecomp algorithm does provide savings later on in the inference procedure, it still generates functions greater than the threshold specified. As a result, it becomes quite hard to control the time and space complexity of DynaDecomp.

In order to provide greater control over the time and space complexity we modify DynaDecomp to create our own algorithm called DynaDecompPlus. In the next section we provide the details of our algorithm

6.2 Improvements to DynaDecomp

The major issue with the DynaDecomp algorithm is that it is very difficult to control the largest function size generated by the algorithm. Therefore we developed a modified version of DynaDecomp called DynaDecompPlus which ensures that functions larger in size than the given threshold are never generated.

Before presenting the details of DynaDecompPlus, we describe the small changes that have to be made to DynaDecomp so that we can use δ to track the error instead of ϵ.

6.2.1 Using L^∞ Bounds with DynaDecomp

Modifying DynaDecomp to use δ for error tracking is simple. The algorithm proceeds exactly as in Figure 6.1. The only difference is that to decompose the function f' into the set of functions $\tilde{F}$, we use a δ-decomposition (4.1) instead of an ϵ-decomposition (3.1) to calculate the error. Hence the error changes from $\epsilon = \max\{\epsilon, \epsilon_{f'}\}$ to $\delta = \delta + \delta f'$. With these changes, we can use the L^∞ bounds derived in chapter 4 to bound the returned result, $\log p$.

6.2.2 The DynaDecompPlus Algorithm

The DynaDecompPlus algorithm is presented in Figure 6.3. The basic principle behind the algorithm is that instead of decomposing the function resulting from processing a particular bucket, decompositions are performed while processing a bucket.

The idea is that before we multiply all the functions in a bucket, we check to see if the resulting function will exceed our threshold. If it does then instead of multiplying all the functions, we multiply as many functions as we can without exceeding the thresh-

Input: A model for n variables $X = \{X_1, \ldots, X_n\}$ and functions $\psi_j(D_j \subseteq X)$, that encodes $P(X) = \prod_j \psi_j(D_j)$; A set $E = X \backslash H$ of observed variables; An elimination order R over the variables in H; threshold M.
Output: The log-likelihood $\log P(E)$; an error δ.
Initialize: $\delta = 0$; $F \leftarrow \{\psi_j(D_j)\}$
for $i = 1$ *to* n **do**
  $k \leftarrow R[i]$
  $T \leftarrow$ All functions $f \in F$ that contain variable X_k
  $F \leftarrow F \backslash T$
  $a \leftarrow length(T)$; $f' \leftarrow 1$; $S \leftarrow \emptyset$
  for $j = 1$ *to* a **do**
    if *Multiplying* f' *and* $T[j]$ *will result in a function of size greater than* M **then**
      $S \leftarrow S \cup T[j]$
    else
      $f' \leftarrow$ Multiplication of f' and $T[j]$
    end
  end
  if S *is not empty* **then**
    $(\delta_{f'}, \tilde{F}) \leftarrow decompose(S, f')$
    $(f', \tilde{F}) \leftarrow subsume\_functions(f', \tilde{F})$
    $\delta \leftarrow \delta + \delta_{f'}$
  end
  Now we sum out variable X_k from f': $f' \leftarrow \sum_{X_k} f'$
  $F \leftarrow F \cup f'$
  $F \leftarrow F \cup \tilde{F}$
end
multiply all constant functions in F and put in p
return $\log p$, δ

Figure 6.3: The DynaDecompPlus Algorithm

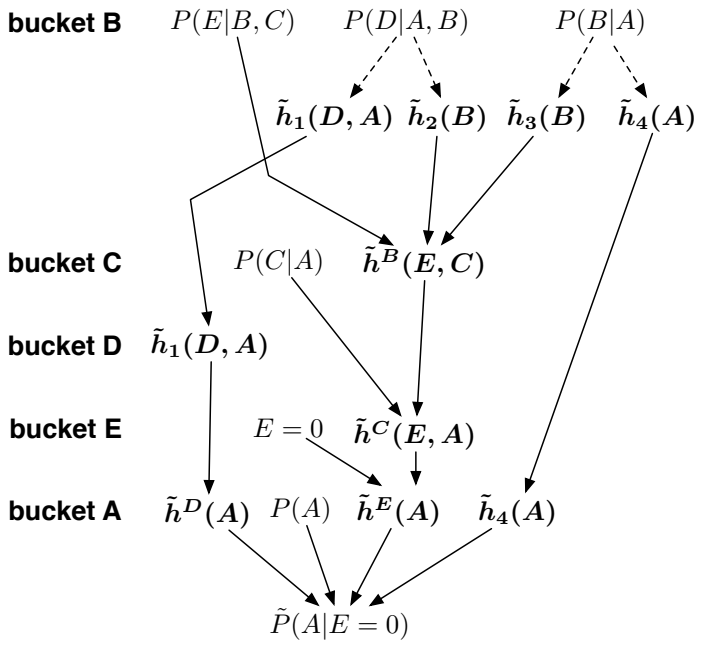

Figure 6.4: A trace of algorithm DynaDecompPlus

old. The rest of the functions are then decomposed and the variable is eliminated at the end.

To make this clearer, we look at the trace of the algorithm presented in Figure 6.4. The problem we are trying to solve is $P(A|E=0)$. But we do not want to generate any functions greater than size three. So given this threshold and the elimination order $o = (A, E, C, D, B)$, we start the DynaDecompPlus algorithm. At the very first bucket B, we see that if we multiply all the functions in this bucket, we will generate a function $h^B(A, B, C, D, E)$ which is greater than our threshold. So instead, we choose the decompose the functions $P(D|A, B)$ and $P(B|A)$. By performing the decomposition early and then proceeding with bucket elimination we can see that we never generated any function greater than size three. If we compare this to the trace of DynaDecomp in Figure 6.2, it also had set the maximum size to three, but since it first multiplied all the functions in bucket B, it actually generated a function of size five.

Note on subsume_functions

The DynaDecompPlus algorithm calls a method known as *subsume_functions*. This method is designed to take a set of functions and if there exists any function that is defined over a set of variables which are a subset of another function's variables, then it multiplies the two functions. For example if we had a set of functions that contained a function $f_1(A, B, C, D)$ and another function $f_2(B, D)$, the method subsume_functions will multiply these two functions and produce a new result $f_1'(A, B, C, D)$.

In the next chapter we provide the results of a variety of experiments we performed that compare the performance of DynaDecomp and DynaDecompPlus. We also compare the tightness of the corrected MAS bounds derived in chapter 3 with the L^∞ bounds derived in chapter 4.

Chapter 7

Experiments

In this chapter we present results of a variety of experiments that we performed in order to evaluate the theoretical results derived in the previous chapter. In section 7.1 we describe the experimental setup we used to perform our experiments. Next, in section 7.2 we show results that were obtained using simple Ising models. Finally, in section 7.3 we show the results obtained using some instances from the UAI 2006 inference competition.

7.1 Experimental Setup

All the experiments focus on the inference problem of computing the probability of evidence. The probability distribution we consider is always normalized or we know the true normalization constant.

The experiments were run on a desktop system containing two Intel® Core™2 Duo processors running at 3 Ghz, 4 GB of main memory and running the 64-bit Ubuntu operating system. The programming environment was Matlab. In order to code the

algorithms, some functions from the Bayes Net Toolbox [18] were used.

To determine the elimination order we used the MIN-FILL heuristic. The subsets for a decomposition were chosen at random. To optimize a decomposition, we used the L^2 optimization method discussed in chapter 5.

7.2 Results on Ising Models

We considered a $15X15$ Ising model with binary nodes and random pairwise potentials. For evidence, we randomly assigned values to about 20% of the nodes. Since we use random splits for decomposition, we ran each algorithm ten times for each value of the threshold and chose the result that had the smallest error value.

First we compare the performance of the corrected MAS bounds derived in chapter 3 versus the L^∞ MAS bounds derived in chapter 4. Figure 7.1 shows the result of running the DynaDecomp algorithm as described in [24] for different values of the threshold parameter. The clear square represents the exact value of $\log P(E)$ while the green circle shows the value of the approximation $\log \tilde{P}(E)$. The black line on the left shows the corrected bounds whereas the red line on the right shows the L^∞ bounds. We see that the L^∞ bounds are tighter than the corrected bounds for all values of the threshold.

Next, we check if using overlapping decompositions with DynaDecomp results in tighter bounds. Figure 7.2 shows the result of running the DynaDecomp algorithm with overlapping decompositions. The bounds in Figure 7.2 are marginally tighter than the bounds in Figure 7.1 which suggests that using overlapping decompositions gives slightly better results than using disjoint decompositions. We also see that the L^∞ bounds are again tighter than the corrected bounds for all values of the threshold.

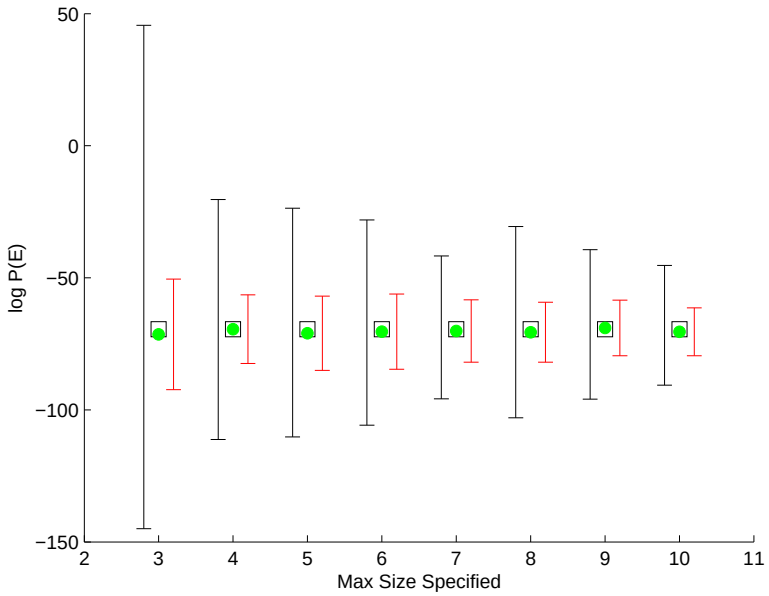

Figure 7.1: Bounds for $\log P(E)$ using DynaDecomp algorithm with no overlapping decompositions allowed

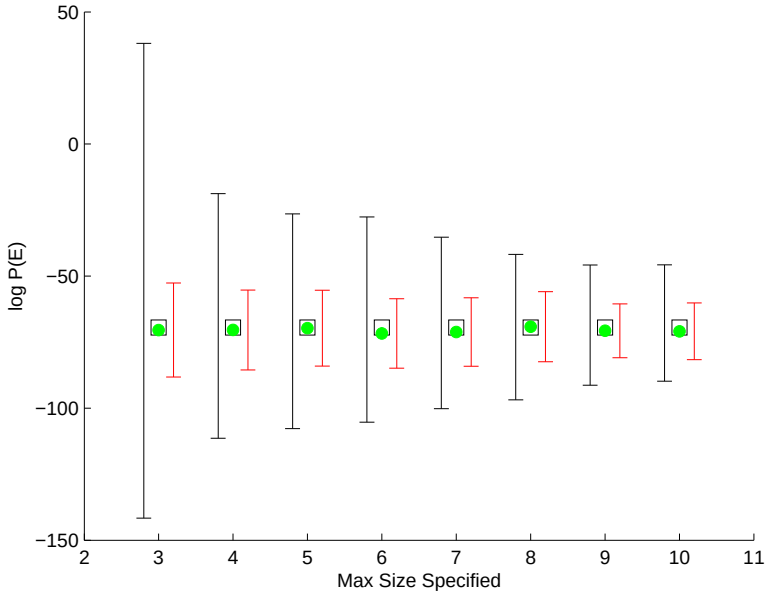

Figure 7.2: Bounds for $\log P(E)$ using DynaDecomp algorithm with overlapping decompositions allowed

Now we look at the performance of the corrected bounds and the L^∞ bounds on the DynaDecompPlus algorithm. Figure 7.3 shows the results of running DynaDecompPlus for different values of the threshold parameter. Since the L^∞ bounds in Figure 7.3 are not very clear, we plotted just the L^∞ bounds in Figure 7.4. Again we see the trend that the L^∞ bounds are tighter than the corrected bounds. In fact, in all the experiments we ran the L^∞ bounds were much tighter than the corrected bounds.

Having compared the performance of the bounds, we now focus on the comparison between DynaDecomp and DynaDecompPlus. We first look at the control the algorithms provide over the time and space complexities.

Figure 7.5 is a plot of the accuracy of the result versus the maximum function size created by DynaDecomp. The plot shows multiple runs of DynaDecomp at each value for the threshold parameter. We can see that even when we set the threshold at $M = 3$, DynaDecomp still generated functions over 7 to 9 variables. This makes it hard to control the time and space complexity of DynaDecomp. For example assuming that we have binary variables and want to limit ourselves to functions not larger than 2^{10} values, we have no easy way of setting the threshold parameter M because as seen in Figure 7.5 there is no simple relation between the threshold value and the maximum function size created by DynaDecomp.

Figure 7.6 is a plot of the accuracy of the result versus the maximum function size created by DynaDecompPlus. It shows that DynaDecompPlus never generates a function that is larger than the threshold specified. Thus, looking at our earlier example, if we wanted to limit ourselves to functions not larger than 2^{10} values, we just have to set the threshold parameter $M = 10$. Comparing Figure 7.6 with Figure 7.5 we can see that DynaDecompPlus provides much greater control over the time and space complexities than DynaDecomp.

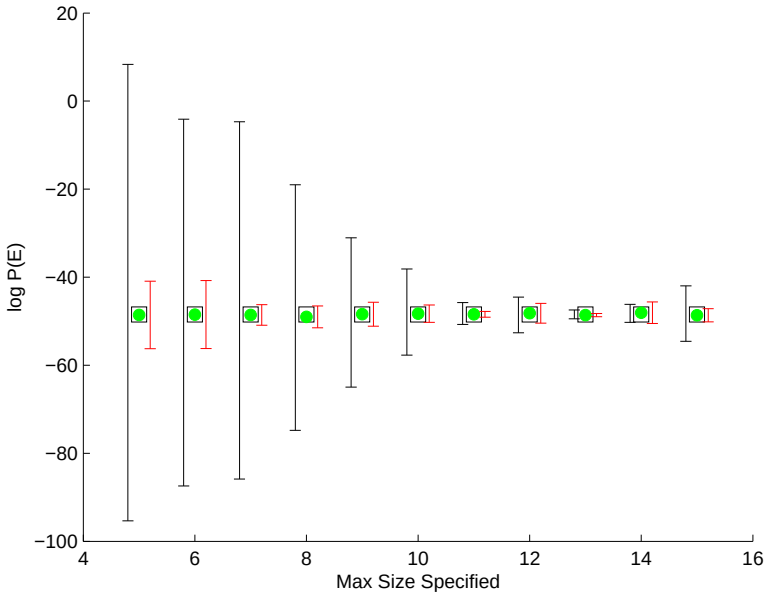

Figure 7.3: Bounds for $\log P(E)$ using DynaDecompPlus algorithm

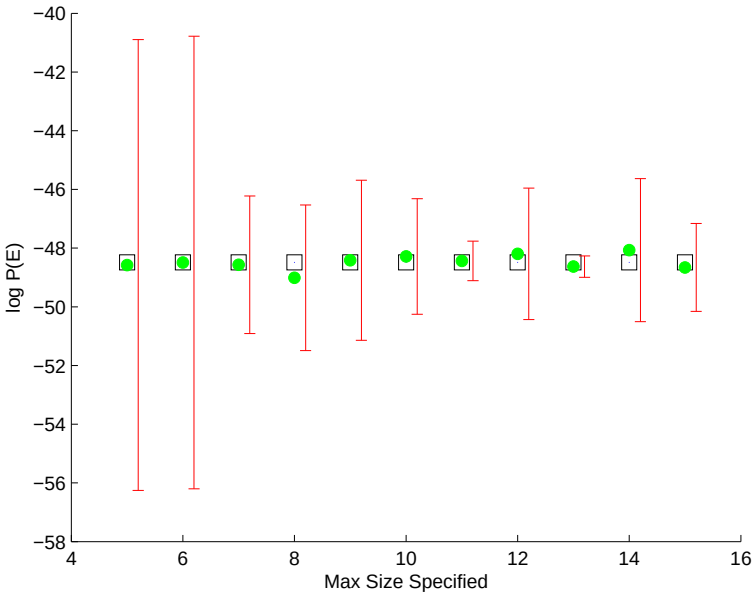

Figure 7.4: L^∞ Bounds for $\log P(E)$ using DynaDecompPlus algorithm

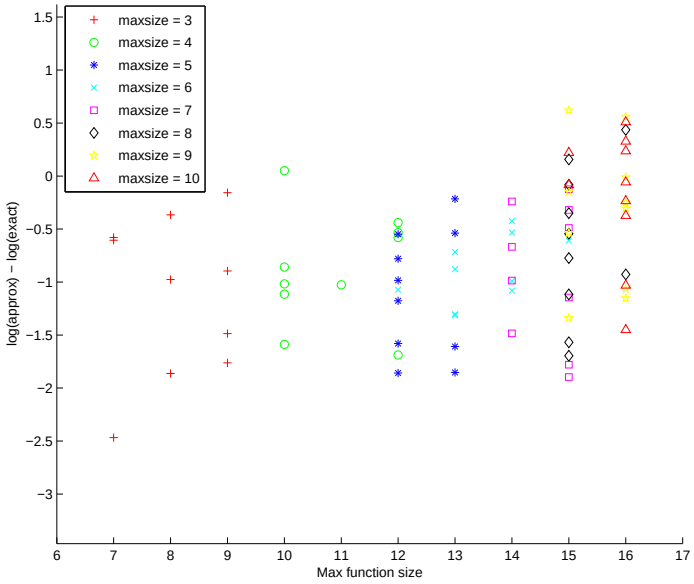

Figure 7.5: Plot of accuracy versus maximum function size for the DynaDecomp algorithm

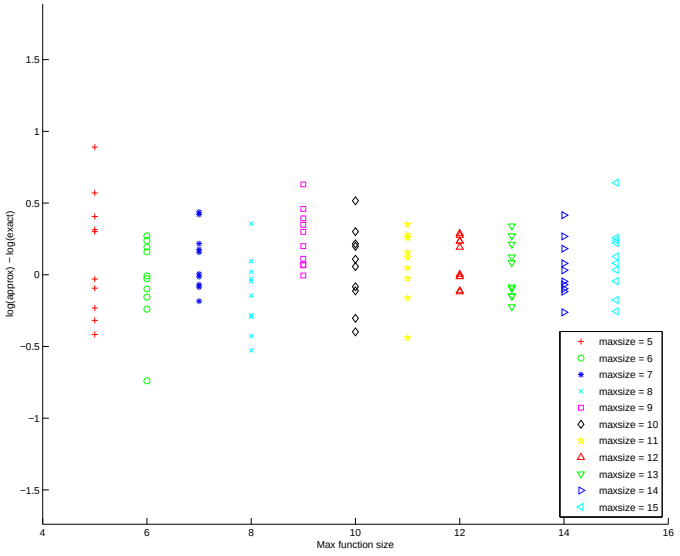

Figure 7.6: Plot of accuracy versus maximum function size for the DynaDecompPlus algorithm

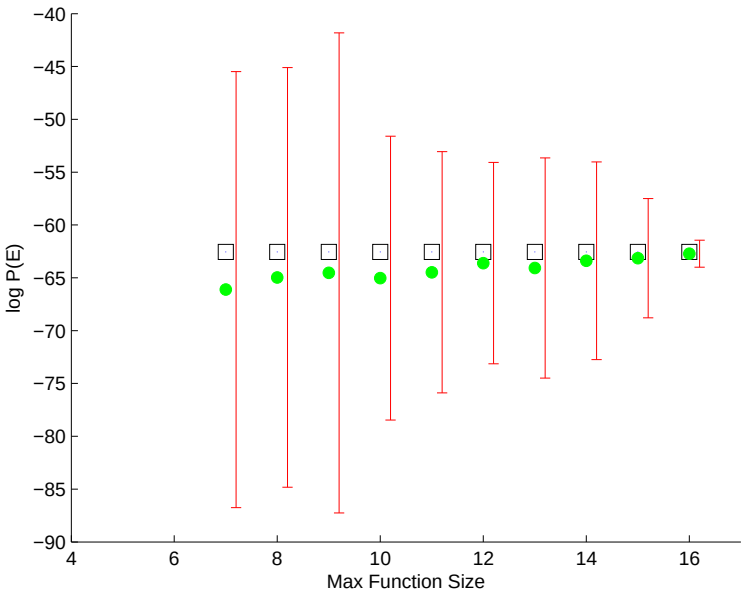

Figure 7.7: L^∞ Bounds for $\log P(E)$ vs Max Function Size for the DynaDecomp algorithm

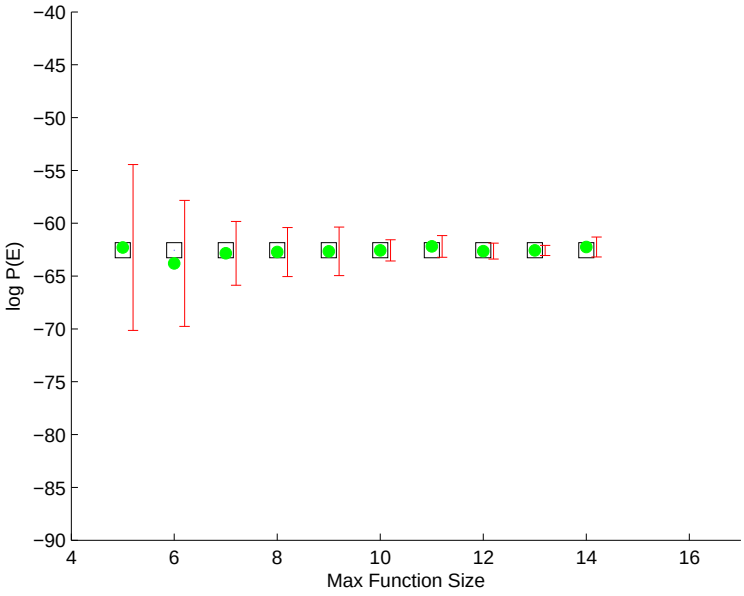

Figure 7.8: L^∞ Bounds for $\log P(E)$ vs Max Function Size for the DynaDecompPlus algorithm

Having shown that DynaDecompPlus provides greater control over the complexity than DyanDecomp, we now look at the performance of DynaDecompPlus and DynaDecomp for the same complexity. Figure 7.7 shows a plot of the L^∞ bounds against the maximum function size that DynaDecomp generated. Figure 7.8 shows a plot of the L^∞ bounds against the maximum function size that DynaDecompPlus generated for the same problem. We can see that for the same complexity DynaDecompPlus is more accurate and generates tighter bounds than DynaDecomp.

7.3 Results on UAI Data

In the previous section we considered only simple Ising models. To compare the performance of the bounds and the two algorithms on more complex models, we ran experiments on seventeen instances from the UAI 2006 inference competition. The instances were taken from the UAI06 Problem Repository available at [1]. We chose instances that had no zero probabilities and to compute the exact solution, we used the **aolibPE** package for exact inference in Bayesian Networks available at [17].

We first tried to run the DynaDecomp algorithm. However, for the heuristic ordering we chose, and a range of threshold values from $M = 3$ to $M = 15$, it failed to solve any of the instances as it kept running out of memory.

Next, we ran the DynaDecompPlus algorithm for different values of the threshold parameter. We provide the results for thresholds of $9, 12$ and 15 in Figure 7.9, Figure 7.10 and Figure 7.11 respectively. We can see that as we increase the size of the threshold the bounds get tighter. We again see that the L^∞ bounds are much tighter than the corrected bounds. In fact the corrected bounds are extremely loose. We do not see the ends of the corrected bounds because we limited the Y-axis to lie between

-200 and 200 for plotting purposes.

Although the L^∞ bounds are tighter than the corrected bounds, we see that in most cases even they are not tight enough for the log probability of evidence. For example, we see that in many cases the upper bound on $\log P(E)$ is greater than zero and the lower bound is about -100. This states that the probability of evidence lies somewhere between e^{-100} and 1 which is not very useful.

From these experiments we can conclude that the corrected MAS bounds from chapter 3 are not useful even for simple Ising models and are quite unusable for any complex models. The L^∞ bounds on the other hand work reasonably well for simple models but as the models get more complicated they too result in not very useful bounds.

With regards to the algorithms, we can see that DynaDecomp does a poor job of controlling the time and space complexities. For complex models it has trouble performing inference even for small values of the threshold parameter. DynaDecompPlus on the other hand can efficiently control the time and space complexities and can perform inference with good accuracy.

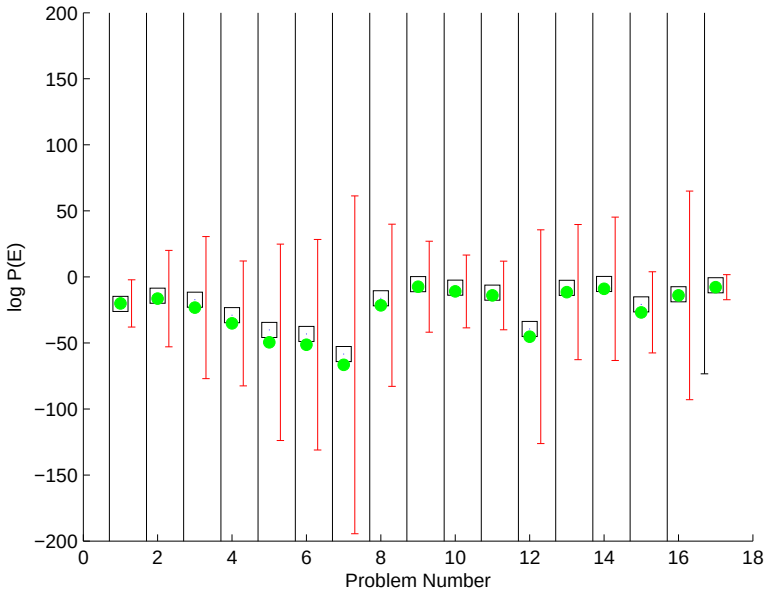

Figure 7.9: Bounds for $\log P(E)$ using DynaDecompPlus algorithm with a maximum function size of 9

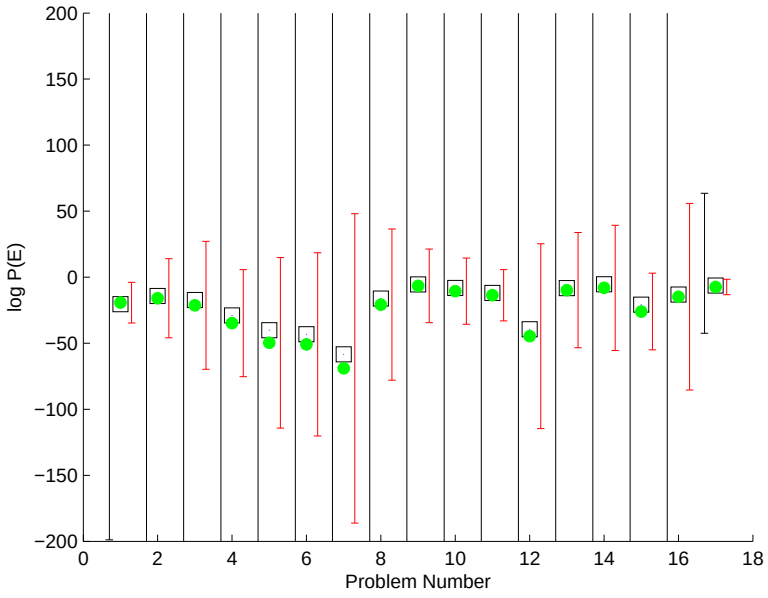

Figure 7.10: Bounds for $\log P(E)$ using DynaDecompPlus algorithm with a maximum function size of 12

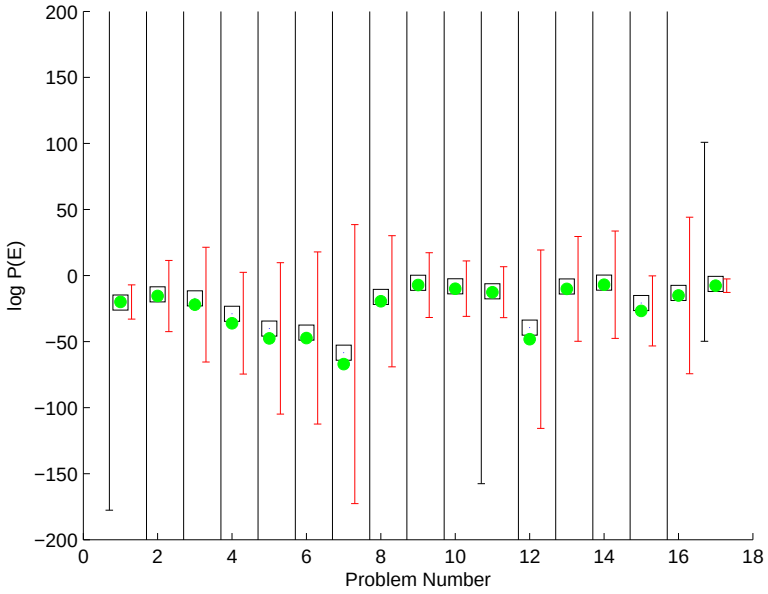

Figure 7.11: Bounds for $\log P(E)$ using DynaDecompPlus algorithm with a maximum function size of 15

Chapter 8

Conclusions

The research presented in this thesis focussed on a specific approximation scheme, called the Multiplicative Approximation Scheme (MAS), that was introduced by Wexler and Meek. It is a general approximation scheme that can be applied to inference algorithms so as to perform bounded approximate inference in graphical models.

8.1 Contributions

We analyzed the multiplicative approximation scheme and proved that in reality the bounds that were claimed by Wexler and Meek turn out to be incorrect when dealing with normalized probability distributions. We then proceeded to derive the correct bounds for these situations.

We also introduced a new way to calculate the error of a local decomposition. Our method used a measure that was based on the L^∞ norm. Using the L^∞ error measure, we derived new bounds on the results of an inference procedure.

We also analyzed the ϵ-decomposition optimization strategy. Wexler and Meek had provided a closed form solution using L_2 optimization on disjoint subsets. We provided a method that allowed us to generate overlapping subsets in our decompositions using the closed form solution for disjoint subsets.

We then proceeded to analyze the algorithm developed by Wexler and Meek called DynaDecomp, which is the application of MAS to bucket elimination. We demonstrated the limitations of DynaDecomp particularly with respect to the lack of control over how much space the algorithm would require. We then modified DynaDecomp and developed our own DynaDecompPlus algorithm that gave us the necessary control over the space the algorithm needed.

Finally we provided experimental evidence that demonstrated how in practice the corrected bounds for MAS turn out to be very loose whereas our L^∞ bounds were much tighter. We also showed that how using overlapped decompositions resulted in tighter bounds than using disjoint decompositions. We also compared the workings of DynaDecomp with DynaDecompPlus and showed how DynaDecompPlus provided much better control over space utilization. We also showed how it could solve problems that DynaDecomp had trouble solving.

8.2 Future Work

With the extensions we have provided, MAS still has room for additional improvements that can be pursued in the future.

The biggest room for improvement is with the local decomposition. For example, say we had a function defined over five variables, $f(X_1, X_2, X_3, X_4, X_5)$, and we would like to decompose this functions into functions over not more than three variables, there

are many different options for us to try. We could create five single variable functions $\tilde{f}_i(X_i)$ or we might choose a disjoint partition $\tilde{f}_1(X_1, X_3, X_4)$ & $\tilde{f}_2(X_2, X_5)$ or we might create an overlapping partition of the form $\tilde{f}_1(X_1, X_2, X_3)$ & $\tilde{f}_2(X_3, X_4, X_5)$ and so on. Currently we make this decision by choosing the subsets at random. Therefore, it would be worthwhile to explore some more sophisticated techniques to choose these subsets.

Ideally we would like to choose the subsets such that the value of the error measure ϵ or δ for the decomposition is the smallest. We could consider a brute force approach of trying all possible decompositions, but with there being an exponential number of decompositions to try, this would be infeasible for all but the most trivial cases.

One possible approach is to use the concept of mutual information, which measures the mutual dependence of the two variables or conditional mutual information, which is the mutual information of two variables conditioned on a third variable. There has been recent work done by Narasimhan and Bilmes [19] and by Chechetka and Guestrin [3] that uses conditional mutual information to learn graphical models with bounded tree-width. Some of the ideas presented in these papers could be used to provide a solution to our problem of finding good decompositions.

Bibliography

[1] UAI 2006 problem repository in ergo file format. `http://graphmod.ics.uci.edu/repos/pe/uaicomp06/`, 2006.

[2] S. Arnborg. Efficient algorithms for combinatorial problems with bounded decomposability - a survey. *BIT*, 25(1):2–23, 1985.

[3] A. Chechetka and C. Guestrin. Efficient principled learning of thin junction trees. In *In Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada, December 2007.

[4] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks (research note). *Artificial Intelligence*, 42(2-3):393–405, 1990.

[5] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, 2001.

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

[7] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 340 Pine Street, Sixth Floor, San Francisco, CA 94104-3205, 2003.

[8] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1987.

[9] R. Dechter and I. Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003.

[10] M. Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Proceedings of the 2nd Annual Conference on Uncertainty in Artificial Intelligence (UAI-86)*, New York, NY, 1986. Elsevier Science.

[11] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6:905–936, May 2005.

[12] 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.

[13] M. Jordan, editor. *Learning in Graphical Models*. The MIT Press, 1998.

[14] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. In *Learning in graphical models*, pages 105–161. MIT Press, Cambridge, MA, USA, 1999.

[15] F. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, Feb 2001.

[16] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[17] R. Mateescu. aolibPE package for exact inference in Bayesian networks. `http://graphmod.ics.uci.edu/group/aolibPE`, 2008.

[18] K. Murphy. Bayes net toolbox for Matlab. `http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html`, 2007.

[19] M. Narasimhan and J. Bilmes. Pac-learning bounded tree-width graphical models. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 410–417, Arlington, Virginia, United States, 2004. AUAI Press.

[20] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, September 1988.

[21] N. Robertson and P. D. Seymour. Graph minors. iii. planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49 – 64, 1984.

[22] R. D. Shachter, B. D'Ambrosio, and B. A. Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 126–131, Boston, Massachusetts, United States, 1990.

[23] G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2(1-4):327–351, 1990.

[24] Y. Wexler and C. Meek. MAS: a multiplicative approximation scheme for probabilistic inference. In *Advances in Neural Information Processing Systems 21*, pages 1761–1768. 2009.

[25] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.