Efficient methods for exact and approximate inference in discrete graphical models

by

Tal El-Hay

Submitted to the School of Computer Science & Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

at the

THE HEBREW UNIVERSITY OF JERUSALEM
JERUSALEM, ISRAEL.

September 2001

Certified by . . . .
Nir Friedman
Doctor, Hebrew University of Jerusalem
Thesis Supervisor

Efficient methods for exact and approximate inference in discrete graphical models

by

Tal El-Hay

Submitted to the School of Computer Science & Engineering
on September 21, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Graphical models allows the representation of large domain probabilistic models by encoding their qualitative properties. In many cases, inference of marginal and conditional probabilities is an intractable problem. Therefore, many methods for exact and approximate inference which suits different kinds of graphical models have been developed. This thesis contains two main parts. The first part deals with global variational approximations and introduces approximating distributions that use expressive graphical models. The second part concerns efficient methods for exact computations of marginal probabilities. These methods are used as a subroutine by variational approximation schemes.

Global variational approximation methods in graphical models allow efficient approximate inference of complex posterior distributions using a simpler model. The choice of the approximating model determines a tradeoff between the complexity of the approximation procedure and the quality of the approximation. In the first part of the thesis, we consider variational approximations based on two classes of models that are richer than standard Bayesian networks, Markov networks or mixture models. As such, these classes allow to find better tradeoffs in the spectrum of approximations. The first class of models are *chain graphs*, which capture distributions that are partially directed. The second class of models are directed graphs (Bayesian networks) with additional latent variables. Both classes allow representation of multi-variable dependencies that cannot be easily represented within a Bayesian network.

The variational methods considered in the first part of the thesis use exact inference on the approximating distribution as a subroutine. Junction tree inference is an efficient method for computing posterior probabilities in graphical models. The resulting tree represents the posterior probability using a tree of clique potentials. From this tree we can directly compute the posterior over sets of variables that are contained in a single clique. However, in variational approximations and in many applications of graphical models, we need to compute the distribution of sets of variables that are dispersed in several cliques. Current methods accomplish this task by a series of clique mergings and marginalizations of unnecessary variables. In the second part of the thesis, we examine how to find a sequence of clique mergings that minimizes the total number of operations in computing such queries from a junction tree. We describe a dynamic programming algorithm that finds the optimal sequence. We analyze the complexity of this procedure, and show when it can be done efficiently. To deal with harder cases, we describe a branch-and-bound method and greedy methods for finding an approximation for the optimal sequence.

Contents

Chapter 1

Introduction

Graphical models [24, 18, 6] are probabilistic models that represent complex probability distributions defined over large sets of random variables. These models are composed of an annotated graph whose vertices represent random variables, and a parameterization of the probability distribution. The edges of the graph encode dependency relations between the random variables. The two most popular types of graphical models are Bayesian networks that contain only directed edges and Markov networks that contain only undirected edges.

The overall graph structure encodes conditional independence assumptions that are put into the model. These assumptions can be represented as a relation among triples of subsets of random variables, that states whether two subsets of random variables are independent given an assignment to the third subset. In graphical models conditional independencies are a consequence of *missing* edges in the graph structure.

The joint probability distribution over the complete set of random variables is represented as a product of local functions that are defined over the product domains of adjacent random variables. For example, in a Bayesian network the local functions are conditional probability distributions, representing the distribution of a random variable given value assignments to his parents in the graph. The random variables in a graphical model might be discrete or continuous. A discrete graphical model is one in which all the random variables are discrete.

In the last decade there has been a growing interest on graphical models in the computational learning community, which is a result of the benefits of using them in applications that involve complex models. The benefits of graphical models arise from the fact that conditional independence assumptions result in absence of edges. This enables a compact representation of the probability distribution, since the local functions comprising it will involve relatively small subsets of random variables. The compact representation has a computational merit, allowing efficient inference of marginal and conditional probabilities. Another benefit of graphical models is the fact that qualitative properties of the distribution can be inferred by looking at the graph structure alone. These properties are the dependency relations between the random variable. They can lead to a deeper understanding of the probabilistic model. Finally, we can benefit from using graphical models when we want to learn complex probability distributions from data. Prior knowledge about conditional independence relations can be incorporated into the model. The reduction in the number of parameters, gained by independence assumptions, prevents over-fitting of the learned model to the data. Also, we can employ learning procedures that adjust the complexity of the model according to the amount of available training data.

A central task in using graphical models is *inference* of marginal and conditional probabilities. This task is used as tool for probabilistic prediction of random variables given a specific state of the system that is modeled. It is also used as a subroutine that calculates *expected sufficient statistics* for learning procedures [22]. Several algorithms were designed for inference in graphical models. These algorithms exploit the compact representation for efficient inference. In general, these algorithms will work faster for distributions with more independence assumptions resulting in graph structures that are more sparse. One of the most popular inference algorithms is the *junction tree algorithm.* This algorithm uses an intermediate structure, a *junction tree*, which is a tree whose nodes represents subsets of variables termed *cliques.* It enables efficient inference for subsets of variables that are contained in a clique. Although current inferences algorithm might be extremely efficient for some graphical models, it is a well known fact that the general problem of inference if NP-hard [4].

The hardness of exact inference methods has led to the design of a wide variety of approximation algorithm. Approximate inference is also NP-hard [7], but using approximate inference algorithms might still be helpful for distributions with specific properties. approximate inference algorithms can be divided into two major classes: stochastic approximation algorithms and deterministic approximation algorithms. Stochastic approximations approximate distributions by sampling. Deterministic approximations include search algorithms, structural approximations and more. Structural approximation algorithms approximate a complex distribution by a simpler distribution that contains a smaller amount of parameters and whose structure renders efficient inference. In graphical models, the simpler distribution can be defined by removing arcs from the graph structure, thus imposing more independencies. It can also be achieved by reducing the numbers of values of the random variables or by simplifying the local functions comprising the distribution.

Variational approximation algorithms are considered as a subclass of structured approximation algorithms. Given an assignment to a subset of random variables in the graphical model, variational algorithms approximate the posterior distribution of the other random variables by another simpler distribution. The approximating distribution is defined using an extra set of parameters. The parameters of the approximating distribution are used to define a lower bound (or an upper bound) on the likelihood of the assignment The values of the parameters of the approximating distribution are found by trying to optimize the bound.

The simplest variational approximation is the *Mean-field* approximation. This approximation was originally suggested by Peterson and Anderson [25] as an approximation scheme for Boltzmann machines but it can be easily generalized to any discrete graphical model. This method approximates the posterior distribution by one on which all the random variables are independent. Peterson and Anderson have shown that for Boltzmann machines they gained accuracies that matches that of stochastic approximation algorithms and runs 10-30 times faster. The mean field approximation is suitable for graphical models with large number of edges and having weak interactions. In order to be able to employ variational methods for other graphical models and in order to improve the accuracy of these methods, Saul and Jordan [29] proposed a structured variational approximation which approximates the posterior by a distribution composed of independent substructures of random variables. The structured approximation for discrete graphical models was generalized and formulated in a manner that enables the approximation of the posterior by any other distribution with a different graphical structure [2, 30]. Variants of Mean field and structured approximations were also formulated for graphical models with continuous random variables. Another direction for improving the Mean field approximation was taken by Jaakkola and Jordan [17]. They proposed to use a mixture of mean field approximation in order to be able to approximate multi-modal posteriors.

Both structured variational approximation and the mixture approximation methods allow for a trade-off between accuracy and complexity. In the structured approximation more accuracy is gained by adding structure, while in the mixture approximation we can increase the number of mixture components.

In this thesis we will suggest ways to generalize and improve the two methods in order to achieve greater accuracy given the available computational resources. The methods introduced are intended to enhance the range of approximating distributions and increase the ability to trade-off accuracy for complexity.

We start by considering extensions of structured approximations. Current structured approximations can use Bayesian networks *or* Markov networks as an approximating distribution. These two classes of models have different expressive power in terms of the dependency relations encoded by them. We provide uniform treatment of both classes by examining a new class of models. The new class of models is more expressive than Bayesian and Markov networks, and includes each one of them as a special class.

We then consider how to add *extra* hidden variables to the approximating model. This method generalizes both the structured approximation and the mixture model approximation. It enables us to control the complexity of the approximating model both through the structure and through the number of values of the hidden variables. The extra hidden variables will also enable us to maintain the dependency between different variables but control the level of dependency, thus keeping the dependencies in a compressed manner . A straight-forward insertion of extra hidden variables to the variational approximation results in an intractable optimization problem, we need combine additional approximation steps. We present a natural generalization of methods suggested by Jaakkola and Jordan [17] for mixtures of mean field models.

The variational approximations we will present use exact inference as a subroutine because they require computation of marginal probabilities of subsets of random variables in the approximating network. Since the procedure needs multiple queries of marginals, we will use the junction tree algorithm which is suitable for the task. In some cases it will need marginals of subsets that are not contained in a single clique. This task can not be addressed by the junction tree algorithm in a straight forward manner. In order to address this problem Xu [31] suggested a *clique merging* algorithm. This algorithm enables computing the joint distribution of a subset of random variables that are dispersed in several cliques by a series of multiplication and marginalization of functions that are defined over domains of subsets of random variables. During the operation of the algorithm intermediate functions are created. The domain size of the intermediate functions depends on the order of operations. Therefore, the complexity of the algorithm depends on the order of operations.

In the second part of this thesis we will address the problem of finding an optimal sequence of operations for the clique merging algorithm. We will suggest a dynamic programming algorithm for the problem. We will analyze its complexity and show when it can be done efficiently. To deal with harder case we will describe a branch-and-bound method, which is a heuristic search method, and greedy methods to approximate the optimal sequence. Finally we will present empirical results on the effect of these methods on the complexity of inference.

The rest of the thesis is organized as follows: Chapter 2 provides background on graphical models and inference. Then it surveys current approximation methods. Finally, it provides an introduction to variational methods for graphical models. Chapter 3 presents the improvement suggested for variational approximations in this thesis. It begins with a presentation of the structured approximation method [2, 30]. Then it presents the augmentation of dependency structure and the addition of extra hidden variables. Chapter 4 deals with methods to find the optimal sequence of clique margining in junction trees. And finally, Chapter 5 provides conclusions and suggestions for further research.

Chapter 2

Background

This chapter provides the background required for reading the rest of the thesis. Section 2.1 reviews the concept of probabilistic graphical models with an emphasis on Bayesian Networks and Markov networks. Section 2.2 provides an outline of a popular exact inference algorithm for Bayesian networks. Section 2.3 provides a brief survey on current approximate inference methods. Finally, Section 2.5 provides an outline of variational approximation methods, which are the main topic of the next chapter.

2.1 Graphical probabilistic models

We start with some notation. We use capital Roman letters X, Y, Z to denote random variables, and bold letters $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ to represent sets of random variables. We denote by $dom(X)$ the domain of X, that is the set values that the random variable X can take. Similarly, we denote by $dom(\boldsymbol{X})$ the domain of the set $\boldsymbol{X}$. This is simply the cross-product of the domains of variables in $\boldsymbol{X}$. Value assignments to random variables X, Y, Z are represented by small letters x, y, z, respectively. Similarly, we use bold letters $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ to represent assignments to $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$, respectively. We use the shorthand $P(\boldsymbol{x})$ for the statement $P(\boldsymbol{X} = \boldsymbol{x})$.

Let $\boldsymbol{X} = \{X_1, ..., X_n\}$ be a finite set of random variables. *Graphical probability models* represent the probability over the domain of $\boldsymbol{X}$ in a manner that encodes conditional independencies among subsets of random variables. Such models are specified by a graph and a set of parameters. The graph describes qualitative properties of the distribution. The vertices of the graph represent random variables and the edges describe dependency relations between them. The set of parameters describes the quantitative features of the underlying distribution.

The two main types of graphical models are *directed* and *undirected* (depending on the type of graph). Directed graphical models, that encode the dependency relationships using a directed acyclic graph (DAG), are termed *Bayesian networks* or *Belief networks.* Undirected models are termed *Markov network* or *Markov random fields.* The difference between the two types is in the manner that graphical structure implies dependency relationships. This point will be clarified in the following sections.

An important feature of graphical is the fact that conditional independence assumptions simplify the model, allowing compact representation and efficient *inference.*

Given a model, we often want to compute the probability of events in the probability space. Inference is the task of answering queries about the probability distribution. Given some *evidence*, which is an assignment $\boldsymbol{e}$ to a subset $\boldsymbol{E} \subset \boldsymbol{X}$, there are two related tasks considered as inference. One is to deduce the probability of the evidence $P(\boldsymbol{e})$. The other is to compute the conditional distribution of another subset $\boldsymbol{U} \subset \boldsymbol{X}$ given the evidence, $P(\boldsymbol{U} \mid \boldsymbol{e})$. Computation of $P(\boldsymbol{e})$ requires marginalization of the distribution from $\boldsymbol{X}$ to $\boldsymbol{E}$. This means that we have to sum or integrate probabilities over all the possible configurations of $\boldsymbol{X} \setminus \boldsymbol{E}$. The hardness of inference follows from the fact that the size of the configuration space grows exponentially in the number of the variables that are marginalized out. In many graphical models the conditional independencies can be exploited to make inference tractable. A comprehensive discussion of graphical models and inference can be found in [24, 18, 6].

2.1.1 Bayesian networks

As mentioned above, the class of graphical models encoding conditional dependencies using a directed acyclic graph is known as Bayesian networks. Applications of Bayesian networks cover areas such as speech-recognition, image understanding, computational biology, medical diagnosis and more. We start with a formal definition of a Bayesian network,

Definition 2.1.1 *A Bayesian network for a random variable set* $\boldsymbol{X} = \{X_1, ..., X_n\}$ *is a pair* $B = \langle G, \Theta \rangle$. *The first component* G, *is a directed acyclic graph whose vertices correspond to the random variables* $X_1, ..., X_n$, *and whose edges represent direct dependencies between the variables. The second component of the pair, namely* Θ, *represents the set of parameters that quantifies the distribution. The set of parameters defines local conditional probability distribution functions for every random variable* X_i *given its parents in the graph,* $P_B(X_i \mid \mathbf{Pa}_i)$, *where* $\mathbf{Pa}_i$ *denotes the set of parents of* X_i. *A Bayesian network defines the joint probability over* $\boldsymbol{X}$ *by,*

$$P_B(\boldsymbol{x}) = \prod_{i=1}^{n} P_B(x_i \mid \mathbf{pa}_i)$$

A Bayesian network represents a probability distribution over a set of random variables $\boldsymbol{X}$ using local conditional probability distributions. Note that the local distributions can be represented using tabular or parametric functions. This representation allows us to infer global conditional independence relations from the structure of the graph G alone, without referring the parameters,

Theorem 2.1.1 *Let* B *be a Bayesian network over a set of variables* $\boldsymbol{X}$. *Then for every* X_i *in* $\boldsymbol{X}$, X_i *is independent of its non-descendants in* G *given its parents in* P_B.

Thus, structured probability representation implies conditional independence relations. In fact, the following theorem states that any distribution complying with the conditional independence assumptions implied by a graph structure, as stated in Theorem 2.1.1, can be represented in a compact manner:

Theorem 2.1.2 *Let* G *be a DAG whose vertices correspond to random variables from the set* $\boldsymbol{X}$. *Let* P *be a joint distribution over* X *complying with the conditional independence*

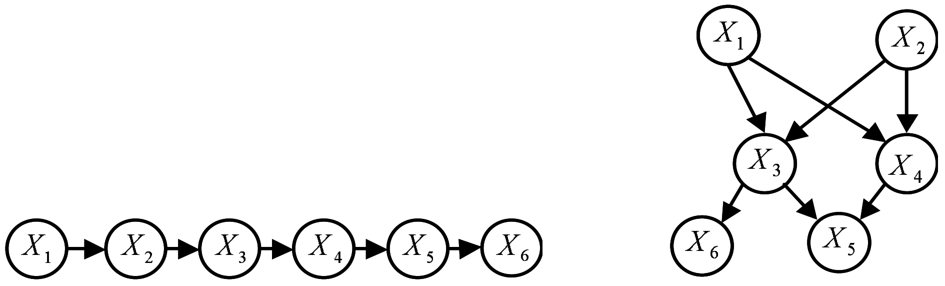


Figure 2-1: Two Bayesian networks defined over 6 random variables. The network in (a) is also known as the Markov chain model.

assumptions induced by the graph: every X_i is independent of its non-descendants given its parents in the graph. Then P can be represented as,

$$P(\boldsymbol{x}) = \prod_i P(x_i \mid \mathbf{pa}_i)$$

That is, P can be represented by a Bayesian network whose graph component is G.

A simple example for a Bayesian network is the *Markov-chain* model. A Markov-chain is a probability distribution assuming the following *Markov property*,

$$P(x_i \mid x_{i-1}...x_1) = P(x_i \mid x_{i-1}) \tag{2.1}$$

that is, the probability of X_i being in state x_i is independent of the states of $X_1, ..., X_{i-2}$ given the state of X_{i-1}. A graphical representation of a small Markov chain is shown in Figure 2-1(a). In this example the probability distribution is defined as

$$P(\boldsymbol{x}) = P(x_1) \prod_{i=2}^{6} P(x_i \mid x_{i-1}) \tag{2.2}$$

A more complex Bayesian network is shown in Figure 2-1(b). The distribution defined by this network is

$$P(\boldsymbol{x}) \quad = \quad P(x_1)P(x_2)P(x_3 \mid x_1, x_2)P(x_4 \mid x_1, x_2)P(x_5 \mid x_3, x_4)P(x_6 \mid x_3)$$

In addition to the conditional independency relations that can be inferred using Theorem 2.1.1. The structure of a Bayesian network induces more conditional independence relations. Such relations can be deduced by inspection of the graph using a property called *d-separation*:

Definition 2.1.2 *Let G be a DAG. A* v-structure *in G is a substructure in the graph, involving 3 nodes X Y and Z, where Z is a child of both X and Y. We denote such a v-structure as, $X \to Z \leftarrow Y$. Let $X_1 - ... - X_n$ be an undirected path in G and $\boldsymbol{E}$ be a subset of nodes in G. The path $X_1 - ... - X_n$ is* active *given evidence $\boldsymbol{E}$ if,*

- *Whenever there is a v-structure $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$, then X_i or one of its descendants is in $\boldsymbol{E}$.*

- *No other node along the path is in $\boldsymbol{E}$.*

Let X and Y be two nodes in G. We say that X and Y are d-separated given $\boldsymbol{E}$, denoted $dsep_G(X; Y \mid \boldsymbol{E})$, if there is no active path between X and Y given $\boldsymbol{E}$.

Analyzing the dependency structure of a Bayesian network, we can exploit the notion of d-separation using the following theorem,

Theorem 2.1.3 *If $B = \langle G, \theta \rangle$ and $dsep_G(X; Y \mid \boldsymbol{E})$, then X and Y are independent given $\boldsymbol{E}$ in the distribution P_B.*

We will illustrate the usage of this theorem using the examples in Figure 2-1. In the Markov chain we can see that every variable separates its ancestors from its descendants. For the Bayesian network in the right side of figure 2-1 we can use Theorem 2.1.3 to deduce more intricate dependency relations. For example we can conclude that given x_3, X_6 is independent of the other variables in the network. We can also see that X_1 and X_2 are marginally independent, i.e. $P(X_1, X_2) = P(X_1) \cdot P(X_2)$, however they become dependent given any other variable.

2.1.2 Markov networks

Markov networks are annotated undirected graphs whose nodes represent random variables from a set $\boldsymbol{X}$. As in Bayesian networks the graph structure is used to represent the dependency relations in a probability distribution over $\boldsymbol{X}$, but the connection between the graphical representation and the probability structure differs from Bayesian networks.

The distributions that can be represented using a Markov network are composed of real functions over the domains of *maximal cliques* in the undirected graph. Cliques are sub-graphs in which every node is connected to every other node. A maximal clique is one that is not a subset of another clique. The functions comprising these distributions, termed *factors*, are not probability distributions on their own and are defined as,

Definition 2.1.3 *Let $\boldsymbol{Y}$ be a set of random variables. We define a* factor *over $\boldsymbol{Y}$ to be a function from $dom(\boldsymbol{Y})$ to the set of non-negative real numbers.*

Formally a Markov network is defined as follows:

Definition 2.1.4 *A Markov network for a random variable set $\boldsymbol{X} = \{X_1, ..., X_n\}$ is a pair $M = \langle H, \Phi \rangle$. The first component H, is an undirected graph whose vertices correspond to the random variables $X_1, ..., X_n$. The second component $\Phi = \phi_1, ..., \phi_k$ is a collection of factors that are defined over the maximal cliques in G: $\boldsymbol{C_1}, ..., \boldsymbol{C_k}$. A Markov network defines the joint probability of $\boldsymbol{X}$ by,*

$$P(\boldsymbol{x}) = \frac{1}{Z_P} \prod_{i=1}^{k} \phi_i(\boldsymbol{c_i})$$

where Z_P is a normalizing constant. The factors ϕ_i are called clique potentials.

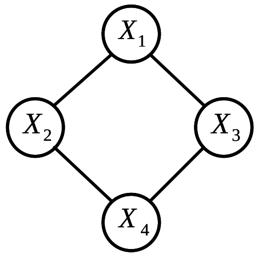


Figure 2-2: A simple Markov network. A distribution that factorizes over this network can be expressed as a product of functions of the maximal cliques. In this particular example the maximal cliques are pairs of connected nodes.

Figure 2-2 shows a simple Markov network representing a distribution that can be factored as,

$$P(\boldsymbol{x}) = \frac{1}{Z_P}\phi_1(x1, x2)\phi_2(x1, x3)\phi_3(x2, x4)\phi_4(x3, x4)$$

In an analogous way to Bayesian networks, we will present the connection between the graphical structure and the conditional independencies in the Markov network distribution through the definition of *separation*,

Definition 2.1.5 *Let $M = \langle H, \Phi \rangle$ be a Markov network. Let $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ be disjoint subsets of nodes in the graph. We say that $\boldsymbol{Z}$* separates *$\boldsymbol{X}$ and $\boldsymbol{Y}$, denoted $sep_H(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z})$, if every path between a node in $\boldsymbol{X}$ and a node in $\boldsymbol{Y}$ must pass through a node in $\boldsymbol{Z}$.*

Finally, the following theorem instructs us how to infer conditional independencies,

Theorem 2.1.4 *Let $M = \langle H, \Phi \rangle$ be a Markov network over $\boldsymbol{X} = X_1, ..., X_n$. Suppose the probability distribution P, defined by M, is positive, i.e. for all $\boldsymbol{x} \in dom(\boldsymbol{X})$, $P(\boldsymbol{x}) > 0$. Then the following holds: For every disjoint subsets $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$, if $sep_H(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z})$ then $\boldsymbol{X}$ is independent of $\boldsymbol{Y}$ given $\boldsymbol{Z}$.*

Returning to Figure 2-2, we can infer that X_1 and X_4 are independent given X_2 and X_3 and vice versa.

To conclude this section we will specify two points of comparison between Markov networks and Bayesian networks. First, both Markov networks and Bayesian networks represent probability distributions as a product of factors defined over subsets of random variables, $\boldsymbol{D}_1, ...\boldsymbol{D}_k$

$$P(\boldsymbol{x}) = \frac{1}{Z_P}\prod_{i=1}^{k}\phi_i(\boldsymbol{d}_i) \tag{2.3}$$

In Markov networks the subsets $\{\boldsymbol{D}_i\}$ are the maximal cliques of the undirected graph and Z_P is a normalization constant also known as *the partition function.* In Bayesian networks the subsets $\{\boldsymbol{D}_i\}$ are the *families* in the directed graph, defined as $D_i = \{X_i, \mathbf{Pa}_i\}$, and $Z_P = 1$. The factors in Bayesian networks are the conditional probabilities $P(X_i \mid \mathbf{Pa}_i)$ of random variables given their parents in the directed graph.

A second point of comparison is the collection of conditional independence relations that are implied by the two graphical models. To clarify this point, we will introduce the notion of a *dependency model* for a joint distribution P. A dependency model is a function that assigns a truth value to triples of disjoint subsets. For every disjoint subsets $\boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$, the value of the function is *true* if and only if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent given an assignment to $\boldsymbol{Z}$. We denote this relation by $Ind(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z})$. Given a distribution that is represented by a Markov network or by a Bayesian network we can infer some of the conditional independencies but not all of them. If all the dependency relations can be inferred only by inspection of the graph structure, the graph may be regarded as a dependency model for the distribution and it is said to be a *perfect map* for the distribution. Formally a perfect map is defined as follows,

Definition 2.1.6 *An annotated DAG G is a perfect map of a distribution P if for every disjoint subsets $\boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$ the following holds:*

$$Ind(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z}) \Leftrightarrow dsep_G(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z})$$

An annotated undirected graph H is a perfect map of a distribution P if:

$$Ind(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z}) \Leftrightarrow sep_H(\boldsymbol{X};\boldsymbol{Y} \mid \boldsymbol{Z})$$

There are probability distributions whose dependency model can be represented using an undirected graph but not using a DAG and vice versa. As an example for the first case, consider the Markov network of Figure 2-2. There is no Bayesian network that can capture the dependency relations implied by this Markov network. To prove this we note that every pair of random variables in that network are marginally dependent. Therefore, there must be a directed path between every pair of variables. Since we can separate X_1 from X_4 and X_2 from X_3, there can not be edges connecting X_1 to X_4 and connecting X_2 to X_3. If there are edges between all the other pairs then one variable would have two parents and there will be another with non at all. In that case, given these two variables the others will be dependent. Therefore, we can assume without loss of generality that there are edges between the pairs $(X_1, X_2), (X_2, X_4)$ and (X_4, X_3). In this case, X_2 and X_3 are independent given X_1 and X_4. But this is not true in the Markov network. As an example for a dependency model that can be represented with a Bayesian network but not with a Markov network, consider a Bayesian network over the variables X_1, X_2 and X_3 where the first two are the parents of X_3. In that case X_1 and X_2 are marginally independent but become dependent given the value of X_3. No Markov network can capture such relations.

2.2 Exact inference

As mentioned in Section 2.1, the term inference refers to the task of calculating the likelihood of some evidence $\boldsymbol{e}$, $P(\boldsymbol{e})$, as well as the task of calculating the marginal distribution of some subset $\boldsymbol{U} \subset \boldsymbol{X}$, $P(\boldsymbol{U} \mid \boldsymbol{e})$. These tasks are related since,

$$P(\boldsymbol{u} \mid \boldsymbol{e}) = \frac{P(\boldsymbol{u}, \boldsymbol{e})}{P(\boldsymbol{e})}$$

Thus, if we compute $P(u, e)$ for all u, we can normalize to obtain $P(u \mid e)$.

In general, inference is a hard problem. Suppose for example that $\boldsymbol{X}$ is a set of binary valued random variables, and that we do not have any independence assumptions about their joint distribution. Using the previous notations, and denoting $\boldsymbol{W} = \boldsymbol{X} \setminus (\boldsymbol{U} \cup \boldsymbol{E})$, the marginal joint probability of e and u is

$$P(\boldsymbol{u}, \boldsymbol{e}) = \sum_{\boldsymbol{w} \in dom(\boldsymbol{W})} P(\boldsymbol{w}, \boldsymbol{u}, \boldsymbol{e})$$

Hence, in order to compute the marginal, we have to sum over $2^{|\boldsymbol{W}|}$ configurations.

In the following subsections we shall see that we can exploit the structure of graphical models to reduce the complexity of inference. The methods presented there can render inference an efficient task in many cases. The general task of inference, though, is NP-hard even for graphical models. If the Bayesian network is encoded as a DAG and a collection of conditional probability distributions, and assuming the worst case representations of the local distribution having the size of a full table, $\mathcal{O}(|dom(\{X_i\} \cup \mathbf{Pa}_i)|)$, we can state the following theorem [4]:

Theorem 2.2.1 *The following problem is NP-hard:*
Given a Bayesian network B, a variable X in it, and a value x of X, decide whether $P_B(X = x) > 0$

2.2.1 Variable elimination

In many graphical models we can exploit the factored representation in order to perform efficient marginalization. Suppose we have a distribution P that can be factored as

$$P(\boldsymbol{x}) = \frac{1}{Z_P} \prod_{i=1}^{k} \phi_i(\boldsymbol{d}_i) \tag{2.4}$$

where $\boldsymbol{d_i}$ are subsets of $\boldsymbol{X}$.[1] Marginalization will be performed by a series of multiplication and marginalization of factors. Therefore, before we proceed we shall define those operations. We shall use the notation $\boldsymbol{x} \models \boldsymbol{w}$ to imply that $\boldsymbol{w}$ is a projection of $\boldsymbol{x}$ on the set $\boldsymbol{W} \subseteq \boldsymbol{X}$.

Definition 2.2.1 *Suppose $\boldsymbol{V}$ and $\boldsymbol{W}$ are sets of variables. Let ϕ and ψ be factors on $\boldsymbol{V}$ and $\boldsymbol{W}$ respectively. We shall define the following operations on factors,*

Multiplication: *The product $\phi \cdot \psi$ is a factor on $\boldsymbol{U} = \boldsymbol{V} \cup \boldsymbol{W}$ which is defined as,*

$$(\phi \cdot \psi)(\boldsymbol{u}) = \phi(\boldsymbol{v}) \cdot \psi(\boldsymbol{w})$$

where $\boldsymbol{u} \models \boldsymbol{v}$ and $\boldsymbol{u} \models \boldsymbol{w}$.

[1]As we mentioned before, this representation holds for Markov networks as well as Bayesian network. In the later case, the subsets are the families in the graph and $Z_Q = 1$.

```
Variable Elimination Procedure

    Input:
        – A factored distribution $P(\boldsymbol{x}) = \frac{1}{Z_P} \prod_{i=1}^{k} \phi_i(\boldsymbol{d}_i)$.
        – A subset $\boldsymbol{U} \subseteq \boldsymbol{X}$.

    Output: The probability table $P(\boldsymbol{U})$.

    $\boldsymbol{W} \leftarrow \boldsymbol{X} \setminus \boldsymbol{U}$ // $\boldsymbol{W}$ is the set of variables that should be marginalized.
    $\mathcal{F} \leftarrow \{\phi_i\}_{i=1}^{k}$ // $\mathcal{F}$ is the set of intermediate factors.
    While $\boldsymbol{W}$ is not empty do,
        Choose a variable $X_k \in \boldsymbol{W}$, $\boldsymbol{W} \leftarrow \boldsymbol{W} \setminus \{X_k\}$.
        $\mathcal{F}' \leftarrow \{\phi \mid \phi \in \mathcal{F}, X_k \in dom(\phi)\}$
        $\psi \leftarrow \sum_{x_k} \prod_{\phi \in \mathcal{F}'} \phi$
        $\mathcal{F} \leftarrow (\mathcal{F} \setminus \mathcal{F}') \cup \{\psi\}$
    $P(\boldsymbol{U}) \leftarrow \frac{1}{Z_P} \prod_{\phi \in \mathcal{F}} \phi$
```

Figure 2-3: Variable elimination procedure for marginalizing a distribution from $\boldsymbol{X}$ to $\boldsymbol{U}$. On each iteration, one variable from $\boldsymbol{X} \setminus \boldsymbol{U}$ is marginalized out using only the factors containing it.

Marginalization: *Let $\boldsymbol{W} \subseteq \boldsymbol{V}$ then the marginalization of ϕ from $\boldsymbol{V}$ to $\boldsymbol{W}$ denoted by $\psi = \sum\limits_{V \setminus W} \phi$ is a factor on $\boldsymbol{W}$ defined as,*

$$\psi(\boldsymbol{w}) = \sum_{\{\boldsymbol{v} \mid \boldsymbol{v} \models \boldsymbol{w}\}} \phi(\boldsymbol{v})$$

Suppose we want to compute the probability table of a subset $\boldsymbol{U} \subset \boldsymbol{X}$. A naive procedure is to perform exhaustive summation: $P(\boldsymbol{u}) = \sum_{\{\boldsymbol{x} \mid \boldsymbol{x} \models \boldsymbol{u}\}} P(\boldsymbol{x})$. However, the amount of computations for this procedure would be exponential in the size of $\boldsymbol{X}$. The *variable elimination* procedure performs marginalization more efficiently using the factored form of the distribution. This procedure iterates on the variables that are not in $\boldsymbol{U}$. On each iteration, one variable $X_k \notin \boldsymbol{U}$ is marginalized out to obtain a factored distribution which does not contain X_k. This is done by replacing the factors that contain X_k by a single factor which is their product and then marginalizing X_k from this factor. Figure 2-3 summarizes this procedure.

As an example for variable elimination we shall return to the Markov-chain example given in Section 2.1.1. Recall that the joint distribution is

$$P(\boldsymbol{x}) = P(x_1) \prod_{i=2}^{n} P(x_i \mid x_{i-1}) \tag{2.5}$$

Suppose we want to calculate $P(x_n = 1)$, assuming all the variables are binary valued. A naive calculation would require summation of 2^{n-1} terms. Every term requires $n-1$ multiplications, resulting in a total of $(n-1) \cdot 2^{n-1}$ multiplications. Using the variable elimination procedure we can define $\psi_1(x_1) = P(x_1)$. Then, on every iteration we set

$$\psi_i(x_i) = \sum_{x_{i-1}} \psi_{i-1}(x_{i-1}) \cdot P(x_i \mid x_{i-1})$$

Thus, every iteration requires 2 additions and 2 multiplications, and the total number of multiplications is $2 \cdot (n-1)$.

For the general case, the time complexity of the variable elimination algorithm is proportional to the total size of intermediate factors. The space requirements depend on the size of the biggest intermediate clique. This depends on the order of the variables that are eliminated and on the graph structure of the model.

2.2.2 The junction tree algorithm

The *junction tree algorithm* is an inference algorithm for graphical models, that is well suited for answering multiple queries efficiently. Given evidence $\boldsymbol{e}$, this algorithm is capable of computing conditional probabilities of every single variable in $\boldsymbol{X}$, and of some subsets of variables, in time complexity that is similar to that required for a single query. The ability to answer multiple queries is important in its own right, and is also crucial for learning algorithms. As we shall see in the next chapter, this feature is also exploited by variational approximation algorithms.

The junction tree algorithm is a dynamic programming scheme. The algorithm stores intermediate factors and reuses them for different calculations. The factors are associated to a dedicated structure called *junction tree*, which is defined as follows:

Definition 2.2.2 *A* Junction tree *is a tree $\mathcal{T}$ with a set of nodes $\mathcal{I}$ and a set of edges $\mathcal{E}$. Each node $i \in \mathcal{I}$ is associated with a set $\boldsymbol{C}_i \subseteq \boldsymbol{X}$ called a* Clique. *For each edge $(i,j) \in \mathcal{E}$ we associate a* separator *which is the set $\boldsymbol{S}_{(i,j)} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$. A junction tree maintains the* Running intersection property*: For every i and j, $\boldsymbol{C}_i \cap \boldsymbol{C}_j$ is contained in every clique in the path between i and j.*

The *junction tree algorithm* receives as input a model represented by functions as in Equation 2.4 and an *evidence* $\boldsymbol{E} = \boldsymbol{e}$ where $\boldsymbol{E} \subseteq \boldsymbol{X}$. The algorithm returns a junction tree in which every subset $\boldsymbol{D}_m$ of the factored representation is contained in some clique. In addition, every clique $\boldsymbol{C}_i$ and separator $\boldsymbol{S}_{(i,j)}$ are associated with a *potential* which is the conditional probability of $\phi_i = P(\boldsymbol{C}_i \mid \boldsymbol{e})$ and $\phi_{(i,j)} = P(\boldsymbol{S}_{(i,j)} \mid \boldsymbol{e})$, respectively. Figure 2-4 shows junction trees for the Bayesian networks in Figure 2-1.

The main property of junction trees is that they allow one to represent the posterior probability over $\boldsymbol{X}$ compactly

$$P(\boldsymbol{x} \mid \boldsymbol{e}) = \frac{\prod_{i \in \mathcal{I}} P(\boldsymbol{c}_i \mid \boldsymbol{e})}{\prod_{(i,j) \in \mathcal{E}} P(\boldsymbol{s}_{(i,j)} \mid \boldsymbol{e})} = \frac{\prod_{i \in \mathcal{I}} \phi_i(\boldsymbol{c}_i)}{\prod_{(i,j) \in \mathcal{E}} \phi_{(i,j)}(\boldsymbol{s}_{(i,j)})} \tag{2.6}$$

Once we have a junction tree, we can compute the posterior probability for a set of variables.

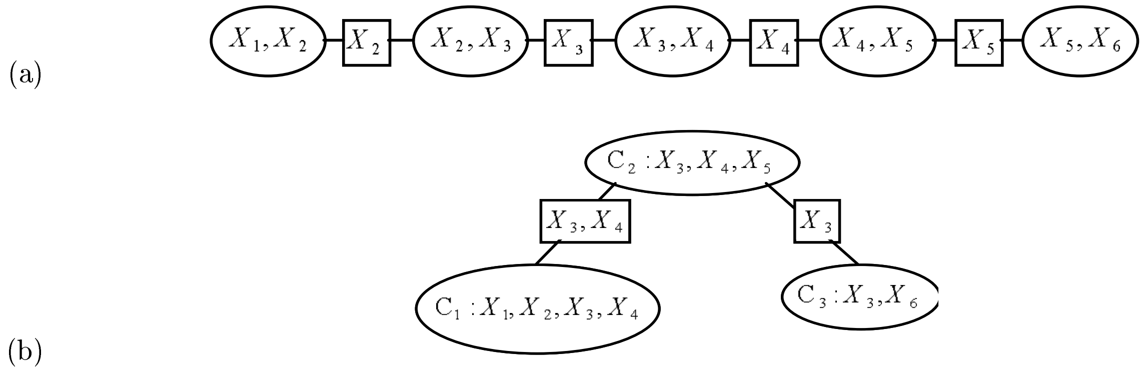


Figure 2-4: Junction trees for the Bayesian networks from Figure 2-1. The ellipses represent cliques, the squares represent separators. On the lower junction tree we also show the clique labels.

If a set $\boldsymbol{U}$ is contained in a clique $\boldsymbol{C}$, we can calculate its posterior probability by marginalization of the clique $\boldsymbol{C}$.

An important feature of the separators in the junction tree is that given a full assignment to a separator, The probability distribution of a subset of variables on one side of the separator in the junction tree is conditionally independent of the distribution of subsets in the other side. We can see for example from the junction tree in Figure 2-4(b) that given X_3 and X_4, $\{X_1, X_2\}$ are independent from $\{X_5, X_6\}$.

The running time of the junction tree algorithm is linear in the sum of the domain sizes of every clique. Hence, it is exponential in the size of the biggest clique. We shall give here two typical examples for Bayesian networks on which exact inference with these methods is intractable. Figure 2-5 shows a two layered Bayesian network with dense connections. In this network the nodes in the second layer might have a large number of parents. The local probability distribution functions for such models are given in a parametric form to allow compact representation. The parametric form does not prevent the exponential complexity of inference. Figure 2-6 presents a situation in which inference is hard even if every node has a limited number of parents. This Bayesian network represents a time dependent process with hidden variable X_t^i and observed variables O_t. Thus, the hidden state might be represented by N hidden variables for each time slice. The size of the cliques created by the variable elimination algorithm or by the junction tree algorithm is at least $N+1$. Therefore, inference complexity is $\mathcal{O}(2^N)$.

The problem of constructing an optimal junction tree, i.e. one that induces minimal amount of computations is NP-hard [28]. Details about construction of junction trees, their usage in inference and proofs of their properties can be found in [24, 18, 6].

2.3 Approximate inference

The theoretical hardness result for exact inference, given in Theorem 2.2.1, has practical consequences in some real life graphical models. This fact leads us naturally to resort to approximation algorithms. Although approximation of a marginal conditional distribution is also NP-hard [7], many approximation algorithms have been developed to enlarge the range of graphical mod-

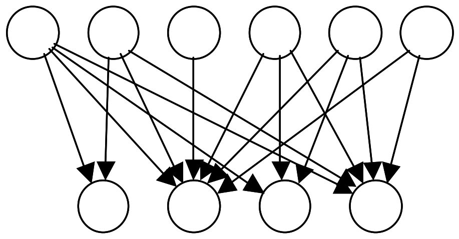


Figure 2-5: A two layered Bayesian network with dense connections. Inference complexity is exponential by the maximal number of parents of the nodes.

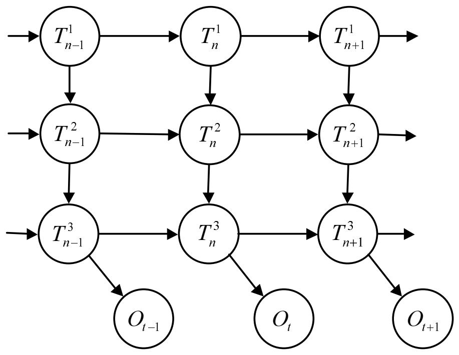


Figure 2-6: A Bayesian network representing a time dependent process with 3 hidden state variables. Inference complexity is exponential by the number of hidden variables in each time slice.

els and inference tasks that are computationally feasible.

2.3.1 An overview of current approximation methods

Since the general problem of approximate inference is NP-hard, a variety of approximation algorithms have been developed to handle probability distributions having specific characteristics. For example, some inference algorithms are suitable for concentrated distributions, some for highly stochastic distributions. The performance of other type of approximation algorithms depend on the dependency structure of the approximation distribution. There are also inference algorithm specialized for dealing with probability models of time dependent processes.

One class of randomized approximation algorithms are *Monte Carlo methods* which are based on sampling the distribution. The hardness of sampling for multi variable distributions has led to the development of *Markov Chain Monte Carlo* methods (*MCMC*) [21] that generate successive samples from a distribution that is guarantied to converge to the true distribution. However the rate of convergence is not known in advance and in practice it could take a significant amount of time to converge. MCMC methods are suitable for highly stochastic distributions. Another randomized algorithm for Bayesian networks whose conditional probabilities are not *extreme*, i.e. are not arbitrarily close to zero, is the *bounded-variance algorithm* [8]. This algorithm gives, with high probability, a good approximation in polynomial time.

A deterministic family of approximation algorithms, which is suitable for peaked distributions, is the *search algorithms* performing a heuristic search for the most likely assignments and approximates the joint distribution using these assignments only. One example is the *bounded conditioning algorithm* [9]. These algorithms are able to provide online bounds on their performance. However they do not perform well if the evidence is not consistent with high mass assignments.

An important class of approximation algorithm are *structural approximations.* These algorithms are deterministic algorithms assessing probabilities by simplifying the probability structure. Simplifying assumptions include:

- Additional conditional independencies (thus, removing edges from the graphical model).
- Removal of irrelevant nodes.
- Reducing the number of values of a node.
- In parametric models, using simpler functions to represent local probability distributions.

These algorithms usually run faster than Monte-Carlo methods. However, their accuracy depends on the structural simplifications. Many of them provide an adjustable tradeoff between accuracy and computational complexity.

Variational approximations have gained a great deal of interest in recent years. This family of algorithms falls into the class of structured approximations, and has the advantages and limitations of this class. In addition they can provide bounds on some probabilities of interest and they fit naturally in learning algorithms. In the next section we shall focus on variational methods.

The wide range of approximation algorithms can not be completely covered in a short overview. We shall mention here only another one interesting approximation algorithm which is the *loopy belief propagation algorithm.* This algorithm updates the posterior probability distributions of the families of random variables by a local message passing scheme, on which every family updates the state of it's neighboring families. For singly connected graphical models the algorithm is equivalent to the junction tree algorithm. Otherwise it is regarded as an approximation algorithm. Recent work has shown interesting relations between this algorithm and variational methods [14].

Before we focus on variational approximations we shall review some information theoretic quantities [5] that will be useful in this section and in the next chapter.

2.4 Basic information theoretic quantities

In order to evaluate approximation schemes we shall use the *KL-divergence* as a distance measure between two distributions.

Definition 2.4.1 *The* Relative entropy *or* Kullback Leibler divergence *between two probability mass functions $P(\boldsymbol{x})$ and $Q(\boldsymbol{x})$ defined over a sample space $dom(\boldsymbol{X})$ is defined as,*

$$\begin{aligned} KL(P||Q) &= \sum_{\boldsymbol{x}} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \\ &= E_P\left[\log \frac{p}{q}\right] \end{aligned}$$

This distance measure is always non-negative and equals zero if and only if $P = Q$, except for points with total measure zero. However, it is not a metric, since it is not symmetric and does not satisfy the triangle inequality.

Two other quantities that we will use are the *entropy* and the *conditional entropy* of a set of random variables. These quantities measure the randomness or uncertainty level of sets of random variables.

Definition 2.4.2 *The* entropy $H(\boldsymbol{X})$ *of a set of discrete random variables* $\boldsymbol{X}$ *is defined by*

$$H(\boldsymbol{X}) = -\sum_{\boldsymbol{x}} P(\boldsymbol{x}) \log P(\boldsymbol{x})$$

Definition 2.4.3 *Suppose* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *are sets of random variables, the* conditional entropy $H(\boldsymbol{Y} \mid \boldsymbol{X})$ *is defined as*

$$H(\boldsymbol{Y} \mid \boldsymbol{X}) = \sum_{\boldsymbol{x}} H(\boldsymbol{Y} \mid \boldsymbol{X} = \boldsymbol{x}) P(\boldsymbol{x})$$

Finally, the *mutual information* of two sets of random variables measures the dependency level between the two sets:

Definition 2.4.4 *The* mutual information $I(\boldsymbol{X}; \boldsymbol{Y})$ *of the random variable sets* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *is the KL-distance between the joint distribution and the product distribution, i.e.*

$$\begin{aligned} I(\boldsymbol{X}; \boldsymbol{Y}) &= KL(P(\boldsymbol{X})P(\boldsymbol{Y}) || P(\boldsymbol{X}, \boldsymbol{Y})) \\ &= \sum_{\boldsymbol{x},\boldsymbol{y}} P(\boldsymbol{x}, \boldsymbol{y}) \log \frac{p(\boldsymbol{x}, \boldsymbol{y})}{p(\boldsymbol{x})p(\boldsymbol{y})} \end{aligned}$$

Note that since the KL-divergence is always non negative, the mutual information is also non negative. Using the definitions above we can express the mutual information also as

$$I(\boldsymbol{X}; \boldsymbol{Y}) = H(\boldsymbol{X}) - H(\boldsymbol{X} \mid \boldsymbol{Y}) = H(\boldsymbol{Y}) - H(\boldsymbol{Y} \mid \boldsymbol{X}) \tag{2.7}$$

Thus, this quantity measures the expected reduction of uncertainty in one set given that we know the state of another set, or in other words, the amount of information that one set contains about the other.

2.5 Variational approximations

The term variational methods stems from the techniques of the *calculus of variations*, which were used for finding the extremum of path integrals depending on an unknown function and its derivatives. Since the introduction of this technique, the term has broadened to include various techniques in different applications, among them approximation methods [27].

The guiding principle of using variational approximation methods, is to simplify the target function by decoupling the degrees of freedom of its components [19]. This is done by defining an approximating function with additional parameters known as *variational parameters*. The evaluation problem is then transformed into an optimization problem, that minimizes the error of the approximation. In general, the optimization problem has no analytic solution and is solved by iterating a set of fixed point equations. These points will be clarified using the mean field example in Section 2.5.2.

In this section we will give a brief introduction to variational methods in graphical probabilistic models. More extensive tutorials are given by Jordan et al. [19] and by Jaakkola [16].

2.5.1 Approximating distributions by KL-divergence minimization

Let P be a distribution over the set of random variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. We denote by $\boldsymbol{O} \subset \boldsymbol{X}$ the subset of observed variables and by $\boldsymbol{T} = \boldsymbol{X} \setminus \boldsymbol{O}$ the set of hidden variables. Our task is to approximate the distribution $P(\boldsymbol{T} \mid \boldsymbol{o})$ with another *simpler* distribution $Q(\boldsymbol{T}; \Theta)$, where Θ is the set of parameters for the approximating network Q.

Given a parametric family of the approximating network, we wish to find the set of parameters Θ that minimizes the distance between $Q(\boldsymbol{T}; \Theta)$ and the posterior distribution $P(\boldsymbol{T} \mid \boldsymbol{o})$. As a distance measure we shall use the KL-divergence.

The KL divergence between $Q(\boldsymbol{T}; \Theta)$ and the posterior distribution $P(\boldsymbol{T} \mid \boldsymbol{o})$ is,

$$KL(Q(\boldsymbol{t})||P(\boldsymbol{t} \mid \boldsymbol{o})) = E_{Q(\boldsymbol{T})}\left[\log \frac{Q(\boldsymbol{T})}{P(\boldsymbol{T} \mid \boldsymbol{o})}\right] \tag{2.8}$$

Finding the parameters for Q will allow us to compute a lower bound for $\log P(\boldsymbol{o})$. since,

$$KL(Q(\boldsymbol{t})||P(\boldsymbol{t} \mid \boldsymbol{o})) \quad = \quad E_{Q(\boldsymbol{T})}\left[\log Q(\boldsymbol{T})\right] - E_{Q(\boldsymbol{T})}\left[\log P(\boldsymbol{T}, \boldsymbol{o})\right] + \log P(\boldsymbol{o})$$

Defining

$$\mathcal{F}[Q] = E_{Q(\boldsymbol{T})}\left[\log P(\boldsymbol{T}, \boldsymbol{o})\right] - E_{Q(\boldsymbol{T})}\left[\log Q(\boldsymbol{T})\right] \tag{2.9}$$

We get that

$$\log P(\boldsymbol{o}) = \mathcal{F}[Q] + KL(Q||P) \geq \mathcal{F}[Q] \tag{2.10}$$

The inequality is true because the relative entropy is non-negative. Hence, $\mathcal{F}[Q]$ is a lower bound on the log-likelihood. The difference between $\mathcal{F}[Q]$ and the true log-likelihood is the KL-divergence. Therefore, minimizing the KL-divergence is equivalent to finding the tightest lower bound. As we shall see later, in many cases computing the lower bound might be easier than computing the exact log-likelihood.

2.5.2 The mean field approximation in discrete graphical models

One of the simplest methods of variational approximation is the mean field approximation. The application of this method to graphical models was first introduced by Peterson and Anderson [25] as an approximation scheme for Boltzmann machines. This method approximates the posterior distribution by another distribution in which all the random variables are independent. Here we shall derive mean field equations for discrete graphical probabilistic models from the KL-minimization principle.

Suppose the domain of $\boldsymbol{X}$ is discrete and P is modeled in the following factored way:

$$P(\boldsymbol{x}) = \frac{1}{Z_P}\prod_{i=1}^{k}\phi_i(\boldsymbol{d}_i)$$

where $\boldsymbol{D}_i$ are subsets of $\boldsymbol{X}$. We will approximate $P(\boldsymbol{T} \mid \boldsymbol{o})$ by a completely factorized distribution. Thus, Q can be represented as,

$$Q(\boldsymbol{t}) = \prod_j Q(x_j)$$

Given the parameters and the evidence of P, the lower bound, $\mathcal{F}$ is a function of the parameters of Q, $\{Q(x_j)\}$. We wish to find the maximum of $\mathcal{F}$, satisfying the set of constraints:

$$\sum_{x_j} Q(x_j) = 1$$

for every $X_j \in \boldsymbol{T}$. We will use Lagrange multipliers and define the following Lagrangian:

$$\mathcal{J}(Q) = \mathcal{F}[Q] - \sum_j \lambda_j(\sum_{x_j} Q(x_j) - 1)$$

Where $\mathcal{F}[Q]$ is defined in Equation 2.9. In our case it equals:

$$\begin{aligned}
\mathcal{F}[Q] &= \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log P(\boldsymbol{t},\boldsymbol{o}) - \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log Q(\boldsymbol{t}) \\
&= \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log\frac{1}{Z_P}\prod_{i=1}^{k}\phi_i(\boldsymbol{d}_i) + \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log\prod_j Q(x_j) \\
&= \sum_{i=1}^{k}\sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log\phi_i(\boldsymbol{d}_i) - \log Z_P - \sum_j\sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log Q(x_j)
\end{aligned}$$

The optimum for our optimization problem is found by equating the partial derivatives of the Lagrangian to zero. In order to save space we shall denote $Q(x_j)$ by θ_{x_j}. The partial derivative $\frac{\partial Q(\boldsymbol{t})}{\partial\theta_{x_j^\circ}}$ equals $Q(\boldsymbol{t} \mid x_j^\circ)$. Therefore

$$\begin{aligned}
\frac{\partial\mathcal{J}}{\partial\theta_{x_j^\circ}} &= \sum_i\sum_{\boldsymbol{x}\models\boldsymbol{o}} Q(\boldsymbol{t} \mid x_j^\circ)\log\phi_i(\boldsymbol{d}_i) - \sum_{j'\neq j}\sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid x_j^\circ)\log\theta_{x_{j'}} - \sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid x_j^\circ)(\log\theta_{x_j^\circ} + 1) - \lambda_j \\
&= \sum_i E_{Q(\cdot|x_j^\circ)}\left[\log\phi(\boldsymbol{D}_i,\boldsymbol{o})\right] - \sum_{j'\neq j} E_{Q(\cdot|x_j^\circ)}\left[\log\theta_{x_{j'}}\right] + \log\theta_{x_j^\circ} + 1 - \lambda_j
\end{aligned}$$

where $\phi(\boldsymbol{D}_i,\boldsymbol{o})$ is a random variable whose value is $\phi(\boldsymbol{d}_i)$ if $\boldsymbol{d}_i$ is consistent with $\boldsymbol{o}$. Otherwise it is 0. Note that if $\boldsymbol{U}$ is a subset of $\boldsymbol{X}$ and $X_j \notin \boldsymbol{U}$ then for x_j° and x_j' and for any function of

$dom(\boldsymbol{U})$ the following holds:

$$E_{Q(\cdot|x_j^\circ)}\left[f(\boldsymbol{U})\right] = E_{Q(\cdot|x'_j)}\left[f(\boldsymbol{U})\right] \tag{2.11}$$

That is, such terms are constant with respect to any assignment to X_j. Denoting the set of indices of the subsets $\boldsymbol{D}_i$ that include X_j by $\mathcal{I}_j = \{i \mid X_j \in \boldsymbol{D}_i\}$ and denoting the sum of all the terms that are constant with respect to x_j by λ'_j, we can rewrite the partial derivative as

$$\frac{\partial \mathcal{J}}{\partial \theta_{x_j^\circ}} = \sum_{i\in\mathcal{I}_j} E_{Q(\cdot|x_j^\circ)}\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] + \log \theta_{x_j^\circ} + \lambda'_j \tag{2.12}$$

Equating the derivative to zero and rearranging we get that the condition for an optimal point is,

$$Q(x_j^\circ) = C_j \cdot e^{\sum_{i\in\mathcal{I}_j} E_{Q(\cdot|x_j^\circ)}[\log \phi(\boldsymbol{D}_i,\boldsymbol{o})]} \tag{2.13}$$

where C_j is a normalization constant.

Note that the right hand side does not depend on $Q(x_j^\circ)$. It can be shown that F is a concave function of the parameters of X_i when all the other parameters are fixed. Therefore, sequential updates of the parameters of one variable at a time increases the bound $\mathcal{F}$ on the log-likelihood.

The update procedure defined in equation 2.13 finds a *local* maximum. The complexity of this scheme depends on the size of the subsets $\boldsymbol{D}_j$. This is true because we need to compute the conditional probabilities of the form

$$Q(\boldsymbol{d}_i \mid x_j^\circ) = \prod_{X_{j'}\in\boldsymbol{D}_i\setminus\{X_j\}} Q(x_{j'})$$

where $\boldsymbol{d}_i \models x_j^\circ$ and $\boldsymbol{d}_i \models x_{j'}$ for all j'.

We shall illustrate an application of the mean field equations on Boltzmann machines. Boltzmann machines are undirected graphical models over $\{X_1, ...X_n\}$ where $x_i \in \{0,1\}$. The joint probability distribution function of Boltzmann machines is given as a product of pairwise potential functions:

$$\begin{aligned} P(\boldsymbol{x}) &= \frac{1}{Z_P} e^{\sum_{(i,j)\in\mathcal{E}} \psi_{ij}x_ix_j + \sum_i \psi_{i0}x_i} && (2.14)\\ &= \frac{1}{Z_P} \prod_{(i,j)\in\mathcal{E}} \phi_{ij}(x_i,x_j) \prod_i \phi_i(x_i) && (2.15)\end{aligned}$$

where $\mathcal{E}$ is the set of edges in the undirected graph, $\phi_{ij}(x_i,x_j) = e^{\psi_{ij}x_ix_j}$ and $\phi_i(x_i) = e^{\psi_{i0}x_i}$. Given evidence $\boldsymbol{o}$ we would like to approximate the marginal probabilities of the hidden variables $X_j \in \boldsymbol{T}$ using a completely factorized model. Denoting $Q(X_j = 1) = \mu_j$ and $Q(X_j = 0) = 1-\mu_j$ and using Equation 2.13 we get the following equations:

$$\begin{aligned} \mu_j &= C_j \cdot e^{\sum_{i\in\mathcal{I}'_j} \psi_{ij}\mu_i + \sum_{i\in\mathcal{I}''_j} \psi_{ij} + \psi_{j0}} \\ 1-\mu_j &= C_j \cdot e^0 = C_j \end{aligned}$$

Where $\mathcal{I}'_j$ is the set of indices of the hidden variables that are neighbors of X_j. $\mathcal{I}''_j$ is the set of indices of variables whose observed value is 1 and are neighbors of X_j. Normalizing the parameters we get the update equations for μ_j:

$$\mu_j = \sigma\left(\sum_{i\in\mathcal{I}'_j}\psi_{ij}\mu_i + \sum_{i\in\mathcal{I}''_j}\psi_{ij} + \psi_{j0}\right) \tag{2.16}$$

where σ is the sigmoid function, $\sigma(z) = (1+e^{-z})^{-1}$.

To clarify how the mean field method relates to the guiding principles of variational approximations mentioned at the beginning of the section, let us examine the log-likelihood and its bound once again: Recall that the probability of the model is written as

$$P(\boldsymbol{x}) = \frac{1}{Z_P}\prod_i \phi_i(\boldsymbol{d}_i)$$

The variables of our domain are *coupled* because every X_i is contained in more then one subset. This coupling might make the variable elimination inefficient for computing the marginal probability,

$$P(\boldsymbol{o}) = \sum_{\boldsymbol{t}} P(\boldsymbol{t},\boldsymbol{o})$$

Rewriting equation 2.9 we can lower bound the log-likelihood, using another posterior distribution $Q(\boldsymbol{T})$, by,

$$\mathcal{F}[Q] = \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log P(\boldsymbol{t},\boldsymbol{o}) - \sum_{\boldsymbol{t}} Q(\boldsymbol{t})\log Q(\boldsymbol{t})$$

Q can be regarded as a new set of parameters transforming our problem into an *optimization* problem: We now wish to find the set of parameters that maximize the lower bound. The log-likelihood is recovered exactly if $Q(\boldsymbol{T}) = P(\boldsymbol{T}\mid\boldsymbol{o})$, but this is as hard as the original problem. Constraining the set of additional parameters to represent a factored distribution simplifies $\mathcal{F}$ at the expense of accuracy. The new set of parameters $\{Q(x_j)\}$ *decouples* the variables in $\{X_j\}$ into smaller groups as can be seen by writing $\mathcal{F}[Q]$ explicitly,

$$\mathcal{F}[Q] = \sum_i \sum_{x_i,\mathbf{pa}_i}\left[\prod_{j\in\{i,\mathbf{Pa}_i\}} Q(x_j)\right]\log\phi_i(\boldsymbol{d}_i) - \log Z_P - \sum_j\sum_{x_j} Q(x_j)\log Q(x_j) \tag{2.17}$$

Now, instead of summing $|dom(\boldsymbol{T})|$ expressions to compute the log likelihood, we calculate a lower bound on it with a number of operations that is approximately equal to the total size of the domains of the individual families. As we have seen before, the ratio can be exponential in the number of variables.

2.5.3 The convex duality principle

In this section we shall see that the method of approximating a distribution by another through the minimization of the KL-divergence, introduced in Section 2.5.1, arises as a special case of a more general variational principle termed the *convex duality principle*. Apart from mathematical elegance the introduction of the convex duality principle has additional merits:

- It enables the formulation of variational approximations for some dense graphical models whose local probability functions are given in a parametric form.
- It provides a tool for improving the accuracy of the approximations and increasing the ability to tradeoff between complexity and accuracy.

We should note that apart from using this principle, there are also other techniques for devising variational transformations for intractable functions.

The convex duality principle allows us to compute lower and upper bounds on a family of functions. These can be exploited to formulate variational transformation for some classes of probability distributions. The bounds are attained using the dual representation of concave functions. A generalized formulation of the principle can be find at Rockafellar and Wets [27]. We shall use the formulation from Jordan et al. [19] which will suit our needs. The principle states that if $f(\boldsymbol{x})$ is a real valued continuous *concave* function defined over a convex subset in $\mathcal{R}^n$ then $f(\boldsymbol{x})$ can be represented via its *conjugate* or *dual* function as,

$$f(\boldsymbol{x}) = \min_{\boldsymbol{\lambda}}\{\boldsymbol{\lambda}^T\boldsymbol{x} - f^*(\boldsymbol{\lambda})\} \tag{2.18}$$

where $\boldsymbol{\lambda} \in \mathcal{R}^n$ and $\boldsymbol{\lambda}^T$ is the transpose of $\boldsymbol{\lambda}$. The conjugate function $f^*(\boldsymbol{\lambda})$ can be obtained from the following dual expression,

$$f^*(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}}\{\boldsymbol{\lambda}^T\boldsymbol{x} - f(\boldsymbol{x})\} \tag{2.19}$$

At a first glance the relation between this principle and our previous discussion seems vague. Before we show this connection we will illustrate its usage with a simple example. Consider the function $f(x) = \log x$. The conjugate function can be found by equation 2.19 to be $f^*(\lambda) = \log\lambda + 1$. Using equation 2.18, we can see that

$$\log x = \min_{\lambda}\{\lambda x - \log\lambda - 1\} \tag{2.20}$$

Thus, for every $\lambda > 0$, we get the upper bound $\log x \leq \lambda x - \log\lambda - 1$ which is a linear function of x. Note that the tightness of this bound depends on how far is x from the value x' that attains the minimum in equation 2.19, equality is achieved at $\lambda = \frac{1}{x}$. This bound will be used in the next chapter.

Approximating a local probability model

Another example of using the convex duality principle is providing upper bounds to the logistic function, $\sigma(x) = (1 + e^{-x})^{-1}$. This function is used to model local conditional probabilities in a class of Bayesian network called *sigmoid belief networks*. In those models the nodes represent binary valued random variables. The conditional probability of a child node given his parents is modeled as,

$$P(X_i = 1 \mid \mathbf{pa}_i) = \sigma\left(\sum_{\{j|X_j\in\mathbf{Pa}_i\}} J_{ij}x_j + h_i\right) \tag{2.21}$$

This local model renders a compact representation even for dense Bayesian networks of the type shown in Figure 2-5. Jaakkola and Jordan [15] used the dual representation to compute an upper bound on probabilities in such networks. This approximation demonstrates how the convex duality principle can be applied for approximation of dense parametric models and it demonstrates that in some cases the variational transformations should be tailored to the specific problem.

The logistic function is not a concave function but it is *log concave*, i.e. $\log(1+e^{-x})^{-1}$ is concave. A sufficient condition for the concavity of a function is the non-positivity of the second derivative. We can verify the concavity of the log-logistic function by using this condition:

$$\begin{aligned}\frac{\partial \log(1+e^{-x})^{-1}}{\partial x} &= (1+e^{x})^{-1}\\ \frac{\partial^2 \log(1+e^{-x})^{-1}}{\partial x^2} &= \frac{-e^{x}}{(1+e^{x})^2} < 0\end{aligned}$$

We can find an upper bound using the conjugate function:

$$f^*(\lambda) = min_x\{\lambda x - \log\sigma(x)\} = -\lambda\log\lambda - (1-\lambda)\log(1-\lambda)$$

Denoting $H(\lambda) = -\lambda\log\lambda - (1-\lambda)\log(1-\lambda)$ and using equation 2.18 we get that

$$\log\sigma(x) \leq \lambda x - H(\lambda)$$

thus,

$$\sigma(x) \leq e^{\lambda x - H(\lambda)} \tag{2.22}$$

Jaakkola and Jordan used this bound to compute an upper bound on marginal probabilities for two layered sigmoid belief networks that are expressed as follows:

$$P(\boldsymbol{x}) = \prod_{j\in L_1} P(x_j)\cdot\prod_{i\in L_2} P(x_i \mid \mathbf{pa}_i)$$

They used these networks as a probabilistic reformulation of the QMR knowledge base which is used in medical diagnosis [15]. The problem they addressed was to compute bounds on the likelihood of full assignment to the variables in the second layer (L_2). Using the bound from Equation 2.22 and the fact that $1-\sigma(x) = \sigma(-x)$, the joint distribution can be bounded as follows:

$$\begin{aligned}P(\boldsymbol{x}) &= \prod_{j\in L_1} P(x_j)\cdot\prod_{i\in L_2}\sigma(\ (2x_i-1)(\sum_{X_j\in\mathbf{Pa}_i} J_{ij}x_j + h_i)\ )\\ &\leq \prod_{j\in L_1} P(x_j)\cdot\prod_{i\in L_2} e^{\lambda_i\cdot(2x_i-1)(\sum_{X_j\in\mathbf{Pa}_i} J_{ij}x_j+h_i)-H(\lambda_i)}\\ &= \prod_{i\in L_2} e^{\lambda_i h_i - H(\lambda_i)}\cdot\prod_{j\in L_1} P(x_j)e^{(\sum_{X_i\in\boldsymbol{Ch}_i}\lambda_i(2x_i-1)J_{ij})x_j}\end{aligned}$$

Given a full assignment to the variables in the second layer, the upper bound is a completely factorized representation of the joint distribution of the variables in the first layer and the evidence.

In order to get the tightest upper bound we should optimize it with respect to the variational parameters $\{\lambda_i\}_{i \in L_1}$. Details of the optimization scheme can be found in [15].

The convex duality principle and KL-distance minimization

We will now show that the KL-minimization principle can be derived by a dual transformation. Suppose P is a probability model over a set of discrete valued random variables. We would like to compute a lower bound for

$$\log P(\boldsymbol{o}) = \log \sum_{\boldsymbol{t}} P(\boldsymbol{t}, \boldsymbol{o}) \tag{2.23}$$

We shall denote by $\boldsymbol{x}$ a $|dom(\boldsymbol{T})|$ dimensional vector. The components of $\boldsymbol{x}$ will be indexed by the possible assignments to $\boldsymbol{T}$, and their values are

$$\boldsymbol{x}_{\boldsymbol{t}} \equiv \log P(\boldsymbol{t}, \boldsymbol{o})$$

The target function is defined as

$$f(\boldsymbol{x}) \equiv \log \sum_{\boldsymbol{t}} e^{\boldsymbol{x}_{\mathbf{t}}} = \log P(\boldsymbol{o})$$

Finally, $\boldsymbol{\lambda}$ will be a vector in the dual space which can be regarded as

$$\boldsymbol{\lambda}_{\boldsymbol{t}} \equiv Q(\boldsymbol{t}; \Theta)$$

The function $f(\boldsymbol{x})$ is concave. Therefore, we can write the dual function using equation 2.19 and the above definitions,

$$f^*(\boldsymbol{\lambda}) = \min_{\boldsymbol{x}} \{\boldsymbol{\lambda}^T \boldsymbol{x} - \log \sum_{\boldsymbol{t}} e^{\boldsymbol{x}_{\mathbf{t}}}\} \tag{2.24}$$

In order to find the minimum we equate the partial derivative to zero,

$$\lambda_{\boldsymbol{t}_\circ} - \frac{e^{\boldsymbol{x}_{\mathbf{t}_\circ}}}{\sum_{\boldsymbol{t}} e^{\boldsymbol{x}_{\mathbf{t}}}} = 0$$

Thus at a minimum point we get that

$$\boldsymbol{x}_{\boldsymbol{t}_0} = \log \lambda_{\boldsymbol{t}_\circ} + \log \sum_{\boldsymbol{t}} e^{\boldsymbol{x}_{\mathbf{t}}}$$

Plugging this equation to Equation 2.24 and using the fact that $\sum_{\boldsymbol{t}} \lambda_{\boldsymbol{t}} = 1$ the dual function becomes

$$f^*(\boldsymbol{\lambda}) = \sum_{\boldsymbol{t}} \lambda_{\boldsymbol{t}} \log \lambda_{\boldsymbol{t}} \tag{2.25}$$

Using Equation 2.18 and the definitions from the beginning of this section we get the following lower bound:

$$\log P(\boldsymbol{o}) \quad = \quad f(\boldsymbol{x})$$

$$\begin{aligned} &\geq \sum_{\boldsymbol{t}} \lambda_{\boldsymbol{t}} x_{\boldsymbol{t}} - \lambda_{\boldsymbol{t}} \log \lambda_{\boldsymbol{t}} \\ &= \sum_{\boldsymbol{t}} Q(\boldsymbol{t}) \log P(\boldsymbol{t}, \boldsymbol{o}) - \sum_{\boldsymbol{t}} Q(\boldsymbol{t}) \log Q(\boldsymbol{t}) \end{aligned}$$

which is the same lower bound we got by minimizing the KL divergence.

Chapter 3

Augmenting variational approximations

The mean field method introduced in section 2.5.2 approximates the posterior by a distribution in which the hidden variables are independent of each other. This assumption renders the mean field approximation suitable for graphical models with highly dense connections and whose conditional probabilities are given in a parametric form. In this class of networks, the method can be justified by the fact that every node averages the influence of a *large* number of neighbors. The mean field approximation breaks down when applied on sparse graphical models. As we have seen previously, inference can be hard even for graphical models with bounded fan-in. For such models the mean field method is not expected to work well.

The mean field approximation can be improved by assuming that the posterior distribution of the hidden variables has a non-trivial structure. Saul and Jordan [29] proposed the *structured mean field* approximation. In this method the graphical model is decomposed to independent *substructures*, containing several nodes. The idea of structured approximation was extended and generalized by the emergence of algorithms that can use tractable Bayesian networks and Markov models to approximate complicated ones [2, 30].

Another direction for improvement is using mixture models approximations [17]. In this method, the approximating distribution is a mixture of distributions, in which the variables are assumed to be independent given the component of the mixture. This method adds the ability to approximate multi-modal posteriors. Also, this modeling induces dependencies between the variables. Thus enabling modeling the posterior without independency assumptions.

Both refinements of the mean field approximation provide a way to improve accuracy at the cost of increasing complexity at an adjustable rate: In the structured approximation adding interactions between variables and in the mixture approach adding mixture components. In this chapter we shall propose generalizations of current approximation methods. We shall enhance both the approach of adding structure to the approximation and of using mixture models. Section 3.1 presents the structured approximation equations for discrete valued graphical models. Section 3.2 presents a way to broaden the class of dependency relations imposed on the approximating probability distribution. In section 3.3 we shall enhance the structured approximation through the addition of extra hidden variables to the approximating model. This chapter extends the material presented in [10].

3.1 Structured approximations

Structured approximations approximate a probability distribution using another probability distribution with non-trivial dependency structure. In this section we re-derive standard structured approximation schemes with Bayesian networks (such as the ones in [12, 2, 30]) using tools that will facilitate later developments.

Suppose P is a distribution defined over the set of random variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. Let $\boldsymbol{O} \subset \boldsymbol{X}$ be the subset of observed variables. We denote by $\boldsymbol{T} = \boldsymbol{X} \backslash \boldsymbol{O}$ the set of hidden variables. Our task is to approximate the distribution $P(\boldsymbol{T} \mid \boldsymbol{o})$ by another distribution $Q(\boldsymbol{T}; \Theta)$, where Θ is the set of parameters for the approximating distribution Q.

In this chapter we will restrict P to be a discrete graphical model, i.e. $\{X_1, \ldots, X_n\}$ are discrete random variables and P is given as

$$P(\boldsymbol{x}) = \frac{1}{Z_P} \prod_{i=1}^{k} \phi_i(\boldsymbol{d}_i) \tag{3.1}$$

Where $\boldsymbol{D}_1, \ldots \boldsymbol{D}_k$ are subsets of $\boldsymbol{X}$. The principles presented in this chapter apply for many classes of graphical models. Though some parametric forms require additional variational transformation in order to apply these methods.

The approximating distribution will be represented as another graphical model. It can be either a discrete multinomial Bayesian network or a discrete Markov network. We will begin by using a Bayesian network that is defined as follows:

$$Q(\boldsymbol{t}; \Theta) = \prod_j Q(x_j \mid \mathbf{u}_j) = \prod_j \theta_{x_j \mid \mathbf{u}_j} \tag{3.2}$$

where $\mathbf{U}_j$ denotes the parents of X_j in the *approximating* Network, and $\theta_{x_j \mid \mathbf{u}_j}$ are the parameters of the distribution.

Following section 2.5.1 we wish to find the set of parameters Θ that minimizes the KL-divergence

$$KL(Q(\boldsymbol{T}) || P(\boldsymbol{T} \mid \boldsymbol{o})) = E_{Q(\boldsymbol{T})}\left[\log \frac{Q(\boldsymbol{T})}{P(\boldsymbol{T} \mid \boldsymbol{o})}\right] \tag{3.3}$$

As we have seen, this is equivalent to maximizing the lower bound $\mathcal{F}[Q]$ on the log-likelihood defined on Equation 2.9. Before we discuss the maximization procedure we will generalize the definition of $\mathcal{F}[Q]$ to include conditional distributions. Given a subset of random variables $\boldsymbol{C} \subseteq \boldsymbol{T}$ and an assignment $\boldsymbol{c}$, $\mathcal{F}[Q(\boldsymbol{T})|\boldsymbol{c}]$ is defined as

$$\mathcal{F}[Q(\boldsymbol{T})|\boldsymbol{c}] = E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log P(\boldsymbol{T}, \boldsymbol{c}, \boldsymbol{o})] - E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log Q(\boldsymbol{T} \mid \boldsymbol{c})] \tag{3.4}$$

Now $\mathcal{F}[Q]$ is a shorthand for $\mathcal{F}[Q(\boldsymbol{T})|\emptyset]$. We shall also use $\mathcal{F}[Q|\boldsymbol{c}]$ as a shorthand for $\mathcal{F}[Q(\boldsymbol{T})|\boldsymbol{c}]$ when the domain of Q is clear from the context. The generalized definition will be used in the formulation of the maximization procedure.

The computational complexity of calculating the lower bound depends on the computational complexity of inference in Q and on the domain size of the factors of P. To see that let us rewrite $\mathcal{F}[Q|\boldsymbol{c}]$ in a factored form, using the following lemma:

Lemma 3.1.1 *Let* $Q(\boldsymbol{t}) = \prod_j Q(x_j \mid \mathbf{u}_j)$, *where* $\boldsymbol{T} = \boldsymbol{X} \setminus \boldsymbol{O}$. *then*

$$\mathcal{F}[Q|\boldsymbol{c}] = \sum_i E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log \phi_i(\boldsymbol{D}_i, \boldsymbol{o})] - \sum_j E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log Q(X_j \mid \mathbf{U}_j)] + \log Q(\boldsymbol{c}) - \log Z_P$$

Where $\phi_i(\boldsymbol{D}_i, \boldsymbol{o})$ *represents a random variable whose value is* $\phi_i(\boldsymbol{d}_i)$ *if* $\boldsymbol{d}_i$ *is consistent with* $\boldsymbol{o}$*; Otherwise it is* 0.

Proof:

$$\begin{aligned}\mathcal{F}[Q|\boldsymbol{c}] &= \sum_{\boldsymbol{x} \models \boldsymbol{c}, \boldsymbol{o}} Q(\boldsymbol{t} \mid \boldsymbol{c}) \log \frac{1}{Z_P} \prod_i \phi_i(\boldsymbol{d}_i) - \sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid \boldsymbol{c}) \log \frac{1}{Q(\boldsymbol{c})} \prod_j Q(x_j \mid \mathbf{u}_j) \\ &= \sum_i \sum_{\boldsymbol{x} \models \boldsymbol{c}, \boldsymbol{o}} Q(\boldsymbol{t} \mid \boldsymbol{c}) \log \phi_i(\boldsymbol{d}_i) - \sum_j \sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid \boldsymbol{c}) \log Q(x_j \mid \mathbf{u}_j) + \log Q(\boldsymbol{c}) - \log Z_P\end{aligned}$$

Thus

$$\mathcal{F}[Q|\boldsymbol{c}] = \sum_i E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})] - \sum_j E_{Q(\boldsymbol{T}|\boldsymbol{c})}[\log Q(x_j \mid \mathbf{u}_j)] + \log Q(\boldsymbol{c}) - \log Z_P$$

∎

Our goal is to find a set of parameters maximizing $\mathcal{F}$ while conforming to the local normalization constrains. The optimal parameters for Q are found by writing the Lagrangian for this problem and differentiating it with respect to them. The Lagrangian is

$$\mathcal{J}_{BN} = \mathcal{F}[Q] - \sum_j \sum_{\mathbf{u}_i} \lambda_{\mathbf{u}_j} (\sum_{x_j} Q(x_j \mid \mathbf{u}_j) - 1)$$

To differentiate the Lagrangian we shall use the following technical results.

Lemma 3.1.2 *Let* $Q = \prod_j \theta_{x_j|\mathbf{u}_j}$. *Let* $\boldsymbol{C} \subset \boldsymbol{T}$ *be a subset of random variables. The partial derivative of* $Q(\boldsymbol{c})$ *with respect to* $\theta_{x_j|\mathbf{u}_j}$ *is*

$$\frac{\partial Q(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} = \frac{Q(\boldsymbol{c}, x_j, \mathbf{u}_j)}{\theta_{x_j|\mathbf{u}_j}} = Q(\boldsymbol{c} \mid x_j, \mathbf{u}_j) Q(\mathbf{u}_j) \qquad \forall\, \boldsymbol{C} \subset \boldsymbol{T}$$

Proof:

$$\frac{\partial Q(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} = \frac{\partial}{\partial \theta_{x_j|\mathbf{u}_j}} \sum_{\boldsymbol{t} \models \boldsymbol{c}} Q(\boldsymbol{t}) = \sum_{\boldsymbol{t} \models \boldsymbol{c}} \frac{\partial}{\partial \theta_{x_j|\mathbf{u}_j}} \prod_{j'} \theta_{x_{j'}|\mathbf{u}_{j'}} = \frac{1}{\theta_{x_j|\mathbf{u}_j}} \sum_{\boldsymbol{t} \models \boldsymbol{c}, x_j, \mathbf{u}_j} \prod_{j'} \theta_{x_{j'}|\mathbf{u}_{j'}}$$

The sum in the last row equals $Q(\boldsymbol{c}, x_j, \mathbf{u}_j)$. Therefore

$$\frac{\partial Q(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} = \frac{Q(\boldsymbol{c}, x_j, \mathbf{u}_j)}{\theta_{x_j|\mathbf{u}_j}} = Q(\boldsymbol{c} \mid x_j, \mathbf{u}_j) Q(\mathbf{u}_j)$$

∎

Corollary 3.1.1 *Let $Q(\boldsymbol{t}) = \prod_j Q(x_j \mid \mathbf{u}_j)$, then*

$$\frac{\partial E_Q\left[f(\boldsymbol{C})\right]}{\partial Q(x_j \mid \mathbf{u}_j)} = Q(\mathbf{u}_j) \cdot E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[f(\boldsymbol{C})\right] + E_Q\left[\frac{\partial f(\boldsymbol{C})}{\partial Q(x_j \mid \mathbf{u}_j)}\right]$$

Using Corollary 3.1.1 we can differentiate $\mathcal{F}$

$$\frac{\partial \mathcal{F}[Q]}{\partial \theta_{x_j|\mathbf{u}_j}} = \frac{\partial}{\partial \theta_{x_j|\mathbf{u}_j}}\left(\sum_i E_Q\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] - \sum_{j'} E_Q\left[\log Q(X_{j'} \mid \mathbf{U}_{j'})\right]\right) \tag{3.5}$$

and we get

$$\frac{\partial \mathcal{F}[Q]}{\partial \theta_{x_j|\mathbf{u}_j}} = Q(\mathbf{u}_j)\sum_i E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] - Q(\mathbf{u}_j)\sum_{j'} E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log Q(X_{j'} \mid \mathbf{U}_{j'})\right] + 1 \tag{3.6}$$

Equating the derivative of the Lagrangian to zero, dividing by $Q(\mathbf{u}_j)$ and rearranging, we get that the condition for a stationary point of the lower bound becomes,

$$\log Q(x_j \mid \mathbf{u}_j) = \sum_i E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] - \sum_{j' \neq j} E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log Q(X_{j'} \mid \mathbf{U}_{j'})\right] + 1 - \frac{\lambda_{\mathbf{pa}_j}}{Q(\mathbf{u}_j)}$$

Denoting

$$\mathcal{E}_{BN}(x_j, \mathbf{u}_j) \equiv \sum_i E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] - \sum_{j' \neq j} E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log Q(X_{j'} \mid \mathbf{U}_{j'})\right] \tag{3.7}$$

The condition for a stationary point becomes

$$Q(x_j \mid \mathbf{u}_j) = \frac{1}{Z_{\mathbf{u}_j}} \cdot e^{\mathcal{E}_{BN}(x_j, \mathbf{u}_j)} \tag{3.8}$$

Where $Z_{\mathbf{u}_j}$ is determined by local normalization. Note that $\mathcal{E}_{BN}(x_j, \mathbf{u}_j)$ does not depend on $Q(x_j \mid \mathbf{u}_j)$.

To better understand the characterization of the solution, we examine the terms $\mathcal{E}_{BN}(x_j, \mathbf{u}_j)$ and $\mathcal{F}\left[Q|x_j, \boldsymbol{u}_j\right]$. It is easy to verify that

$$KL(Q(\boldsymbol{T} \mid x_j, \boldsymbol{u}_j)||P(T \mid x_j, \boldsymbol{u}_j, \boldsymbol{o})) = -\mathcal{F}\left[Q|x_j, \boldsymbol{u}_j\right] + \log P(x_j, \boldsymbol{u}_j, \boldsymbol{o})$$

Thus $\mathcal{F}\left[Q|x_j, \boldsymbol{u}_j\right]$ is a lower bound on $\log P(x_j, \boldsymbol{u}_j, \boldsymbol{o})$. Equation 3.7 and Lemma 3.1.1 provide us the following relation between $\mathcal{F}$ and $\mathcal{E}_{BN}$:

$$\begin{aligned} \mathcal{E}_{BN}(x_j, \mathbf{u}_j) &= \mathcal{F}\left[Q|x_j, \boldsymbol{u}_j\right] + \log Q(x_j \mid \mathbf{u}_j) - \log Q(x_j, \mathbf{u}_j) + \log Z_P & (3.9) \\ &= \mathcal{F}\left[Q|x_j, \boldsymbol{u}_j\right] - \log Q(\mathbf{u}_j) + \log Z_P & (3.10) \end{aligned}$$

This suggests that the global optimization criterion leads also to a local optimization criterion. $Q(x_j \mid \mathbf{u}_j)$ is updated to be the lower bound of $P(x_j \mid \mathbf{u}_j, \boldsymbol{o})$.

In order to find optimal parameters, we use an iterative procedure that updates the parameters of one family on each iteration. An asynchronous update of the parameters according to equation 3.8 guarantees a monotonic increase in the lower bound $\mathcal{F}[Q]$ and converges to a local maximum. This is a consequence of the fact that, for every i and every assignment to the parents $\mathbf{u}_i$, $\mathcal{F}$ is a concave function of the set of parameters $\{Q(x_i \mid \mathbf{u}_i) \mid x_i \in dom(X_i)\}$. Therefore, the stationary point is a global maximum with respect to those parameters.

We can verify the concavity of $\mathcal{F}$ by examining the second order partial derivatives

$$\frac{\partial^2 \mathcal{F}}{\partial \theta^2_{x_j|\mathbf{u}_j}} = -\frac{1}{\theta^2_{x_j|\mathbf{u}_j}} Q(x_j, \mathbf{pa}_j) < 0$$

The mixed partial derivatives are all zero. [1]

The complexity of calculating $\mathcal{E}_{BN}$ as defined in equation 3.7 is determined by the number of variables, the size of the families in P and by the complexity of calculating marginal probabilities in Q. Not all the terms in this equation need to be computed. Let $\boldsymbol{U} \subset \boldsymbol{T}$ be a subset that is independent of X_i given $\mathbf{U}_i$ in Q, i.e. for every $\boldsymbol{u} \in dom(\boldsymbol{U})$, $Q(\boldsymbol{u} \mid x_i, \mathbf{u}_i) = Q(\boldsymbol{u} \mid \mathbf{u}_i)$. Terms of the form $E_{Q(\boldsymbol{u}|x_i,\mathbf{u}_i)}[f(\boldsymbol{u})]$ can be ignored in the update equations since they change the new parameters by a constant factor which will be absorbed in $C_{\mathbf{pa}_i}$. Therefore we can reduce the amount of computations by defining the sets of indices of the factors that depend on X_i given $\mathbf{Pa}_i$ as follows:

$$\begin{aligned} F^p_j &= \{i : Q \not\models Ind(X_j; \boldsymbol{D}_i \mid \mathbf{U}_j)\} \\ F^q_j &= \{j' \neq j : Q \not\models Ind(X_j; X_{j'}, \mathbf{U}_{j'} \mid \mathbf{U}_j)\} \end{aligned}$$

We can redefine $\mathcal{E}_{BN}$ to be

$$\mathcal{E}(x_i, \mathbf{u}_i) \equiv \sum_{j \in F^p_i} E_{Q(\cdot|x_i,\mathbf{u}_i)}[\log \phi(\boldsymbol{D}_j, \boldsymbol{o})] - \sum_{j \in F^q_i} E_{Q(\cdot|x_i,\mathbf{u}_i)}[\log Q(X_j \mid \mathbf{U}_j)] \tag{3.11}$$

This formula involves less calculations and it also enables efficient sequential calculations of $\mathcal{E}_{BN}$ in some cases as will be seen in the following example.

Figure 3-1 presents a Bayesian Network representing an approximated distribution the one presented in figure 2-6. In this approximate distribution the different state variables are independent of each other within a time slice but the dependency of a state variable in it's previous condition is maintained. Such an approximation is expected to be more accurate then the mean field approximation. The distribution defined by this approximation can be written as

$$Q(\boldsymbol{T}) = \prod_i \left(Q(t^i_1) \prod_{n=2}^{N} Q(t^i_n \mid t^i_{n-1}) \right)$$

[1] $\frac{\partial \mathcal{F}}{\partial \theta_{x_j|\mathbf{u}_j}}$ does not depend on $\theta_{x'_j|\mathbf{u}_j}$. In Equation 3.6 we can see that this derivative is composed of $Q(\mathbf{u}_j)$ and conditional probabilities $Q(\cdot \mid x_j, \mathbf{u}_j)$ which do not depend on $\theta_{x'_j|\mathbf{u}_j}$. The terms inside the expectations are parameters of the distributions Q and P excluding $\theta_{x'_j|\mathbf{u}_j}$.

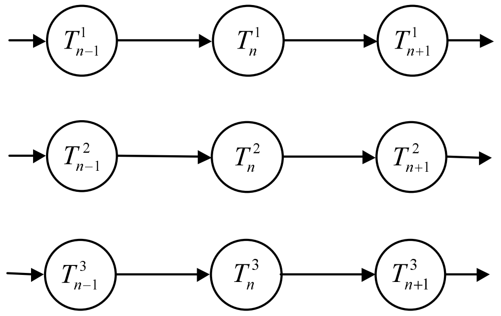


Figure 3-1: A Bayesian network describing an approximating distribution for the network in figure 2-6. The edges connecting variables within a time slice have been dropped but the inter time influence is maintained.

Using equation 3.11 we can write

$$\mathcal{E}(t_n^i, t_{n-1}^i) = \sum_{j=i,i+1} \sum_{r \geq n} E_{Q(\cdot|t_r^i, t_{r-1}^i)} \left[\log P(t_r^j \mid t_{r-1}^j, t_r^{j-1}) \right] - \sum_{r>n} E_{Q(\cdot|t_r^i, t_{r-1}^i)} \left[\log Q(t_r^i \mid t_{r-1}^i) \right]$$

Recall that the complexity of exact inference is exponential in the number of state variable within a single time slice. The complexity of the above approximation is cubic in the number of states that every variable can assume. It is linear in the number of variables.

Returning to the general problem of approximating a discrete graphical model, we can use a Markov network as an approximating model. In that case Q is defined as,

$$Q(\boldsymbol{t}) = \frac{1}{Z_Q} \prod_{i=1}^{l} \psi_i(\boldsymbol{d}_i^q) \tag{3.12}$$

where $\boldsymbol{D}_1^q, ..., \boldsymbol{D}_l^q$ are subsets of $\boldsymbol{T}$. The update equations for the parameters of Q are

$$\psi_i(\boldsymbol{d}_i^q) = C \cdot e^{\mathcal{E}(\boldsymbol{d}_i^q)} \tag{3.13}$$

where

$$\mathcal{E}(x_i, \mathbf{u}_i) \equiv \sum_j E_{Q(\cdot|\boldsymbol{d}_i^q)} \left[\log \phi(\boldsymbol{D}, \boldsymbol{o}) \right] - \sum_{j \neq i} E_{Q(\cdot|\boldsymbol{d}_i^q)} \left[\log \psi_i(\boldsymbol{D}_i^q) \right] \tag{3.14}$$

and C is a *global* normalization constant that is found by satisfying the condition $Z_Q = \sum_{\boldsymbol{t}} \prod_i \psi_i(\boldsymbol{d}_i^q)$. Note that this update procedure is similar to the one that is used for Bayesian networks. The difference is that here we have to satisfy a global constraint on the parameters and on Bayesian networks we have a set of local constraints.

3.2 Enhancing the range of dependency assumptions

As we have seen in section 2.1.2, the classes of dependency models that can be represented by Markov networks and by Bayesian networks are not equivalent. Therefore, for some distributions the best tractable approximations might be represented by Bayesian networks while for

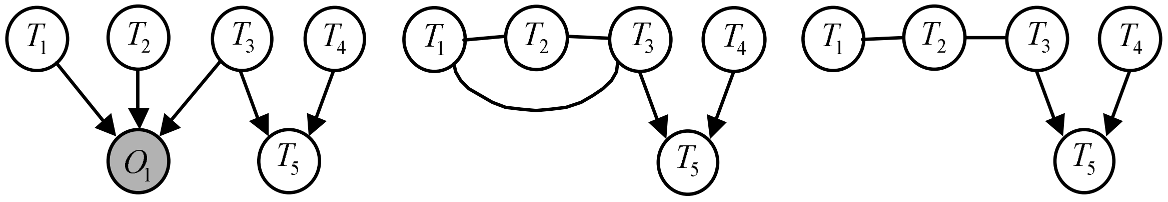


Figure 3-2: (a) A Bayesian network with an observed variable (O_1). (b) A representation of the posterior distribution as a chain graph. (c) an approximating chain graph network.

other distributions the best approximation is a Markov network. We can gain more flexibility in the choice of an approximating distribution by using a more general class of probability models that can capture the dependency models implied by Bayesian networks, Markov networks and dependency models that can be captured by neither of them.

To consider a concrete example, suppose that P is a Bayesian network. What is the form of the posterior $P(\boldsymbol{T} \mid \boldsymbol{o})$? For a concrete example, consider the network of Figure 3-2(a). When, we observe the value of O_1, we create dependencies among the variables T_1, T_2, and T_3. The posterior distribution is neither a Bayesian network nor a Markov network (because of the v-structure in the parents of T_5). Instead, we can write this posterior in the form:

$$\psi(T_1, T_2, T_3) P(T_1) P(T_2) p(T_3) p(T_4) P(T_5 \mid T_3, T_4)$$

where $\psi(T_1, T_2, T_3) = \frac{1}{P(o_1)} P(o_1 \mid T_1, T_2, T_3)$ is a potential that is induced by the observation of o_1.

A natural class of models that has this general form are ones that factorize to a product of conditional distributions and potentials. We shall refer these models as *chain graphs*. Formally, we define a chain graph to have for each variable a (possibly empty) set of parents, and in addition to have a set of potentials on some subsets of variables. Note that our definition of a chain graph distribution is different than the conventional one (see Cowell et al. [6] for the conventional definition). Here we will use the this term because we can conveniently represent them using the graphical structure of a chain graph, as shown in the example of Figure 3-2(b).

The approximating distribution, Q, will have the form:

$$Q(\boldsymbol{T}) = \frac{1}{Z_Q} \prod_j Q(x_j \mid \mathbf{u}_j) \prod_k \psi_k(\boldsymbol{c}_k)$$

where, as before, $\mathbf{U}_j$ are the directed parents of X_j. In addition, ψ_k are potential functions on subsets of $\boldsymbol{T}$, and $Z_Q = \sum_{\boldsymbol{t}} \prod_j Q(x_j \mid \mathbf{u}_j) \prod_k \psi_k(\boldsymbol{c}_k)$ is a normalizing function that ensures that the distribution sums to 1. Figure 3-2(b) shows the chain graph that represents this factorization.

It is easy to check that if P is a Bayesian network, then $P(\boldsymbol{T} \mid \boldsymbol{o})$ can be represented as a chain graph (for each variable X_j in $\boldsymbol{O}$, add a potential over the parents of X_j). In contrast, it is easy to build examples where the posterior distribution cannot be represented by a Bayesian network without introducing unnecessary dependencies. Thus, this class of models is, in some sense, a natural representation of conditional distributions in Bayesian networks.

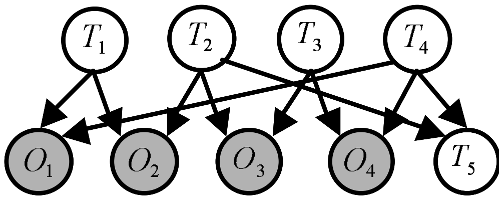


Figure 3-3: A Bayesian network with observed variables. The dependency model induced on the hidden variables can not be captured by a Markov network nor by a Bayesian that are defined only over the hidden variables (see text).

Figure 3-3 shows an example for a model in which the dependency model of the posterior can not be represented by a Bayesian network or a Markov network that is defined only over the hidden variables. If T_5 is not observed the dependency structure of T_1, T_2, T_3 and T_4 is the same as the one of the Markov network in Figure 2-2. Therefore, there is no Bayesian network that represent this dependency structure without extra observed variables. Given assignment to T_1 and to T_3, we have a v-structure, $T_2 \to T_5 \leftarrow T_4$, which can not be modeled by a Markov network.

The previous argument suggests that by considering chain graphs we can represent approximate distributions that are more tractable than the original distribution, yet are closer to the posterior we want to approximate. For example, Figure 3-2(c) shows a simple example for a possible approximate network for representing the posterior of the network of Figure 3-2(a). In this network there are two potentials with two variables each, rather than one with three variables.

Given the structure of the approximating chain graph, we wish to find the set of parameters that maximizes $\mathcal{F}[Q]$, the lower bound on the log-likelihood. As usual, we need to define a Lagrangian that capture the constraints on the model. These constraints contain the constraints that appeared in the Bayesian network case, and, in addition, we require that each potential sums up to one:

$$\sum_{\boldsymbol{c}_k} \psi_k(\boldsymbol{c}_k) = 1 \ \ \forall k$$

To understand this constraint, note that the each potential can be scaled without changing Q, since the scaling constant is absorbed in Z_Q. Thus, without constraining the scale of each potential there is a continuum of solutions, and the magnitude of values in the potentials can explode.

Putting these together, the Lagrangian has the form:

$$\mathcal{J}_{CG} = \mathcal{F}[Q] - \sum_j \sum_{\mathbf{u}_i} \lambda_{\mathbf{u}_j} \sum_{x_j} Q(x_j \mid \mathbf{u}_j) - \sum_k \lambda_k \sum_{\boldsymbol{c}_k} \psi_k(\boldsymbol{c}_k)$$

The main difference from the Bayesian network approximation is in the form of the analogue of Lemma 3.1.2. In the case of chain graphs, we also have to differentiate Z_Q, and so we get slightly more complex derivatives.

Lemma 3.2.1 *If Q is a chain graph over $\boldsymbol{T}$, then*

$$\begin{aligned}
\frac{\partial Q(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} &= \frac{Q(x_j, \mathbf{u}_j)}{\theta_{x_j|\mathbf{u}_j}} \cdot [Q(\boldsymbol{c} \mid x_j, \mathbf{u}_j) - Q(\boldsymbol{c})] \\
\frac{\partial Q(\boldsymbol{c})}{\partial \psi_k(\boldsymbol{c}_k)} &= \frac{Q(\boldsymbol{c}_k)}{\psi_k(\boldsymbol{c}_k)} \cdot [Q(\boldsymbol{c} \mid \boldsymbol{c}_k) - Q(\boldsymbol{c})]
\end{aligned}$$

Proof:

$$Q(\boldsymbol{c}) = \frac{\sum_{\boldsymbol{t} \models \boldsymbol{c}} \prod_j Q(x_j \mid \mathbf{u}_j) \prod_k \psi_k(\boldsymbol{c}_k)}{Z_Q}$$

Denoting $\Psi(\boldsymbol{c}) = \sum_{\boldsymbol{t} \models \boldsymbol{c}} \prod_j Q(x_j \mid \mathbf{u}_j) \prod_k \psi_k(\boldsymbol{c}_k)$, $Q(\boldsymbol{c})$ can be written as $\Psi(\boldsymbol{c})/Z_Q$.

$$\begin{aligned}
\frac{\partial Q(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} &= \frac{\partial}{\partial \theta_{x_j|\mathbf{u}_j}} \frac{\Psi(\boldsymbol{c})}{Z_Q} \\
&= \frac{1}{{Z_Q}^2} \left[\frac{\partial \Psi(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} \cdot Z_Q - \Psi(\boldsymbol{c}) \cdot \frac{\partial Z_Q}{\partial \theta_{x_j|\mathbf{u}_j}} \right] \\
&= \frac{1}{Z_Q} \left[\frac{\partial \Psi(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} - Q(\boldsymbol{c}) \cdot \frac{\partial Z_Q}{\partial \theta_{x_j|\mathbf{u}_j}} \right]
\end{aligned}$$

The first identity of the lemma follows from the following two identities:

$$\begin{aligned}
\frac{1}{Z_Q} \cdot \frac{\partial \Psi(\boldsymbol{c})}{\partial \theta_{x_j|\mathbf{u}_j}} &= \frac{Q(\boldsymbol{c}, x_j, \mathbf{pa}_j)}{\theta_{x_j|\mathbf{u}_j}} \\
\frac{1}{Z_Q} \cdot \frac{\partial Z_Q}{\partial \theta_{x_j|\mathbf{u}_j}} &= \frac{Q(x_j, \mathbf{pa}_j)}{\theta_{x_j|\mathbf{u}_j}}
\end{aligned}$$

The second identity is proved in a similar way. ∎

Corollary 3.2.1 *If Q is a chain graph over $\boldsymbol{T}$, then*

$$\begin{aligned}
\frac{\partial E_Q[f(\boldsymbol{C})]}{\partial \theta_{x_j|\mathbf{u}_j}} &= E_Q\left[\frac{\partial f(\boldsymbol{C})}{\partial \theta_{x_j|\mathbf{u}_j}}\right] + \frac{Q(x_j, \mathbf{u}_j)}{\theta_{x_j|\mathbf{u}_j}} \cdot \left(E_{Q(\cdot|x_j,\mathbf{u}_j)}[f(\boldsymbol{C})] - E_Q[f(\boldsymbol{C})]\right) \\
\frac{\partial E_Q[f(\boldsymbol{C})]}{\partial \psi_k(\boldsymbol{c}_k)} &= E_Q\left[\frac{\partial f(\boldsymbol{C})}{\partial \psi_k(\boldsymbol{c}_k)}\right] + \frac{Q(c_k)}{\psi_k(\boldsymbol{c}_k)} \cdot \left(E_{Q(\cdot|c_k)}[f(\boldsymbol{C})] - E_Q[f(\boldsymbol{C})]\right)
\end{aligned}$$

Note that when we differentiate $\mathcal{F}[Q]$ we get two terms. The first, is $\mathcal{F}[Q(\boldsymbol{T} \mid x_j, \mathbf{u}_j)]$ as before, and the other is $\mathcal{F}[Q]$. However, since $\mathcal{F}[Q]$ does not depend on the value of x_j, it is a absorbed in the normalizing constant $Z_{\mathbf{u}_j}$. Thus, the general structure of the solution remains similar to the simpler case of Bayesian networks:

$$\begin{aligned} Q(x_j \mid \mathbf{u}_j) &= \frac{1}{Z_{\mathbf{u}_j}} e^{\mathcal{E}_{CG}(x_j, \mathbf{u}_j)} \\ \psi_k(\boldsymbol{c}_k) &= \frac{1}{Z_{\boldsymbol{c}_k}} e^{\mathcal{E}_{CG}(c_k)} \end{aligned}$$

where

$$\begin{aligned} \mathcal{E}_{CG}(x_j, \mathbf{u}_j) &= \mathcal{F}[Q|x_j, \mathbf{u}_j] - \log Q(x_j, \mathbf{u}_j) + \log Q(x_j \mid \mathbf{u}_j) \\ \mathcal{E}_{CG}(\boldsymbol{c}_k) &= \mathcal{F}[Q|\boldsymbol{c}_k] - \log Q(\boldsymbol{c}_k) + \log \psi_k(\boldsymbol{c}_k) \end{aligned}$$

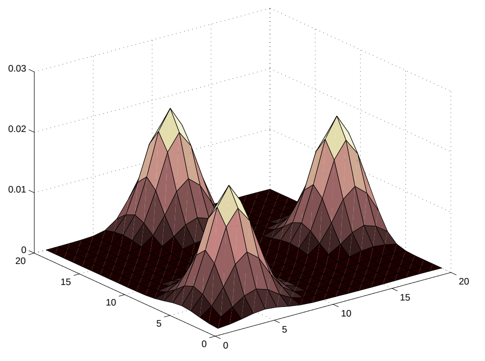
To get an explicit form of these equations, we simply write the chain-graph analogue of Lemma 3.1.1 which has similar form but includes additional terms. As in the case of Bayesian network, we can easily identify terms that can depend on the value of x_j, and focus the computation only on these. This is a straightforward extension of the ideas in Bayesian networks, and so we omit the details.

3.3 Adding hidden variables

Structured approximations were the first method proposed for improving the mean field approximation. Jaakkola and Jordan [17] proposed another direction for improvement of the mean field approximation: to use mixture distributions, where each mixture component is represented by a factorized distribution. The motivation for using mixture distribution emerges from the fact that in many cases the posterior distribution is multi-modal i.e. there are several distinct regions in the domain of the distribution with relatively high probability values. If the location of the different modes of the distribution depends on the values of several variables than the mean field approximation can not capture more than one mode. In this section we present an example that illustrates this limitation of the mean field approximation and the utility of mixture distributions approximations. Then we shall propose a generalize the mixture of mean field approximation to a novel structural approximation which contains *several extra* hidden variables.

3.3.1 The utility of mixture distributions approximations

As an example for the limitation of the mean field approximation, consider a probability distribution over two random variables X and Y, each having 20 values, whose joint distribution is represented in Figure 3-4 (a). The joint distribution of X and Y has three different modes located at $(X = 5, Y = 5)$, $(X = 18, Y = 10)$ and $(X = 10, Y = 18)$. Figure 3-4 (b) presents a possible mean field solution to that distribution. The solution depends on the location of the initial values of the mean field parameters in the update procedure. But no matter what this values are, the solution will always converge to a uni-modal distribution. Using the algorithm of Jaakkola and Jordan, an extremely high accurate solution to the distribution presented in Figure 3-4 (a) can be accomplished by an approximation that uses a mixture distribution with 3 mixture components and by an appropriate initialization of parameters.


(a)

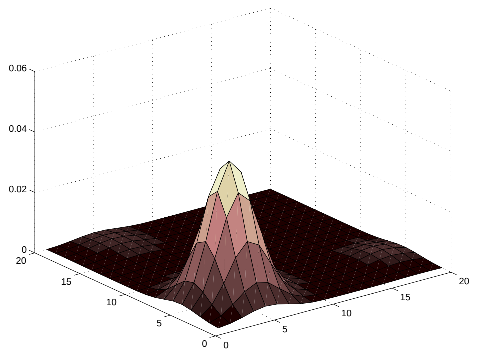

(b)

Figure 3-4: (a) A distribution defined over two random variables with 3 modes. (b) A possible mean field approximation. There is no solution that can capture all the modes.

We can view the mixture distribution approximation in the context of graphical models as an approximation the uses another *extra* hidden variable which is the parent node of all the variables. There are no other edges in the approximating distribution except for the edges that emerge from the extra hidden variable. In the example of Figure 3-4 (a) the original distribution could have been modeled by a Bayesian network over the variable X and Y where X is the parent node of Y. The mean field approximation uses a Bayesian network without edges. The mixture distribution approximation can be viewed as one that uses a Bayesian network over the variables X, Y and an extra variable V, where V is the parent node of X and Y.

The parameters of the mixture distribution could be found by maximizing the lower bound of the log likelihood presented in Equation 2.9. But using this technique in a straight-forward manner would not help us since the extra hidden variables introduces correlations, which leaves us with an optimization problem whose complexity is at least as great as this of the original inference problem. Jaakkola and Jordan overcame this problem by introducing another variational transformation resulting in another lower bound to the log likelihood [17]. In this section we will present the technique proposed by Jaakkola and Jordan and generalize it to augment the structured approximation by addition of extra hidden variables to the approximating distribution. This technique will enable us to improve the accuracy of the approximation while maintaining a reasonable model complexity.

3.3.2 The potential of extra hidden variables

Given the distribution $P(\boldsymbol{X})$ and evidence $\boldsymbol{o}$ we shall approximate the posterior $P(\boldsymbol{T} \mid \boldsymbol{o})$ with another distribution $Q(\boldsymbol{T})$. Now Q is defined over the variable set $\boldsymbol{T} \cup \boldsymbol{V}$ where $\boldsymbol{V}$ is a set of extra hidden variables. Our task is to find the parameters of Q that will maximize the lower bound $\mathcal{F}[Q]$ defined in equation 2.9. Due to the addition of extra hidden variables the lower bound becomes

$$\mathcal{F}[Q] \quad = \quad \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log P(\boldsymbol{t},\boldsymbol{o}) - \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t}) \qquad (3.15)$$

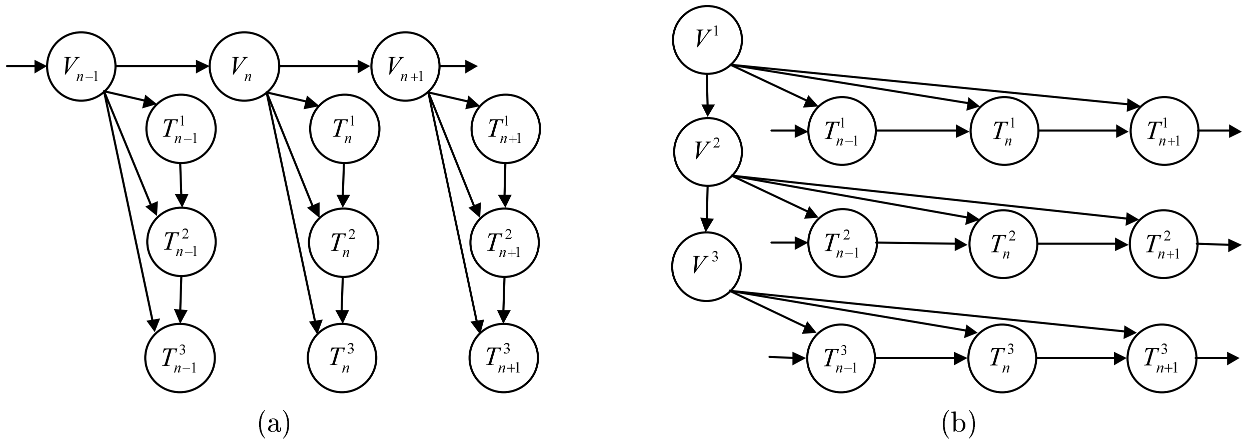


Figure 3-5: Approximating networks for the distribution represented by the network in Figure 2-6 with extra hidden variables. (a) Edges within a time slice are maintained. Correlations between time slice are modeled through the introduction of the hidden variable set $\{V_n\}_{n=1}^N$. (b) Edges between time slices are maintained. Correlations between the three chains are modeled through the hidden variables V^1, V^2 and V^3.

Note that in the last term the expression $\log Q(\boldsymbol{t})$ can not expressed as a sum of simpler terms since now $Q(\boldsymbol{t})$ is a sum over all the extra hidden values.

Before we show how to use the transformation that was used in [17], we shall gain some insight about the benefits of adding hidden variables by looking at an example and then by further analysis of the lower bound.

Figure 3-5 shows two examples of possible approximations for the distribution that is represented by the network in figure 2-6. Recall the structured approximation for this network modeled the approximating distribution by a network with three independent chains. In the networks presented here, the correlations are maintained through the hidden variables. In Figure 3-5(a) we added an extra hidden variable for every time slice. The correlations between time slices are maintained through those hidden variables. The edges within a time slice are maintained in order to preserve intra-time dependencies. In Figure 3-5(b) we maintained the edges between the time slices and added extra hidden variables for every chain. Correlations among the chains are maintained by the connections between the hidden variables.

Given the parameters of the approximating distribution, the computational complexity of inference depends on the domain size of the cliques in the clique-tree that is constructed. The introduction of extra hidden variables enables us to maintain many dependencies while reducing clique size. This is illustrated by the clique trees of the networks in our previous example. Figure 3-6(a) shows Portion of the clique tree of the original network. The corresponding portion of the clique tree of the approximating network in Figure 3-5(a) is shown in Figure 3-6(b). The cliques of the approximating network contain fewer variables. The trade-off between complexity and accuracy can be controlled by different choices of the domain size of the extra hidden variables $\{V_t\}_{t=1}^T$.

The cliques in Figure 3-6 illustrates one perspective of the potential of adding hidden variables. Given a structured approximation, the factored representation assumed by the approximation imposes independencies between variables that reside in different factors. Adding hidden variables allows us to correlate variables without the need to construct large factors.

Another perspective of the potential of extra hidden variable can be shown by reexamination of the lower bound $\mathcal{F}(Q)$. The analysis presented here is a straight forward generalization of the one presented in [17].

First we shall rewrite $\mathcal{F}(Q)$ using equation 3.15

$$\begin{aligned}
\mathcal{F}(Q) &= \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log P(\boldsymbol{t},\boldsymbol{o}) - \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t}) \\
&= \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log P(\boldsymbol{t},\boldsymbol{o}) - \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t} \mid \boldsymbol{v}) \\
&\quad + \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t} \mid \boldsymbol{v}) - \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t}) \\
&= \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{v}) Q(\boldsymbol{t} \mid \boldsymbol{v}) \log \frac{P(\boldsymbol{t},\boldsymbol{o})}{Q(\boldsymbol{t} \mid \boldsymbol{v})} + \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \log Q(\boldsymbol{t} \mid \boldsymbol{v}) - \sum_{\boldsymbol{t}} Q(\boldsymbol{t}) \log Q(\boldsymbol{t}) \\
&= \sum_{\boldsymbol{v}} Q(\boldsymbol{v}) \sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid \boldsymbol{v}) \log \frac{P(\boldsymbol{t},\boldsymbol{o})}{Q(\boldsymbol{t} \mid \boldsymbol{v})} + H(\boldsymbol{T}) - H(\boldsymbol{T} \mid \boldsymbol{V})
\end{aligned}$$

where $H(\boldsymbol{T})$ and $H(\boldsymbol{T} \mid \boldsymbol{V})$ are the entropy and conditional entropy defined in Section 2.4. Applying the definition of $\mathcal{F}$ on the conditioned distribution $Q(\boldsymbol{T}|\boldsymbol{v})$ for every value $\boldsymbol{v}$:

$$\mathcal{F}\left[Q(\boldsymbol{T})|\boldsymbol{v}\right] = Q(\boldsymbol{v}) \sum_{\boldsymbol{t}} Q(\boldsymbol{t} \mid \boldsymbol{v}) \log \frac{P(\boldsymbol{t},\boldsymbol{o})}{Q(\boldsymbol{t} \mid \boldsymbol{v})}$$

We get that $\mathcal{F}(Q(\boldsymbol{T}|\boldsymbol{v}))$ is a lower bound on $\log P(\boldsymbol{o})$. Using the last two equations and Equation 2.7 the lower bound becomes

$$\mathcal{F}(Q) = \sum_{\boldsymbol{v}} Q(\boldsymbol{v}) \mathcal{F}\left[Q(\boldsymbol{T})|\boldsymbol{v}\right] + I(\boldsymbol{T},\boldsymbol{V}) \tag{3.16}$$

The first term is an average on lower bounds that are gained without introducing extra hidden variables. The improvement arises from the second term. Given the structure of the an approximating network without extra hidden variables, the lower bound can be improved if there are several configurations of the parameters of the sub-network defined on $\boldsymbol{T}$ that achieve lower bounds that are near optimal. Using an extra hidden variable set to combine these configurations, will improve the lower bound by the amount of the mutual information between $\boldsymbol{T}$ and $\boldsymbol{V}$.

3.3.3 Relaxing the lower bound

This lower bound present in Equation 3.15 is harder to compute when there are additional hidden variables. Therefore we shall relax the lower bound. We start by rewriting $\mathcal{F}(Q)$ using Equation 2.7 for the mutual information:

$$\mathcal{F}(Q) = \sum_{\boldsymbol{v}} Q(\boldsymbol{v}) \mathcal{F}\left[Q(\boldsymbol{T})|\boldsymbol{v}\right] + H(\boldsymbol{V}) - H(\boldsymbol{V} \mid \boldsymbol{T})$$

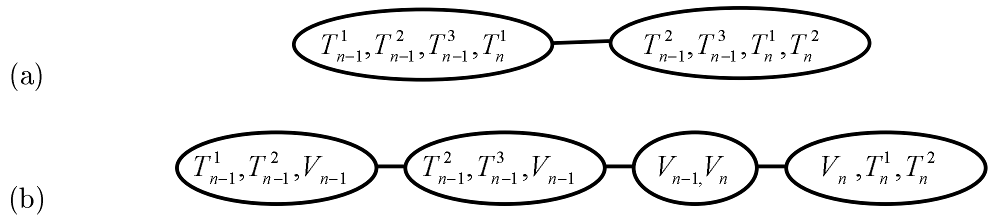

(a)
$T^1_{n-1}, T^2_{n-1}, T^3_{n-1}, T^1_n$
$T^2_{n-1}, T^3_{n-1}, T^1_n, T^2_n$
(b)
$T^1_{n-1}, T^2_{n-1}, V_{n-1}$
$T^2_{n-1}, T^3_{n-1}, V_{n-1}$
V_{n-1}, V_n
V_n, T^1_n, T^2_n

Figure 3-6: (a) A portion of the clique tree of the network presented in Figure 2-6. (b) The corresponding portion of the clique tree of the network defined in Figure 3-5(a). The cliques are smaller but dependencies are maintained through extra hidden variables. For instance, T^1_{n-1} and T^3_{n-1} are dependent given T^2_{n-1} because their cliques share the hidden variable V_{n-1}.

$$= \sum_{\boldsymbol{t}} Q(\boldsymbol{t}) \log P(\boldsymbol{t}, \boldsymbol{o}) + H(\boldsymbol{T}, \boldsymbol{V}) - H(\boldsymbol{V} \mid \boldsymbol{T})$$

The first and second terms are tractable if Q is tractable and if the conditional probability tables of P are of reasonable sizes. The remaining term is the *conditional entropy*,

$$\begin{aligned} -H(\boldsymbol{V} \mid \boldsymbol{T}) &= \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t}, \boldsymbol{v}) \log Q(\boldsymbol{v} \mid \boldsymbol{t}) \\ &= \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t}, \boldsymbol{v}) \log \frac{Q(\boldsymbol{t}, \boldsymbol{v})}{Q(\boldsymbol{t})} \end{aligned}$$

The conditional entropy is hard to compute and to derivate because now $\log Q(\boldsymbol{t})$ can not be decomposed into sums of simpler terms. Instead we can calculate a lower bound for $-H(\boldsymbol{V} \mid \boldsymbol{T})$ by introducing extra variational parameters. The new parameters will emerge from the convexity bound presented in equation 2.20

$$-\log(x) \geq -\lambda x + \log(\lambda) + 1 \tag{3.17}$$

Rewriting the conditional entropy

$$-H(\boldsymbol{V} \mid \boldsymbol{T}) = \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t}, \boldsymbol{v}) \left[-\log \frac{Q(\boldsymbol{t})}{Q(\boldsymbol{t}, \boldsymbol{v})} \right]$$

Applying Equation 3.17 for every term in the summation of the conditional entropy, we get a lower bound for the conditional entropy:

$$\begin{aligned} -H(\boldsymbol{V} \mid \boldsymbol{T}) &\geq \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t}, \boldsymbol{v}) \left[-R(\boldsymbol{t}, \boldsymbol{v}) \frac{Q(\boldsymbol{t})}{Q(\boldsymbol{t}, \boldsymbol{v})} + \log R(\boldsymbol{t}, \boldsymbol{v}) + 1 \right] \\ &= -\sum_{\boldsymbol{t},\boldsymbol{v}} R(\boldsymbol{t}, \boldsymbol{v}) Q(\boldsymbol{t}) + \sum_{\boldsymbol{t},\boldsymbol{v}} Q(\boldsymbol{t}, \boldsymbol{v}) \log R(\boldsymbol{t}, \boldsymbol{v}) + 1 \end{aligned}$$

Obviously, if we added a distinct variational parameter for every assignment $(\boldsymbol{t}, \boldsymbol{v})$, the conditional entropy could have been recovered accurately. But this setting leaves us with an intractable computation. In order to reduce the computational complexity of the lower bound, we assume that R has a similar structure to that of Q

$$R(\boldsymbol{t}, \boldsymbol{v}) = \prod_j \rho_{x_j, \mathbf{u}_j}$$

The new relaxed lower bound, denoted $\mathcal{G}[Q, R \mid \boldsymbol{c}]$ is

$$\mathcal{G}[Q, R \mid \boldsymbol{c}] = E_{Q(\boldsymbol{T},\boldsymbol{V}|\boldsymbol{c})}\left[\log \frac{P(\boldsymbol{T}, \boldsymbol{c}, \boldsymbol{o})}{Q(\boldsymbol{T}, \boldsymbol{V}|\boldsymbol{c})}\right] - \sum_{\boldsymbol{v}} E_{Q(\boldsymbol{T}|\boldsymbol{c})}\left[R(\boldsymbol{T}, \boldsymbol{v})\right] + E_{Q(\boldsymbol{T},\boldsymbol{V}|\boldsymbol{c})}\left[\log R(\boldsymbol{T}, \boldsymbol{V})\right] + 1 \tag{3.18}$$

To see the potential of the lower bound $\mathcal{G}[Q, R]$, recall that the improvement gained by adding hidden variables is the mutual information between the original set of hidden variables and the extra hidden variable. The mutual information is bounded below by

$$I(\boldsymbol{T}, \boldsymbol{V}) \geq H(\boldsymbol{V}) - \sum_{\boldsymbol{t}} E_{Q(\boldsymbol{t})}\left[R(\boldsymbol{T}, \boldsymbol{V})\right] + E_{Q(\boldsymbol{t},\boldsymbol{v})}\left[\log R(\boldsymbol{T}, \boldsymbol{V})\right] + 1$$

If we set $R(\boldsymbol{t}, \boldsymbol{v}) = Q(\boldsymbol{v})$ we get a trivial lower bound of zero. If there are no extra *observed* variables in Q and all the extra hidden variables R can be set in that manner. Therefore, performing further optimization can only improve the lower bound.

The optimization of the lower bound subject to the desired constraints is done via the definition of the following Lagrangian:

$$\mathcal{J}_H = \mathcal{G}[Q, R] - \sum_j \sum_{\boldsymbol{u}_j} \lambda_{\boldsymbol{u}_j} (\sum_{x_j} \theta_{x_j|\mathbf{u}_j} - 1)$$

Using Corollary 3.1.1, and then applying constraints, we get the typical update equations for $\theta_{x_i|\mathbf{u}_i}$:

$$Q(x_j \mid \mathbf{u}_j) = \frac{1}{Z_{\mathbf{u}_j}} \cdot e^{\mathcal{E}_H(x_j, \mathbf{u}_j)} \tag{3.19}$$

Where

$$\mathcal{E}_H(x_j, \mathbf{u}_j) = \mathcal{G}[Q, R \mid x_j, \mathbf{u}_j] - \log(\mathbf{u}_j)$$

As usual, we can decompose this term to a sum of terms:

$$\begin{aligned} \mathcal{E}_H(x_j, \mathbf{u}_j) \quad = \quad & \sum_i E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log \phi(\boldsymbol{D}_i, \boldsymbol{o})\right] - \sum_{j' \neq j} E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log Q(X_{j'} \mid \mathbf{U}_{j'})\right] \\ & + \sum_{j'} E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[\log R(X_{j'} \mid \mathbf{U}_{j'})\right] - E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[R(\boldsymbol{T})\right] \end{aligned}$$

the expression $\mathcal{E}$ is similar to the one obtained for the simpler structural approximation, except for the last two terms that ares from the lower bound on the negative conditional entropy. To evaluate the term $E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[R(\boldsymbol{T})\right]$ we write it as follows:

$$E_{Q(\cdot|x_j,\mathbf{u}_j)}\left[R(\boldsymbol{T})\right] \quad = \quad \frac{1}{Q(x_j, \mathbf{u}_j)} \cdot \sum_{\boldsymbol{t},\boldsymbol{v} \models x_j, \mathbf{u}_j} Q(\boldsymbol{t}, \boldsymbol{v}) R(\boldsymbol{t})$$

$$
\begin{aligned}
&= \frac{1}{Q(x_j,\mathbf{u}_j)} \cdot \sum_{\boldsymbol{t},\boldsymbol{v} \models x_j,\mathbf{u}_j} Q(\boldsymbol{t},\boldsymbol{v}) \sum_{\boldsymbol{v}'} R(\boldsymbol{t},\boldsymbol{v}') \\
&= \frac{1}{Q(x_j,\mathbf{u}_j)} \cdot \sum_{\boldsymbol{t},\boldsymbol{v} \models x_j,\mathbf{u}_j} \sum_{\boldsymbol{v}'} Q(\boldsymbol{t},\boldsymbol{v}) R(\boldsymbol{t},\boldsymbol{v}')
\end{aligned}
$$

This expression can be calculated by a variable-elimination like dynamic programming algorithm.

To complete the story, we need to consider the update equations for the parameters of R. First lets rewrite the lower bound on $\mathcal{G}[Q,R]$:

$$
\mathcal{G}[Q,R] \;=\; E_Q\left[\log \frac{P(\boldsymbol{T},\boldsymbol{o})}{Q(\boldsymbol{T},\boldsymbol{V})}\right] - \sum_{\boldsymbol{t},\boldsymbol{v},\boldsymbol{v}'} Q(\boldsymbol{t},\boldsymbol{v}) R(\boldsymbol{t},\boldsymbol{v}') + \sum_{j'} \sum_{x_{j'},\mathbf{u}_{j'}} Q(x_{j'},\mathbf{u}_{j'}) \log \rho_{x_{j'},\mathbf{u}_{j'}} + 1
$$

Differentiating with respect to $\rho_{x_j,\mathbf{u}_j}$ we get:

$$
\frac{\partial \mathcal{G}[Q,R]}{\partial \rho_{x_j,\mathbf{u}_j}} = - \sum_{\boldsymbol{t},\boldsymbol{v}' \models x_j,\mathbf{u}_j} \sum_{\boldsymbol{v}} Q(\boldsymbol{t},\boldsymbol{v}) \prod_{j' \neq j} \rho_{x_{j'},\mathbf{u}_{j'}} + \frac{Q(x_j,\mathbf{u}_j)}{\rho_{x_j,\mathbf{u}_j}}
$$

Note that the first term does not depend on $\rho_{x_j,\mathbf{u}_j}$. Equating the derivative to zero and rearranging the update equations become:

$$
\rho_{x_j,\mathbf{u}_j} = \frac{Q(x_j,\mathbf{u}_j)}{\sum_{\boldsymbol{t},\boldsymbol{v}' \models x_j,\mathbf{u}_j} Q(\boldsymbol{t}) \prod_{j' \neq j} \rho_{x_{j'},\mathbf{u}_{j'}}} \tag{3.20}
$$

Multiplying by $\rho_{x_j,\mathbf{u}_j}$ in the numerator and the denumerator the update equations can be written as:

$$
\rho_{x_j,\mathbf{u}_j} = \frac{\rho_{x_j,\mathbf{u}_j}}{\sum_{\boldsymbol{t},\boldsymbol{v} \models x_j,\mathbf{u}_j} R(\boldsymbol{t},\boldsymbol{v}) Q(\boldsymbol{t})} Q(x_j,\mathbf{u}_j) \tag{3.21}
$$

Again, we can efficiently compute such equations using dynamic programming.

We note that the Lagrangian is a convex function of both $\theta_{x_j|\mathbf{u}_j}$ and $\rho_{x_j,\mathbf{u}_j}$, and thus asynchronous iterations of these equations improve the lower bound and will eventually converge to a stationary point.

3.4 Experiments on Dynamic Bayesian Networks

To evaluate our methods we performed a preliminary test with synthetic data. We created dynamic Bayesian networks with the general architecture shown in Figure 2-6. All the variables in these networks are binary. We controlled two parameters: the number of time slices expanded, and the number of variables in each slice. The parameters of networks were sampled from a Dirichlet prior with hyper-parameter $\frac{1}{2}$. Thus, there was some bias toward skewed distributions. Our aim was to compute the likelihood of the observation in which all observed variables were set to be 0. We repeated these tests for sets of 20 networks sampled for each combination of the two parameters (number of time slices and number of variables per slice).

We performed variational approximation to the posterior distribution using three types of networks with hidden variables: The first two types are based on the "vertical" and "horizontal" architectures shown in Figure 3-5(a) and (b). We considered networks with 1, 2, and 3 values for the hidden variable. (Note that when we consider a hidden variable with one value, we essentially apply the Bayesian network structured approximation.) The third type are networks that represent mixture of mean field approximations. For this type we considered networks with 1, 4 and 6 mixture components (When there is one mixture component the approximation is simply mean field). We run each procedure for 10 iterations of asynchronous updates. This seems to converge on most runs. To avoid local maxima, we tried 10 different random starting points in each run and returned the best scoring one.

The figure of merit for our approximations is the reported upper-bound on the KL-divergence between the approximation and the true posterior. This is simply $\log P(\boldsymbol{o}) - \mathcal{G}_Q[Q, R]$. (The examples are sufficiently small, so that we can compute $\log P(\boldsymbol{o})$.) We need to examine this quantity since different random networks have different values of $P(\boldsymbol{o})$ and so we cannot compare lower bounds.

Figure 3-7 describes the results of these runs. As we can see the differences grow with the number of time slices. This is expected as the problem becomes harder with additional slices. The general trend we see is that runs with more hidden values perform better. These differences are mostly pronounced in the larger networks. This is probably due to the higher complexity of these networks.

The comparison to mixture of mean fields approximation shows that simple mean field (1 component) is much worse than all the other methods. Second, we see that although mixtures of mean field improve with larger number of components, they are still worse than the structured approximations on the network with 3 variables per slices. We believe that these toy examples are not sufficiently large to highlight the differences between the different methods. For example, differences start to emerge when we examine 6 and 7 time slices.

Our implementation of these variational methods is not optimized and thus we do not believe that running times are informative on these small examples. Nonetheless, we note that running mixtures of mean fields with 6 components took roughly the same time as running the structured approximations with hidden variables of cardinality 3.

One caveat of this experiment is that it is based on random networks, for which the dependencies between variables is often quite weak. As such it is hard to gauge how hard is inference in this networks. We are currently starting to apply these methods to real-life problems, where we expect to improvement over mean field type methods to be more pronounced.

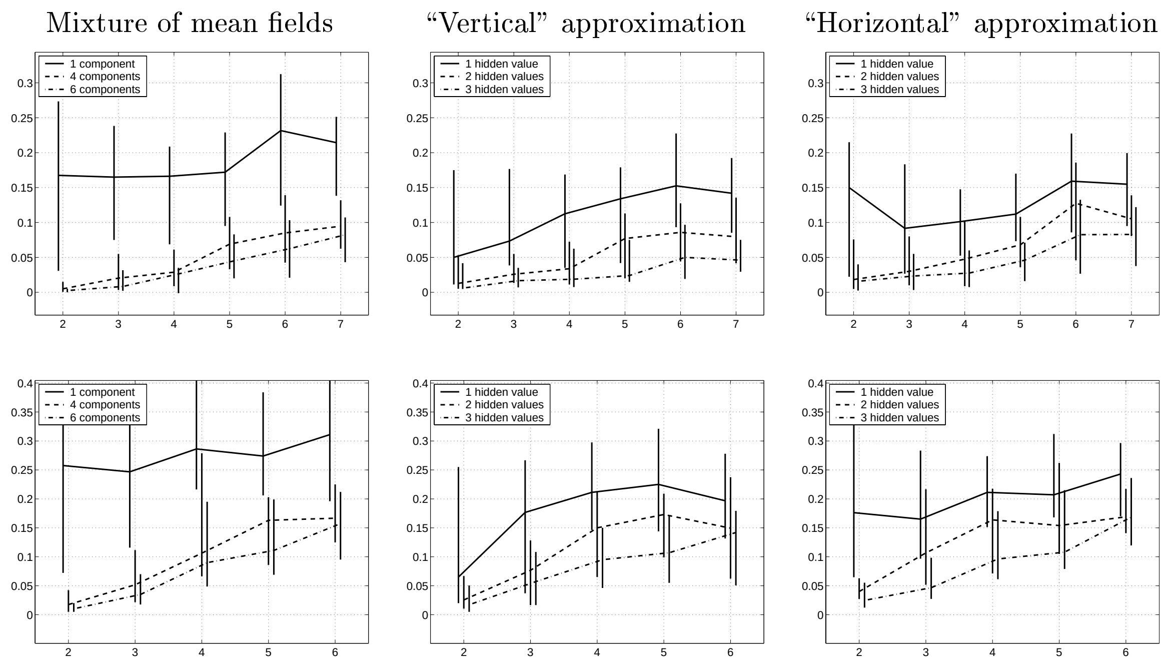


Figure 3-7: Comparison of the two approximating structures of Figure 3-5 and mixture of mean fields. The figures on the left column report results for the mixture of mean fields approximation, with 1, 4, and 6 mixture components. The figures on the middle column report results for the network structure containing additional hidden variable for each time slice (Figure 3-5(a)) with hidden variables with 1, 2, and 3 values. The figures on the right column report results for the network structure containing additional hidden variable for each temporal chain (Figure 3-5(b)) with hidden variables with 1, 2, and 3 values. The figures on the top row report on approximation to networks with 3 variables per time slice and the figures on the bottom row report on networks with 4 variables per time slice. The x-axis corresponds to the number of time slices in the network. The y-axis corresponds to the upper-bound on the KL-divergence $\log P(\boldsymbol{o}) - \mathcal{G}_Q[Q, R]$ normalized by the number of time slices in the network. Lines describe to median performance among 20 inference problems, and error bars describe 25-75 percentiles.

Chapter 4

Efficient marginalization

The variational approximations presented in the previous chapter utilize exact inference as a subroutine. An iterative procedure updates the parameters of the approximating distribution using expectations of functions of several random variables. In order to compute these expectations efficiently, the marginal probabilities are computed using the junction tree algorithm presented in Section 2.2.2. Some of these expectations are of functions whose domain is the one of cliques in the true distribution. These cliques may not be always contained in cliques of the junction tree of the approximating distribution. The junction tree algorithm provides only joint distributions of sets of random variables that are contained within a clique. Therefore, Additional computations are required to compute joint distributions of sets that are dispersed in several cliques.

The problem of computing marginal probabilities of sets of random variables that are dispersed within several cliques was addressed by Xu [31], who proposed an algorithm that uses a series of clique mergings. This procedure results in the creation of intermediate factors. The required marginal probability is computed from the last factor that is created by this procedure.

The size of intermediate factors used by the clique merging procedure depends on the order of operations that are performed by it. This has a direct effect on the complexity of the procedure. In this chapter we address the task of finding a sequence of operations which will result in efficient computation of the marginal probabilities. Section 4.1 presents the algorithm for computing marginal distributions of set of random variables that are dispersed within several cliques. Section 4.2 presents a dynamic programming algorithm to compute an optimal operation sequence. In some cases even the dynamic programming algorithm might be complex, therefore, Section 4.3 proposes heuristic search techniques to overcome this problem. Section 4.4 presents experimental results using these techniques. Finally, Section 4.5 presents proofs for the claims introduced in this chapter.

4.1 Computing marginals in Junction Trees

In this section we will present marginalization algorithms based on [31]. The first stage in computing marginals over a set of variables $\boldsymbol{U}$ is to remove all clique potentials that are not required for the computation. In the second stage, $P(\boldsymbol{U})$ is calculated using the operations discussed in the previous section. We will present an algorithm for trimming the junction tree designed to avoid unnecessary computations. Then, we will show how $P(\boldsymbol{U})$ is calculated from the trimmed tree by variable elimination. Finally, we will discuss a variable elimination version with the constraint of using local computations. In this approach, multiplication is performed only among potentials belonging to adjacent cliques. This restriction will allow us to design optimization algorithms based on the structure of the junction tree.

4.1.1 The minimal subset of clique potentials required for the computation

Recall that junction trees represent posterior probabilities over the set of variables $\boldsymbol{X}$ on which they are defined. Given evidence $\boldsymbol{e}$, the factored representation is

$$P(\boldsymbol{x} \mid \boldsymbol{e}) = \frac{\prod_{i \in \mathcal{I}} P(\boldsymbol{c}_i \mid \boldsymbol{e})}{\prod_{(i,j) \in \mathcal{E}} P(\boldsymbol{s}_{(i,j)} \mid \boldsymbol{e})} = \frac{\prod_{i \in \mathcal{I}} \phi_i(\boldsymbol{c}_i)}{\prod_{(i,j) \in \mathcal{E}} \phi_{(i,j)}(\boldsymbol{s}_{(i,j)})} \tag{4.1}$$

where $\boldsymbol{C}_i$ and $\boldsymbol{S}_{(i,j)}$ are cliques and separators respectively.

Equation 4.1 is true for any subtree of the original junction tree. We can use it to calculate the conditional joint distribution of a subset $\boldsymbol{U}$ by using a subtree whose cliques contain all the variables in $\boldsymbol{U}$. Obviously if $\mathcal{T}_1 \subset \mathcal{T}_2$, any optimal computation scheme will be faster on $\mathcal{T}_1$.Xu [31] showed that if $\boldsymbol{U}$ is not contained in a single clique, there is a unique minimal subtree which contains $\boldsymbol{U}$, and proposed a way to find it.

The minimal subtree is found by an iterative algorithm. On each iteration, the algorithm maintains a subtree $\mathcal{T}'$ of the original clique tree containing $\boldsymbol{U}$. If there is a leaf k whose removal would will result in a new subtree which is also containing $\boldsymbol{U}$, that leaf is removed. Thus, a leaf k can be removed if $\boldsymbol{U} \subseteq \cup_{i \in \mathcal{I}' \setminus \{k\}} \boldsymbol{C}_i$.

Actually, we can use a simpler condition for removing a leaf. A leaf k can be removed if $(\boldsymbol{C}_k \cap \boldsymbol{U}) \subseteq \boldsymbol{S}_k$. Trimming such a leaf maintains the property of containing $\boldsymbol{U}$ because the variables of the separator are contained in the neighbor clique which stay in the trimmed subtree.

The following procedure finds a minimal subtree which contains a set of variables $\boldsymbol{U}$,

Minimal-Subtree Algorithm

```
Set $\mathcal{T}' = \mathcal{T}$
do for each leaf $k$ in $\mathcal{T}'$:
    Let $\boldsymbol{S}_k$ be the separator connecting $\boldsymbol{C}_k$ to the rest of $\mathcal{T}'$.
    if $(\boldsymbol{C}_k \cap \boldsymbol{U}) \subseteq \boldsymbol{S}_k$
        remove $k$ and the edge connected to $k$ from $\mathcal{T}'$.
until there are no more leafs to remove.
// Remove unnecessary variables from the cliques.
For each clique $\boldsymbol{C}_i$ in $\mathcal{T}'$ do
    Marginalize $\phi_i$ from $\boldsymbol{C}_i$ to the set $(\boldsymbol{U} \bigcup \cup_{(i,j) \in \mathcal{E}} \boldsymbol{S}_{(i,j)}) \cap \boldsymbol{C}_i$.
```

The trimmed subtree is minimal because when there are no leafs that can be trimmed, every leaf has a variable from $\boldsymbol{U}$ which is not in the neighbor separator. From the running intersection property it follows that this variable can not be contained in the rest of tree.

The uniqueness of the minimal subtree is also a consequence of the running intersection property. Suppose there is no single clique containing $\boldsymbol{U}$ and there are two minimal subtrees. If they are disjoint, then the running intersection property implies that every clique in the path between them contains $\boldsymbol{U}$, contradicting our assumption. If they are not disjoint, then again from the running intersection property it follows that their intersection contains $\boldsymbol{U}$, contradicting the the fact that these subtrees are minimal.

4.1.2 Marginalization by variable elimination

After trimming the junction tree we can perform the computation of the probability table of $\boldsymbol{U}$ by using the variable elimination operation introduced in section 2.2.1. This procedure eliminates variables that are not contained in $\boldsymbol{U}$ whenever it is possible. On each iteration the procedure chooses a variable $X_k \notin \boldsymbol{U}$ and multiplies all the potentials containing it. Then X_k is marginalized out from the resulting potential. The following pseudo-code summarizes this procedure,

Variable Elimination Procedure

```
While there are variables in the junction tree that are not in $\boldsymbol{U}$.
    Choose a variable $X_k \notin \boldsymbol{U}$.
    Let $\hat{\mathcal{I}} = \{i | X_k \in \boldsymbol{C}_i\}$.
    Let $\hat{\mathcal{E}} = \{(i,j) | X_k \in \boldsymbol{S}_{(i,j)}\}$.
    Define a new potential:
        $$\psi = \frac{\prod_{i \in \hat{\mathcal{I}}} \phi_i}{\prod_{(i,j) \in \hat{\mathcal{E}}} \phi_{(i,j)}}$$
    Marginalize out $X_k$ and any other variable that appears only on $\psi$. Denote the resulting potential as $\psi'$.
    Replace the sets of potentials $\{\phi_i\}_{i \in \hat{\mathcal{I}}}$ and $\{\phi_{(i,j)}\}_{(i,j) \in \hat{\mathcal{E}}}$ by $\psi'$.
```

The complexity of our computation is determined by the order of variables we choose to eliminate. Intuitively, we would like to choose the order of operations in such a way that the potentials created during the intermediate stages of the computation will be of minimal size. In the next sections we will describe a dynamic programming and a search procedure that finds an operation sequence leading to a minimal size intermediate potentials. Here we will introduce a greedy algorithm for this problem.

The greedy algorithm we will use resembles the algorithm used for finding a good triangulation for a moral graph when constructing a junction tree [20, 13]. As mentioned earlier, in order to eliminate a variable X_k, All the factors containing X_k should be multiplied. This results in a factor containing all the variables that were previously in the same factor as X_k. Suppose we look now at the undirected graph $\mathcal{G}_{\mathcal{T}}$ that represents adjacency relations in the junction tree $\mathcal{T}$. Specifically, $\mathcal{G}_{\mathcal{T}}$ contains a node for every variable present in $\mathcal{T}$ and an edge between every pair of variables which are contained in the same clique. We shall define the *weight* of the node X_k in $G_{\mathcal{T}}$ as the product of the domain sizes of X_k and his neighbors. The size of the intermediate factor constructed in order to eliminate X_k equals to the weight of X_k in $G_{\mathcal{T}}$. Also, the elimination of X_k creates a new factor containing all the variables corresponding to the neighbors of X_k in $G_{\mathcal{T}}$. This formulation leads to a the greedy algorithm operating on $G_{\mathcal{T}}$. The algorithm we present creates a queue with the indices of the variables and calculates the sum of the intermediate clique potential size in a variable h',

Greedy Elimination Order Algorithm

```
Set $h' = 0$.
While $G_{\mathcal{T}}$ contains variable not in $\boldsymbol{U}$,
    Choose a node $X_k$ according to some greedy criterion.
    Insert $k$ to a first in first out queue.
    Let $w$ be the weight of $X_k$.
    Set $h' = h' + w$.
    Connect all the neighbors of $X_k$
    Remove $X_k$ and its edges from the graph
Return $h'$ and the queue.
```

The most obvious greedy criterion is to choose the node with the minimal weight. Uffe Kjærulff [20] and Huang and Darwiche [13] purposed a slightly more intricate criterion, The node being chosen is the one that causes the least number of edges to be added to $G_{\mathcal{T}}$. In case of a tie, take the node with the least weight.

4.1.3 Using local computations

In order to design efficient algorithms for finding a good sequence of operations for marginalization we will impose a constraint on the variable elimination procedure. Multiplication and division of potentials will be performed using local computations. On each stage of the computation, A clique potential will be divided by an adjacent operator potential and the resulting potential will be multiplied by the other potential of the clique attached to that separator. We will call this operation *clique merging*. This operation results in a new clique and a potential defined on this clique. The new set of cliques maintains a clique tree structure. In the next section we will argue that the total size of intermediate potentials created by an optimal constrained computation can not be greater the twice the total size of intermediate potentials created by an unconstrained computation.

The computation of the marginal distribution will be performed by a series clique mergings. We will define this operation formally and then we will show how to use it for marginalization.

Definition 4.1.1 *Let $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ be two adjacent cliques. The merging of $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ is a clique $\boldsymbol{C} = \boldsymbol{C}_i \cup \boldsymbol{C}_j$ and a potential ϕ_C on $\boldsymbol{C}$ defined as,*

$$\phi_C = \frac{\phi_i \cdot \phi_j}{\phi_{(i,j)}}$$

Variable	Domain size
X_2	2
X_3	4
X_4	2
X_5	2
X_6	2
X_7	3
X_8	4
X_9	4
X_{10}	4

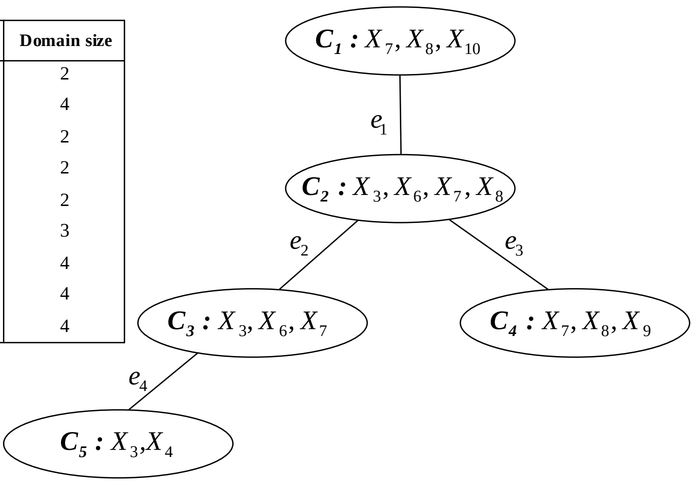


Figure 4-1: A trimmed clique tree ready for computing the probability table of $\boldsymbol{U} = \{X_4, X_9, X_{10}\}$.

The potential resulting from the clique merging operation equals the joint distribution of the clique variables. Consequently we can modify our Junction Tree so it will contain the merged clique instead of the two original cliques and their potentials. We do this by removing the nodes i and j and the edge (i, j) from $\mathcal{T}$, inserting a new node k and replacing all the edges connecting i and j to their neighbors by edges between k and these neighbors. Then we assign the merging of $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ to $\boldsymbol{C}_k$ and ϕ_k. We note that the separators and potentials corresponding to the edges connected to k are the same as those who correspond to the edges connecting i and j to their neighbors. Thus after removing an edge, $\mathcal{T}$ maintains the Junction tree requirements.

After removing all the edges from a Junction tree we end up with a single clique and a potential on its variables. In order to compute marginals efficiently we eliminate every possible variable after each edge removal. We can eliminate every variable that is not contained in one of the separators of the tree and is not contained in the query set U. Now we can define the procedure for finding marginals.

Marginalization Algorithm With Local Computations

```
For each clique $\boldsymbol{C}_i$ do
    Marginalize $\phi_i$ from $\boldsymbol{C}_i$ to the set $(\boldsymbol{U} \bigcup \cup_{(i,j)\in\mathcal{E}} \boldsymbol{S}_{(i,j)}) \cap \boldsymbol{C}_i$.
While there are no cliques containing $\boldsymbol{U}$:
    Pick an edge e=(i,j).
    Remove the edge (i,j) from $\mathcal{T}$.
    Denote the resulting clique and potential by $\boldsymbol{C}_k$ and $\phi_k$.
    Marginalize $\phi_k$ from $\boldsymbol{C}_k$ to the set $(\boldsymbol{U} \bigcup \cup_{(k,l)\in\mathcal{E}} \boldsymbol{S}_{(k,l)}) \cap \boldsymbol{C}_k$.
```

We shall illustrate the marginalization algorithm using the trimmed clique tree presented in Figure 4-1. Suppose our query variables are $\boldsymbol{U} = \{X_4, X_9, X_{10}\}$. Figure 4-1 shows the clique tree after marginalizing every possible variable that is not needed for the query. After the local marginalizations stage the algorithm proceeds with the following stages:

- Removing edge e_1. This results in a new clique containing the variables X_3, X_6, X_7, X_8 and X_{10}.
- Removing edge e_4 gives us a clique containing X_3, X_4, X_6, X_7.
- Removing edge e_3 causes the merging of the clique that was constructed after the first stage with a leaf clique. The new clique contains X_3, X_6, X_7, X_8, X_9 and X_{10}. Now X_8 is marginalized out from this clique.
- The last edge being removed is e_2. The final clique contains X_3, X_4, X_6, X_7, X_9 and X_{10}. This clique is marginalized, leaving us with the probability table of $\boldsymbol{U}$.

The clique merging procedure involves multiplication and division of potentials. The number of multiplications and divisions required for merging two cliques, is equal to the size of the domain of the resulting clique. This number is exponential in the size of the merged clique. The size of the intermediate cliques formed during the marginalization depends on the order of edge removals.

Getting back to the example in figure 4-1, we can see that scheme described previously required 2736 multiplications. Changing the order of edge removals to (e_2, e_4, e_1, e_3) enables us to answer the query using 672 multiplications. In the next sections we shall confront the problem of finding an optimal edge removal order.

4.2 The optimal operation sequence

The algorithm for finding the fastest way to marginalize consists of two stages. First, we trim the tree from potentials that are unnecessary for the query. In the second stage we find the optimal order of potential mergings using a dynamic programming scheme.

The local computation constraint imposed on the marginalization algorithm enables us to exploit the tree structure for designing a dynamic programming optimization algorithm. Before we describe the algorithm we will check how this constraint influences the optimality of the result.

4.2.1 The effect of the local computation constraint on the complexity of the marginalization algorithm

The constrained marginalization algorithm may not be optimal compared to an unconstrained algorithm, in which we are allowed to perform multiplications divisions and marginalizations of potentials in any order consistent with the variable elimination procedure . However, as discussed in the previous section, the running intersection property of junction trees suggests that this constraint is a reasonable one, since variable elimination requires computations on contiguous subtrees of the junction tree. In order to support this intuitive argument, we shall show that the total size of intermediate potentials created by the constrained version can not be too large compared to the unconstrained algorithm.

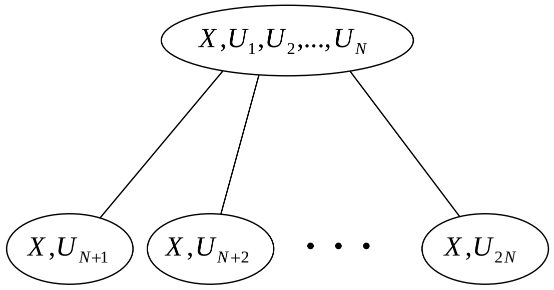


Figure 4-2: A clique tree in which elimination of X by edge removals involves intermediate cliques with total size almost twice the total size of intermediate cliques obtained with an unconstrained marginalization.

Let $\mathcal{T}$ be a junction tree. Denote by $\boldsymbol{V}$ the variables in $\mathcal{T}$, i.e. $\boldsymbol{V} = \cup_{i\in\mathcal{I}}\boldsymbol{C}_i$. As before, $\boldsymbol{U} \subset \boldsymbol{V}$ is the set of query variables. We shall denote by $\mathcal{A}(\mathcal{T}, \boldsymbol{U})$ a sequence of operations which calculates $\psi = P(\boldsymbol{U})$ defined by the junction tree. The operations used in $\mathcal{A}$ are multiplication, division and marginalization of potentials. Let $\phi_1^{\mathcal{A}}, ..., \phi_K^{\mathcal{A}}$ be the sequence of intermediate potentials created after each multiplication performed by $\mathcal{A}$. We shall denote by $S(\mathcal{A}(\mathcal{T}, \boldsymbol{U}))$, the total size of the intermediate potentials, $S(\mathcal{A}(\mathcal{T}, \boldsymbol{U})) = \sum_i |dom(\phi_i^{\mathcal{A}})|$. We shall use the notation $S(\mathcal{A})$, when $\mathcal{T}$ and $\boldsymbol{U}$ are clear from the context. The following claim states that for every arbitrary sequence $\mathcal{A}$ there is a sequence $\mathcal{A}'$ of edge removals such that $S(\mathcal{A}') \leq 2S(\mathcal{A})$,

claim 4.2.1 *Let $\mathcal{T}$ be a junction tree. Let $\boldsymbol{U}$ be a subset of $\boldsymbol{V}$. Suppose $\mathcal{A}(\mathcal{T}, \boldsymbol{U})$ is a sequence of operations computing $P(\boldsymbol{U})$. Then, there exists a sequence of local clique mergings $\mathcal{A}'(\mathcal{T}, \boldsymbol{U})$ such that,*

$$S(\mathcal{A}') \leq 2 \cdot S(\mathcal{A})$$

Proof: See Section 4.5 ▮

The bound on $S(\mathcal{A}')$ is tight. We shall illustrate this using the clique tree presented in figure 4-2. Suppose we want to obtain the probability table of $U_1, ..., U_{2N}$, assuming all the variables in the tree are binary. We need to multiply all the cliques in the tree and eliminate X. Multiplication can be performed by a sequence $\mathcal{A}'$ of edge removals. $\mathcal{A}'$ generates a sequence of N intermediate potentials $\phi_1^{\mathcal{A}'}, ..., \phi_N^{\mathcal{A}'}$. For each $1 \leq i \leq N$ the domain of $\phi_i^{\mathcal{A}'}$ is the domain of $\{X, U_1, ..., U_{N+i}\}$ having size 2^{N+1+i}. Thus the total size of intermediate cliques will be,

$$S(\mathcal{A}') = \sum_{i=1}^{N} 2^{N+1+i} \approx 2^{2N+2} \tag{4.2}$$

Ignoring the local computation requirement, marginalization can be done by multiplying the small clique potentials prior to the big one. This way, for $1 \leq i \leq N-1$, the intermediate potential $\phi_i^{\mathcal{A}}$ will have the domain of $\{X, U_{N+1}, ..., U_{N+i+1}\}$ with size 2^{i+2}. $\phi_N^{\mathcal{A}}$ will over all the variables, giving us a total of,

$$S(\mathcal{A}) = \sum_{i=1}^{N-1} 2^{i+2} + 2^{2N+1} \approx 2^{2N+1} \tag{4.3}$$

Combining 4.2 and 4.3, we get,

$$S(\mathcal{A}') \approx 2S(\mathcal{A})$$

4.2.2 The optimal order of clique merging

Once we found the minimal subtree required for the computation, we want to find out the optimal order of edge removals. Note that every intermediate clique created in the marginalization algorithm is a result of merging all the cliques of some subtree of $\mathcal{T}'$ and marginalizing it as much as possible. At each intermediate stage we can marginalize the subtree $\mathcal{T}'$ to the subset $\boldsymbol{C}_{\mathcal{T}'}$ of all the variables that are contained in $\mathcal{T}'$ and belong to $\boldsymbol{U}$ or to one of the separators outside $\mathcal{T}'$. $\boldsymbol{C}_{\mathcal{T}'}$ can be written as,

$$\boldsymbol{C}_{\mathcal{T}'} = (\boldsymbol{U} \cup \bigcup_{(i,j)\notin\mathcal{E}'} \boldsymbol{S}_{ij}) \cap (\bigcup_{i\in\mathcal{I}'} \boldsymbol{C}_i) \tag{4.4}$$

When we reach the last stage of the marginalization algorithm we end up with a tree consisting of two nodes. Now we should remove the edge connecting them and marginalize the resulting clique to $\boldsymbol{U}$. The cliques corresponding to the last two nodes were gained by merging the two subtrees adjacent to the edge connecting them. Those cliques are marginalized to the subset which consists variables in $\boldsymbol{U}$ and the separator between them.

Following the last observation, the marginalization algorithm in section 4.1.3 can be formalized in a recursive way. The algorithm will pick an edge (i, j). It will recursively marginalize the two subtrees emanating from (i, j) to $\boldsymbol{C}_{\mathcal{T}_{i\to j}}$ and $\boldsymbol{C}_{\mathcal{T}_{j\to i}}$, where $\mathcal{T}_{j\to i}$ and $\mathcal{T}_{i\to j}$ are the subtrees emanating from the nodes i and j of (i, j) respectively. After that, the last edge will be removed from the resulting tree and we will end up with a clique $\boldsymbol{C}$ containing $\boldsymbol{U}$. Finally, $\boldsymbol{C}$ will be marginalized to $\boldsymbol{U}$.

This formulation suggests a recursive algorithm for finding an optimal order of edge removals. Let us denote the minimal cost for merging all the cliques in $\mathcal{T}'$ and marginalizing it to $\boldsymbol{C}_{\mathcal{T}'}$ by $c(\mathcal{T}')$. The last edge that is removed by the optimal algorithm will be denoted by $\mathrm{l}(\mathcal{T}')$. The total cost of marginalizing $\mathcal{T}'$ to $\boldsymbol{C}_{\mathcal{T}'}$. Will be,

$$c(\mathcal{T}') = min_{(i,j)\in\mathcal{E}'}\{c(\mathcal{T}'_{i\to j}) + c(\mathcal{T}'_{j\to i}) + d(\boldsymbol{C}_{\mathcal{T}'_{i\to j}}, \boldsymbol{C}_{\mathcal{T}'_{j\to i}})\} \tag{4.5}$$

The last edge to be removed is,

$$l(\mathcal{T}') = argmin_{(i,j)\in\mathcal{E}'}\{c(\mathcal{T}'_{i\to j}) + c(\mathcal{T}'_{j\to i}) + d(\boldsymbol{C}_{\mathcal{T}'_{i\to j}}, \boldsymbol{C}_{\mathcal{T}'_{j\to i}})\} \tag{4.6}$$

Where $d(\boldsymbol{C}_k, \boldsymbol{C}_l)$ is the cost of merging and marginalizing the two cliques $\boldsymbol{C}_k$ and $\boldsymbol{C}_l$.

The recursive algorithm is inefficient. The cost of the optimal order of edge removals of each subtree $\mathcal{T}'$ is computed more then once, since every subtree $\mathcal{T}''$ containing $\mathcal{T}'$ will call the procedure for computing the cost of $\mathcal{T}'$. The complexity of the recursive algorithm is,

$$T(n) = 2 \cdot \sum_{i=1}^{n-1} T(i) + \mathcal{O}(n) \tag{4.7}$$

Which, as shown in , is exponential in n. The repeated computations property of this recursive algorithm suggests that we can improve it by a dynamic programming scheme avoiding re computations.

The dynamic programming algorithm will use a bottom up approach. It will fill a table $c(\mathcal{T}')$ which will store the minimal cost for constructing every subtree of $\mathcal{T}$ and a table $l(\mathcal{T}')$ of the last edge being removed in an optimal algorithm. First it will set the optimal cost of creating all the subtrees having only one node to be zero. Then the algorithm will iterate from size 2 to size $|\mathcal{T}|$. On each iteration the algorithm goes over all the subtrees with m nodes. For each subtree it finds the optimal cost for marginalizing it and the last edge that is removed in order to do so according to equations 4.5 and 4.6. In order to check all the possibilities of constructing all the subtrees of size m we iterate on all the edges. For each edge we look, for each $1 \leq l \leq m-1$, on the cost of merging subtrees of size l emanating from one node of the edge with subtrees of size $m-l$ emanating from the other node of the edge. This gives us the following algorithm,

Optimal-Order algorithm

```
Do for each subtree $\mathcal{T}'$ with size 1
    Set the c($\mathcal{T}'$) to be 0.
For $m = 2$ to $|\mathcal{T}|$
    Do on all edges $(i, j)$ in $\mathcal{T}$
        Update-Edge($(i, j), m$)
```

Procedure **Update-Edge** will calculate the minimal cost of constructing all the subtrees whose size is m, with the constraint that the last edge being removed is (i, j),

Procedure Update-Edge

```
For $l = 1$ to $m - 1$
    Do on all subtrees $\mathcal{T}_1$ emanating from $j \to i$ and having size $l$
        Do on all subtrees $\mathcal{T}_2$ connected to $< i \to j$ having size $m - l$
            Set $\mathcal{T}'$ to be the union of $\mathcal{T}_1$ and $\mathcal{T}_2$
            Set $c'(\mathcal{T}') = c(\mathcal{T}_1) + c(\mathcal{T}_2) + c(\boldsymbol{C}_{\mathcal{T}_1}, \boldsymbol{C}_{\mathcal{T}_2})$
            If $c'(\mathcal{T}') < c(\mathcal{T}')$ set $c(\mathcal{T}') = c'(\mathcal{T}')$ and $l(\mathcal{T}') = (i, j)$
```

The reconstruction of the optimal order is done recursively. The last edge is given by $l(\mathcal{T})$. Then we find the order of edge removals for marginalizing the two subtrees connected to that edge.

The implementation of this algorithm requires maintaining and accessing the two tables $c(\mathcal{T}')$ and $l(\mathcal{T}')$ whose entries are subtrees. This requires a hashing function that maps every subtree to an index. We can do it by mapping every subtree $\mathcal{T}'$ to a string $h_{\mathcal{T}'}$ of $|\mathcal{T}|$ bits. Every bit will be indexed with the index of some node in $\mathcal{T}$. For each $i \in \mathcal{I}$, $h_{\mathcal{T}'}(i)$ will be equal 1 if $i \in \mathcal{I}'$ and 0 otherwise. The tables will be stored as balanced binary search trees. The size of these search trees will be $\mathcal{O}(S(\mathcal{T}))$, where $S(\mathcal{T})$ is the number of subtrees of $\mathcal{T}$. The access and update times for these tables are $\mathcal{O}(logS(\mathcal{T}))$. Procedure **Update-Edge** requires a fast access to all the subtrees connected to both sides of the edge. For every side $i \to j$ and $j \to i$ of an edge (i, j) we will keep, for every size $1 \leq m \leq |\mathcal{T}| - 1$, a list of the indices of all the subtrees with size m emanating from this side. For every subtree with d leafs, it's index will stored d times in this data structure.

We can prove the correctness of the algorithm by induction on the size of the subtrees. For each tree $\mathcal{T}'$ the algorithm finds the minimal cost procedure for marginalizing the subtree to $\boldsymbol{C}_{\mathcal{T}'}$. Thus, it finds the minimal cost procedure for marginalizing to $\boldsymbol{U}$.

4.2.3 The complexity of the algorithm

The algorithm introduced in the last section performs some computation for each subtree of $\mathcal{T}$. Therefore, the time complexity is at least proportional to the number of subtrees of $\mathcal{T}$. The data structures used by the algorithm require also storage space which is proportional the number of subtrees multiplied by the maximum number of leafs. The algorithm **Optimal-Order** is performed on the minimal subtree found by the procedure **Minimal-Subtree**. In order to analyze the complexity of the algorithm we will first examine how many leafs and how many subtrees there are in the minimal subtree.

The minimal subtree has at most $|U|$ leafs. In order to see that, let us look at the **Minimal-Subtree** algorithm. Note that each leaf in the minimal subtree must contain a variable from $\boldsymbol{U}$, otherwise the algorithm will remove it from the tree. Moreover, each leaf must contain a variable in $\boldsymbol{U}$ that is not contained in the rest of the tree. Particularly, it must contain a variable in $\boldsymbol{U}$ that is not contained in any of the other leafs. Hence, each leaf donates a new variable to the subset $|\boldsymbol{U}|$. Therefore, there can be no more then $|\boldsymbol{U}|$ leafs.

Now we can show a bound on the number of subtrees of the minimal subtree. We will bound the number of subtrees in a tree with a function of the number of leafs in the tree,

claim 4.2.2 *Let $\mathcal{T}$ be a tree with n nodes and d leafs. Denote $S(\mathcal{T})$ to be the number of subtrees in $\mathcal{T}$. Then,*

$$S(\mathcal{T}) \leq \sum_{i=1}^{d} \binom{n}{i} \leq \left(\frac{e \cdot n}{d}\right)^d \tag{4.8}$$

Proof: Note that every subtree of $\mathcal{T}$ has at most d leafs. The number of distinct subtrees with i nodes can be at most $\binom{n}{i}$, which is the number of subsets of nodes the nodes of $\mathcal{T}$ with size i. Thus, the total number of subtrees is at most $\sum_{i=1}^{d} \binom{n}{i}$. ∎

Combining the last two results, and denoting $|\boldsymbol{U}| = d$, we can see that the number of subtrees in the minimal subtree is $S(\mathcal{T}) = \mathcal{O}(n^d)$.

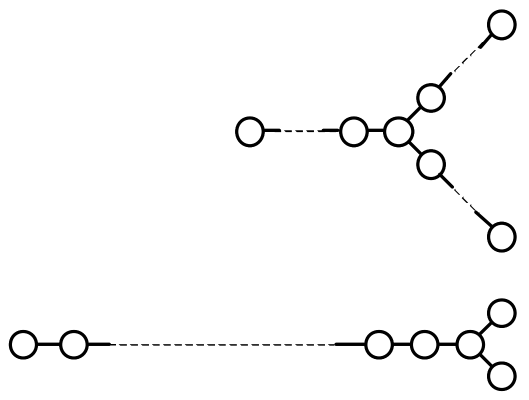


Figure 4-3: An example for the dependency of the number of subtrees on the structure of the tree. Both trees has n nodes and 3 leafs. The upper tree has approximately $(\frac{n}{3})^3$ leafs. The lower one has approximately $\frac{n^2}{2} + 2n$.

Obviously, the number of subtrees depends on the exact structure of the tree. For instance, trees in which the branches split close to the leafs will have fewer subtrees. Figure 4-3 shows two trees with n nodes and 3 leafs. The upper tree has approximately $(\frac{n}{3})^3$ leafs. The lower one has approximately $\frac{n^2}{2} + 2n$.

Now we can work out the complexity of the algorithm. For each subtree we look at all the edges constructing it. For each edge e we compute the cost of merging the subtree when e is the last edge being removed. Therefore we calculate at most $\mathcal{O}(S(\mathcal{T}) \cdot n) = \mathcal{O}(n^{d+1})$ intermediate results. each intermediate result requires searching the two search tree which takes $\mathcal{O}(logS(\mathcal{T}))$. The total number of computations is $\mathcal{O}(n^{d+1} \cdot d \cdot logn)$.

The algorithm needs $\mathcal{O}(n^d)$ space for storing the tables of $c(\mathcal{T}')$ and $l(\mathcal{T}')$. The table that stores the indices of the subtrees connected to the edge sides needs $\mathcal{O}(d \cdot n^d)$ space, because every index of a subtree with d leafs will be stored d times. Thus, the total storage space required by the algorithm is $\mathcal{O}(d \cdot n^d)$.

4.3 Using search methods to reduce the number of computations

The algorithm described in the previous section uses an exhaustive search in the space of all possible solutions to the clique merging problem. In order to reduce the number of computations we shall use a pruning method, which will avoid searching all of the solution space. The search algorithm employs a *Branch and bound* procedure for the pruning.

Branch and bound is a general technique for searching a solution in large spaces. This procedure decomposes the set of possible solutions into smaller sets. For each set it finds a lower bound on the cost of solutions contained in it. Sets having lower bounds smaller than some upper bound on the cost will be further decomposed. This procedure continues until it ends up with one singleton set with the correct solution.

In this section we will first present a representation of the search space, than we will describe the search algorithm.

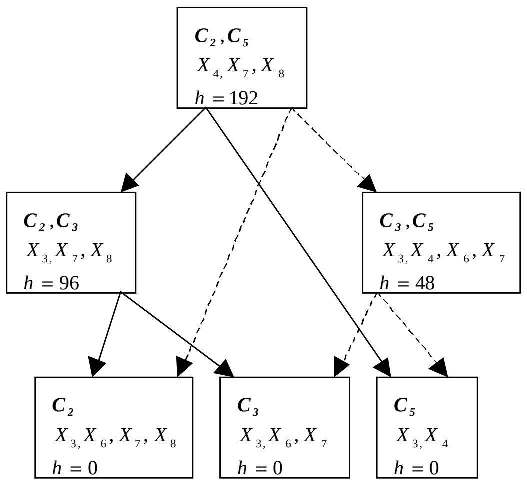


Figure 4-4: An AND-OR graph associated with a subtree of the clique tree in figure 4-1. On every node there is an indication of the cliques, the query variables and the cost associated with it.

4.3.1 AND-OR Graphs

As we saw in equations 4.5 and 4.6, the problem of finding the optimal order for merging a tree can be decomposed into smaller sub-problems in several ways. The solution is obtained by solving the sub-problems and examining which decomposition created a minimal cost solution. This type of problem solving can be represented by a structure termed *AND-OR graph* [23, 26].

AND-OR Graphs are defined as *hyper-graphs*. An hyper-graph is a collection of nodes and *hyper-arcs*. Each hyper-arc connects a parent node to a set of successor nodes. The hyper-arcs are also called *connectors*. AND-OR graph can represent boolean formulas. We can associate a boolean literal with every node. A connector represents a conjunction of the successor nodes connected through it. Every node is a disjunction of its connectors.

In the context of problem solving, every node is associated with a problem to be solved. The possible decompositions of a problem are represented by connectors. A problem represented by a node can be solved by solving *all* the subproblems that are represented by the nodes of *one* of the connectors emanating from the parent node.

We will use an AND-OR Graph in order to represent our search space. Every node will be associated with a subtree $\mathcal{T}$' and a set of query variables. A connecter emanating from a node represents a decomposition of the subtree into two subtrees of $\mathcal{T}$' connected to one of the edges of $\mathcal{T}$'. These connectors will point to nodes associated with the appropriate subtrees and query variables. The AND-OR Graph will have a root node representing the clique tree and the query we want to perform on it. Subtrees containing single cliques are represented by *terminal nodes.*

Figure 4-4 shows the AND-OR graph associated with a subtree of the clique tree in figure 4-1 and the query variables $\{X_4, X_9, X_{10}\}$. Note that a node can be pointed to by several connectors, illustrating that its subproblem can be used for solving more than one problem.

The task of solving the problem represented by the root node can be regarded as finding a *solution graph.* A solution graph is a subgraph of the AND-OR graph which contains the root node. Every non terminal node of the solution graph has one outgoing connector. A solution graph for a node n is defined as the solution graph of the subgraph composed of the descendants of n. In figure 4-4 the solution graph is marked with solid lines.

Let us denote the *cost* of a solution graph of a node n by $c(n)$. In order to define $c(n)$ we will assign every connector with the cost of solving the problem of the parent node by using the solutions of the successor nodes. The cost of a terminal node is the cost of solving the problem it represents. In the clique merging problem, $c(l) = 0$ for every terminal node l. The cost of a solution graph is the sum of costs of the connectors and terminal nodes the graph is comprised of. An *optimal* solution graph is one with a minimal cost. We shall denote the cost of the optimal solution graph of a node n by $h(n)$.

We can find an optimal solution graph using a breadth first search. The AND-OR graph will be constructed within the search process. We will initialize the graph to contain the root node. Nodes are created by *expanding* their ancestors, i.e. creating the connectors and successor nodes. Every new node constructed in this procedure might result in updating the cost of its ancestors according to equations 4.5 and 4.6. This procedure is a top down version of the dynamic programming algorithm presented in section 4.2.

4.3.2 The AO^* algorithm

The AO^* algorithm [23, 26] is a branch and bound technique for finding an optimal solution in AND-OR graphs. During the search process, the algorithm updates the optimal solution cost by an estimation of $h(n)$ for a node n. On every step, one of the leaf nodes of the solution graph that appears to be optimal is expanded. This way nodes that lead to an expensive solution are avoided.

Figure 4-5 provides a detailed explanation of the AO^* algorithm. Here we assume that no cycles can be formed in the AO-Graph. This is true for our specific problem. We will describe briefly the operation of the algorithm.

The algorithm initializes the graph G to contain the root node $INIT$. If $INIT$ is a terminal node it is labeled $SOLVED$. Otherwise $h'(INIT)$ is calculated using a heuristic estimation. Then the algorithm is repeatedly expanding the graph until an optimal solution is found.

In every intermediate step, the algorithm will maintain the estimated cost $h'(n)$ of every node in the graph. Every expanded node will have a mark on the connector that leads to the cheapest solution. Terminal nodes and nodes on which an optimal solution graph has been found will be labeled $SOLVED$.

An expanding iteration is composed of two operations. First, the best partial solution graph is traced through the marked nodes. One of the non-terminal nodes of the solution graph is expanded and a heuristic estimated cost is computed for all of the new successors.

The second operation is an update of the estimated costs, the $SOLVED$ labels and the best path markers. The estimated cost of the expanded node is updated to be the minimum over all the arcs emanating from it, of the estimated costs of the successors connected through the arc and the cost of combining them to solve the problem associated with the parent node. This procedure is done from the expanded node through its ancestors up to the root node. A node n is updated if one of its successors has been updated in a manner that might change the status of n.

- Let $GRAPH$ consist only of the node representing the initial state (call this node $INIT$). Compute $h'(INIT)$.
- Until $INIT$ is labeled solved repeat the following procedure:
  - Trace the labeled arcs from $INIT$ and select for expansion one of the as yet unexpanded nodes that occur on the path. Call the selected node $NODE$.
  - Generate the successors of $NODE$. for each successor (called $SUCCESSOR$) do the following:
    * Add $SUCCESSOR$ to $GRAPH$.
    * If $SUCCESSOR$ is a terminal node, label it solved and assign it an h' value of 0. Otherwise, compute its h' value.
  - Propagate the newly discovered information up the graph by doing the following: Let S be a set of nodes that have been labeled $SOLVED$ or whose h' values have been changed and so need to have values propagated back to their parents. Initialize S to $NODE$. Until S is empty, repeat the following procedure:
    * Select from S a node none of whose descendents in $GRAPH$ occur in S. Call this node $CURRENT$ and remove it from S.
    * Compute the cost of each of the arcs emerging from $CURRENT$. The cost of each arc is equal to the sum of the h' values of each of the nodes at the end of the arc plus whatever the cost of the arc itself is. Assign as $CURRENT$'s new h' value the minimum of the costs just computed for the arcs emerging from it.
    * Mark the best path out of $CURRENT$ by marking the arc that had the minimum cost computed in the previous step.
    * Mark $CURRENTSOLVED$ if all of the nodes connected to it through the new labeled arc have been labeled $SOLVED$.
    * If $CURRENT$ has been labeled $SOLVED$ or if the cost of $CURRENT$ was just changed, then its new status must be propagated back up the graph. So add all of the ancestors of $CURRENT$ to S.

Figure 4-5: The AO^* algorithm

4.3.3 The heuristic estimations

The estimated cost $h'(n)$ of a node n provides the algorithm a mean of pruning the search space. Nodes having high estimated cost relative to others will not be further developed. The nature of the heuristic estimation determines the quality of the solution and the amount of computations needed to reach a solution. On the extreme situation of using a perfect estimator in which h' is the true optimal cost, the algorithm will find the optimal solution without examining unnecessary branches of the AO Graph. It can be shown that if h' never overestimates an optimal solution will be found [23]. In that case, the closer the estimation is to the true optimal cost, the more effective the pruning will be. Usually, when using a lower bound as an estimator, there is a tradeoff between its pruning capabilities and the time required to compute it. When it hard to find a tight lower bound, one can use h' that over estimates. In that case the solution found will not be optimal.

In our experiments with the AO^* algorithm we used two heuristic estimations. The first estimator was a very loose lower bound, which estimated the cost of marginalizing to a set of variables $\boldsymbol{U}$ to be the size of the domain of $\boldsymbol{U}$. This estimator leads to finding an optimal solution, but the search is almost exhaustive.

The other estimator we used is an upper bound which is gained by calculating the cost of a particular solution. We use the greedy procedure presented in section 4.1.2 in order to find a fair solution to the marginalization problem. Then we use the AO^* in order to improve this solution.

4.4 Experimental Results

Our experiments tested the different methods for finding the order of operations. We examined the effect of every method on the total intermediate clique potential size and the run time of marginalization. We also examined the time required by each method to find the best sequence. A combination of these factors may guide us while choosing an optimizing procedure for practical purpose. The search methods we examined were the following:

GR0: Greedy top down search - A greedy algorithm that chooses the edges to be removed in reverse order. The separator being chosen is the one decomposing the problem to the smallest pair of subproblems.

GR1: Greedy variable elimination search in which we use the greedy algorithm presented in section 4.1.2 with the heuristic of [13].

AO0: Exhaustive search - The AO^* algorithm with the domain size of the query variables as the heuristic estimator.

AO1: AO^* search with a heuristic estimator calculated by **GR1**.

We also tested two more greedy algorithms. The first one is a greedy variable elimination in which the variable with the smallest weight is being chosen (see section 4.1.2). The second algorithm approximates the top-down greedy algorithm by choosing the separators according to their domain size in decreasing order. The four algorithm we described above gave significantly better results then the last two algorithm therefore we shall omit them from our discussion.

In our experiments we generated random queries from the following four Bayesian-Networks:

Network	Number of variables	Domain size of variables	Number of cliques	Total clique domain size
Alarm	37	2-4	25	10680
Movies	51	8	44	$1.70 \cdot 10^6$
Pigs	441	3	366	$3.64 \cdot 10^7$
Munin1	189	1-21	161	$1.62 \cdot 10^9$
Munin2	1003	2-21	866	$2.54 \cdot 10^7$
Munin3	1044	1-21	904	$2.99 \cdot 10^7$
Munin4	1041	1-21	876	$1.47 \cdot 10^8$

Table 4.1: The networks used in the experiments. The columns describe the number of variables in the Bayesian-network, the range of the domain size of the variables, the number of cliques in the clique tree and the total domain size of the cliques in the clique tree.

- Alarm - A network for intensive care patient monitoring [3].
- Movies - A network for grading movies. [1] [2]
- Pigs - Pedigree of breeding pigs. The pedigree is used for diagnosing the PSE disease. [3]
- Munin - Four subsets of the MUNIN EMG assistant. [1]

Table 4.1 describes further details about the size of these networks and their corresponding clique trees. The clique trees for the Munin networks were built with the aid of an elimination ordering suggested by Uffe Kjærulff.

For each network we chose 40 random subsets of N query variables, $N = 2, ..., 6$. On every single test, for all the search methods, we examined the search time, the resulting total size of intermediate factors and the resulting time required to perform the marginalization. The exhaustive search method was tested only on the Alarm and the Movies networks. The greedy top-down search served as a base-line to which we compared the other algorithms.

Figures 4-6 to 4-12 show the comparison results for each network. For each network and each method we show the ratio between the total intermediate clique potential size of the base-line method and the other method as a function of the number of leafs in the trimmed clique tree. This gives a measure of the factor of run time acceleration and the memory saving. We also plot a graph that shows, for every experiment, the preprocessing time required to find the order of operations and the saving in running time resulting from using the new method. In this graph we assume that if the order found by the base-line is better then the other order the base line order is used. Instances in which the base-line is better then the other method are presented as having zero saving in runtime. The ratio between the search time and the saving in running time indicates the number of instances required to make the method worth using.

Common to all experiments, we see that the bigger networks and queries get greater improvements in runtime. Also, as expected, the improvement is greater for networks having variables with larger domains.

[1]Generated with data from Internet Movie Database Limited ©1999-2000.

[2]http://www.research.digital.com/SRC/EachMovie.

[3]Created by Claus S. Jensen on the basis of a data base from Søren Andersen (Danske Slagterier, Axeltorv Copenhagen).

Comparing the exhaustive search to the greedy top down search method in the Alarm and Movies networks we see average improvement factors between 1.5 and 4. Note that in the alarm network this method improves performance only if there are over 1500 instances. The variables on the Movies network have a large domain of 8 values. Therefor, preprocessing time is mostly smaller than the computation time of the marginals, making the preprocessing methods worth activating even for a small number of instances. This fact is also true for the larger even if the domain sizes are smaller.

The average improvement factors obtained by the search method **AO1** ranges from close to 1 in the alarm network up to over a 1000 in the Munin networks. Although the average improvement in the Munin networks is extremely big, for most instances of the big queries the improvement is about a factor of 10.

The improvement factors of the greedy algorithm **GR1** seems to be slightly lower than those of **AO1**. Therefore, we also compared the performance of **AO1** and **GR1**. For the alarm and the movies networks the search method gives minor average improvements. In the pigs network the average improvement factor is over 1.5. The tests on Munin1, Munin2, Munin3 and Munin4 showed average improvement factors of over 3, 1.5, 2.5 and 2 respectively.

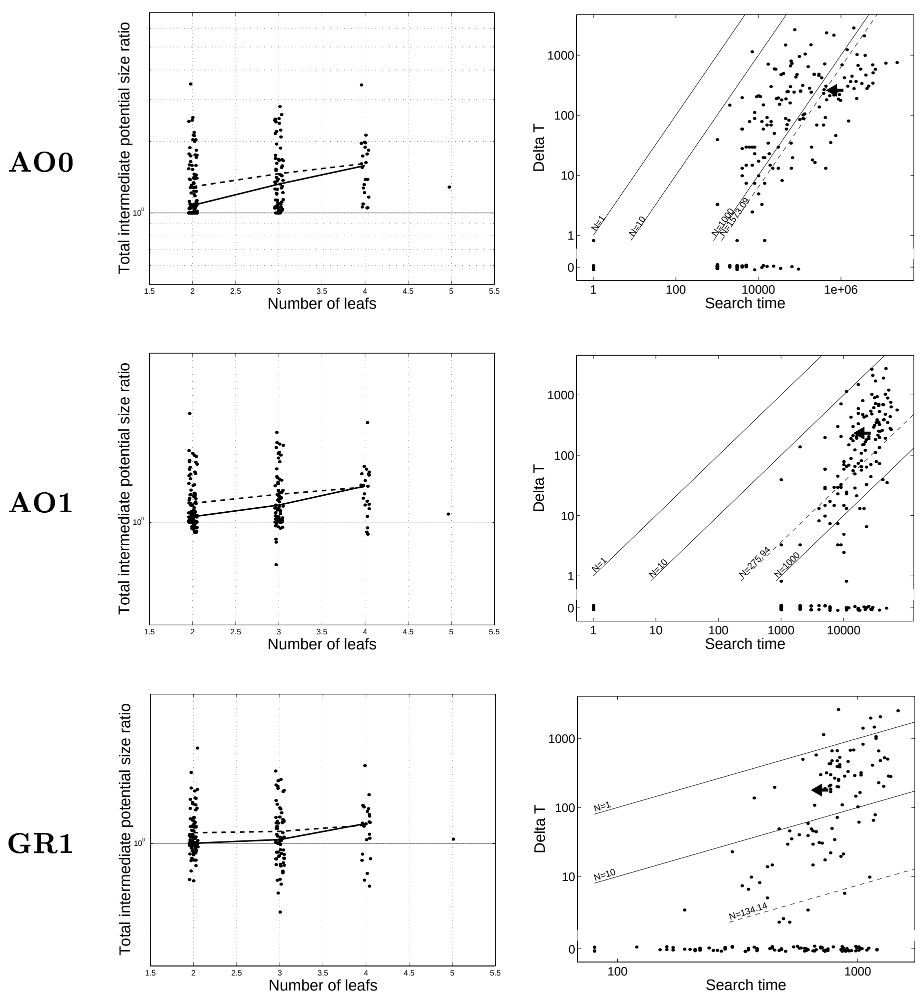
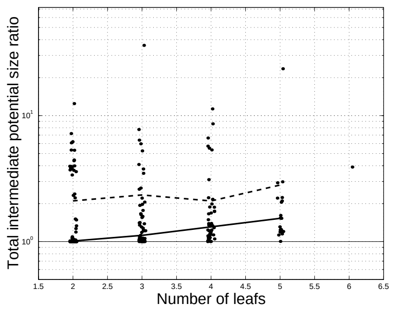
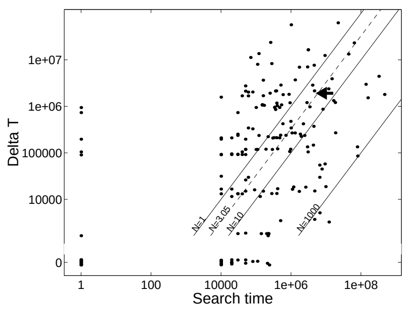


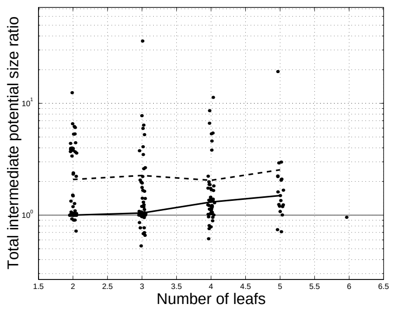
Figure 4-6: Experimental results for the Alarm network: The left plots shows the ratio in the total intermediate clique potential size between the **GR0** method and the other methods as a function of the number of leafs in the trimmed junction tree. Every dot represents a single experiment. The solid and the dashed lines represent the median and the average savings as a function of the number of leafs respectively. In the right plots, the x axis represents the preprocessing time for every experiment. The y axis represents the difference in the time required for marginalization between the **GR0** method and the other method. The numbers on the contours indicate the ratio between the search time and the saving in marginalization time for points falling on that contour. The dashed contour indicates the median of this ratio for the test points. The arrow points to average of this points. Note that the average is highly influenced by outliers but it is the relevant measure for tasks involving a large number of queries. The points with zero delta T represents instances where the baseline is equal or better then the other method. The upper plots compares **AO0**, the middle compares **AO1** and the lower one compares **GR1**.

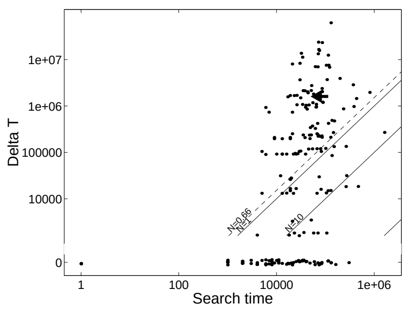
AO0



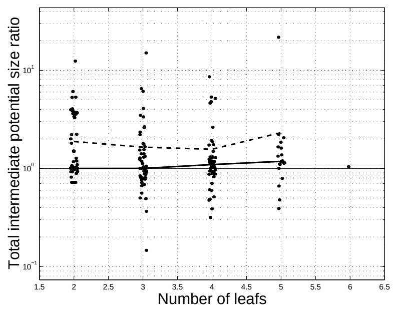


AO1





GR1



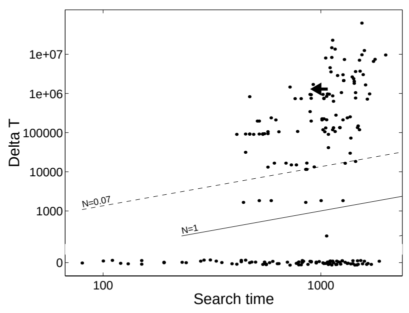


Figure 4-7: Experimental results for the Movies network.

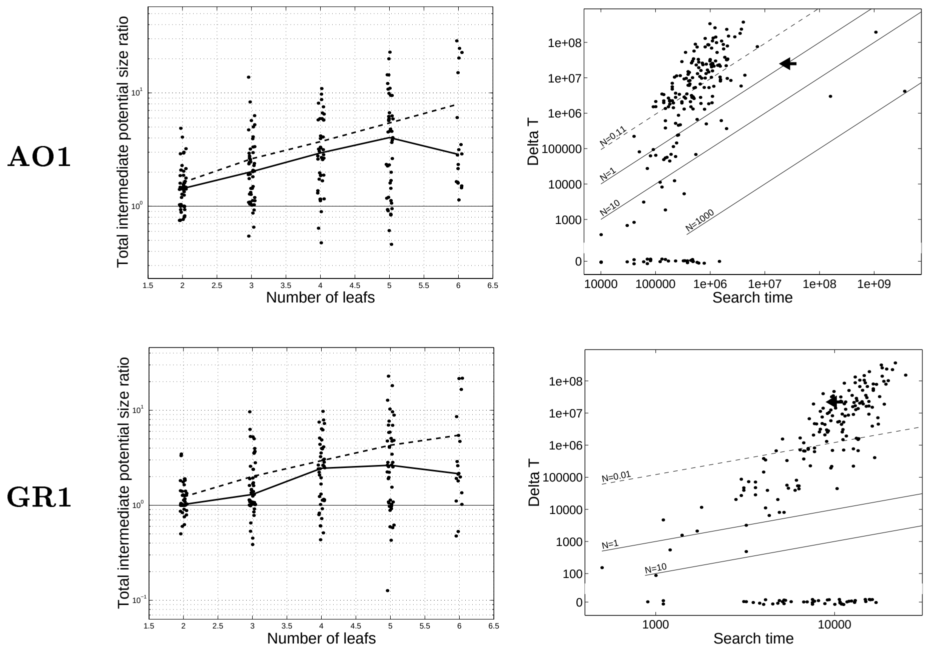


Figure 4-8: Experimental result for the Pigs network. The upper plots compares **AO1** and the lower one compares **GR1**.

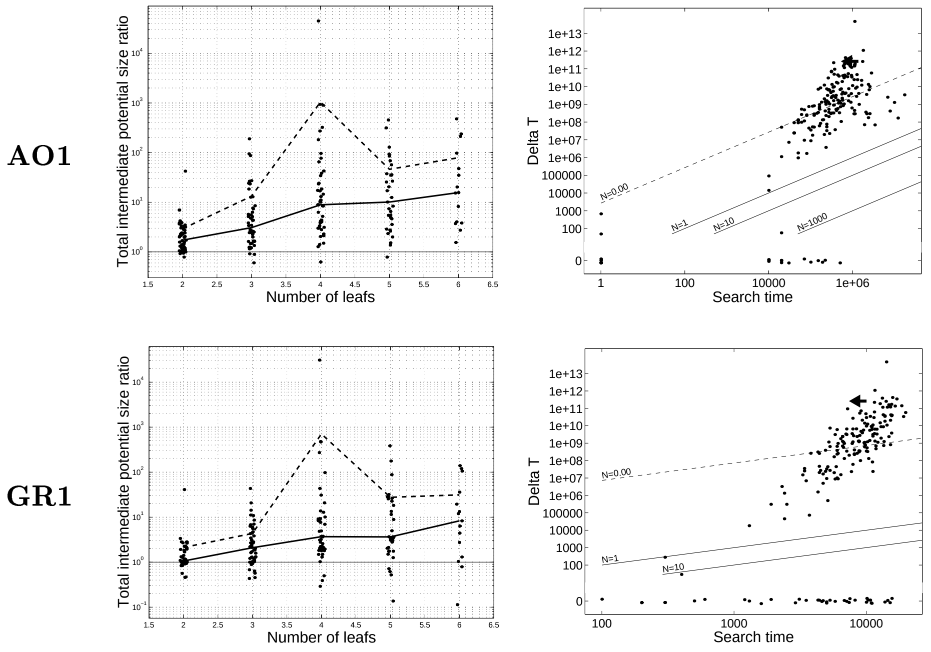


Figure 4-9: Munin1

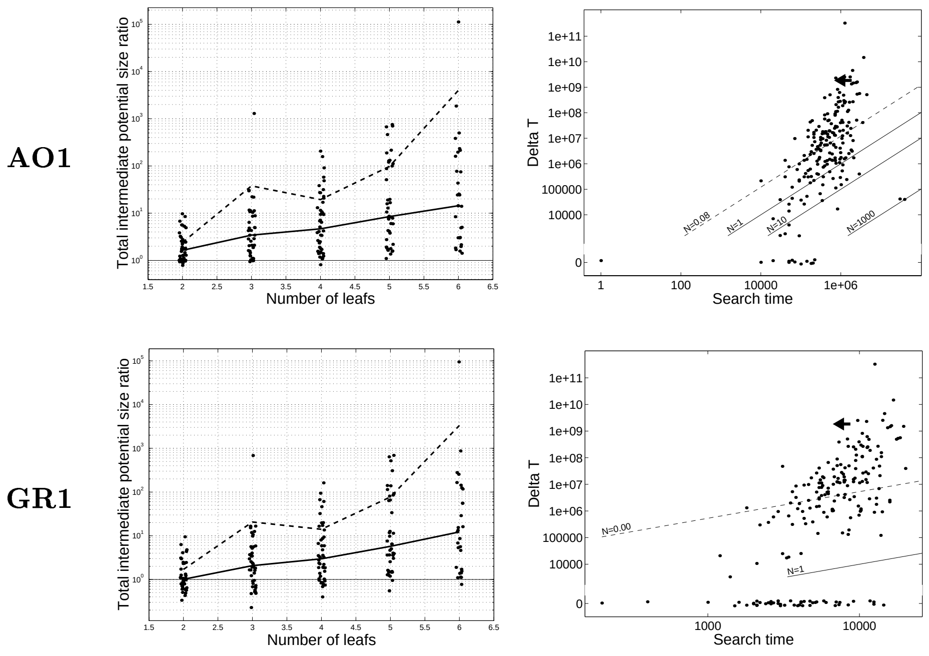


Figure 4-10: Munin2

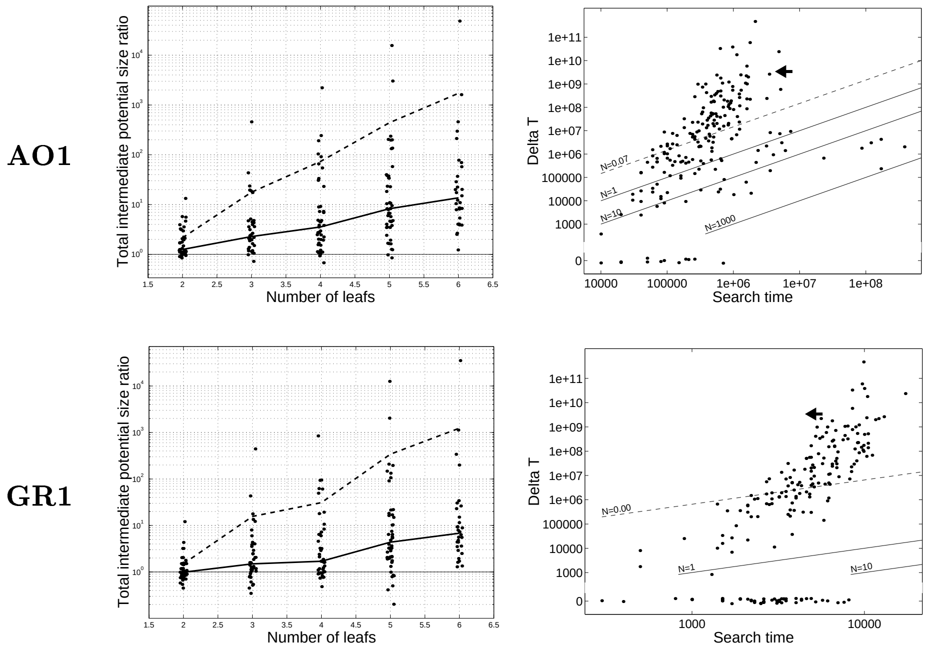


Figure 4-11: Munin3

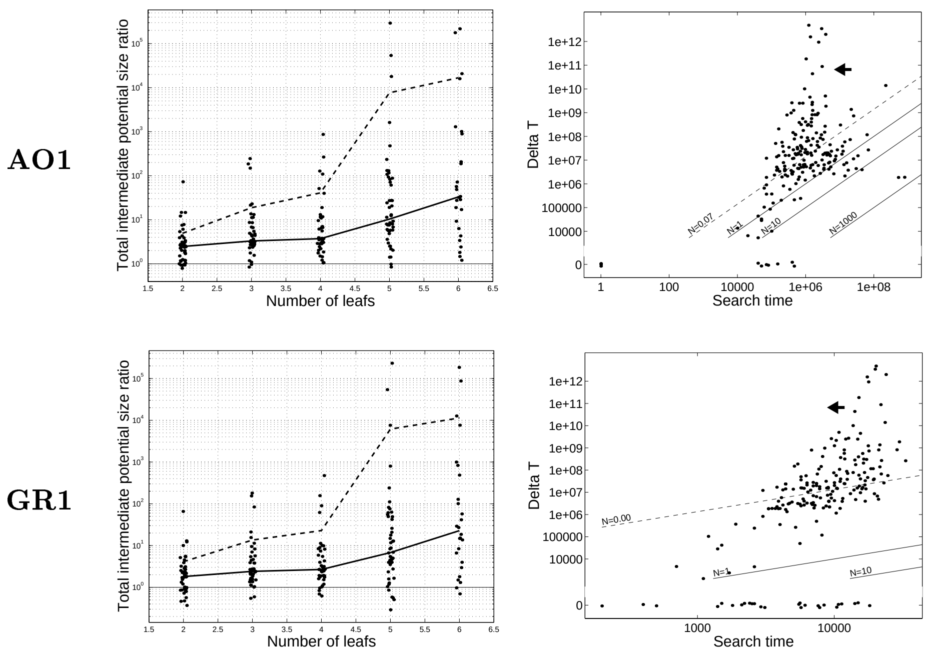


Figure 4-12: Munin4

4.5 Proofs

To prove the bound on $S(\mathcal{A}')$ we will first consider the case where we want to calculate the probability table over all the variables present in a junction tree.

claim 4.5.1 *Let $\mathcal{T}$ be a junction tree over a set $\boldsymbol{V}$ of random variables. Let $\mathcal{A}'(\mathcal{T}, \boldsymbol{V})$ be a sequence of clique mergings that calculates the probability table over $\boldsymbol{V}$. Then,*

$$S(\mathcal{A}') \leq 2 \cdot dom(\boldsymbol{V})$$

Proof: We will prove the claim by induction on the number of cliques. Obviously, for a single clique, $S(\mathcal{A}') = 0$ and the claim is true. Suppose the claim is true for a tree with $n-1$ cliques. Let $\mathcal{T}$ be a junction tree with n cliques. Without loss of generality we shall assume that no clique is contained in a neighboring clique, otherwise we can remove this clique and it separator without performing any computation. Let $\mathcal{A}'(\mathcal{T}, \boldsymbol{V})$ be a sequence of operations that computes $P(\boldsymbol{V})$ by a sequence of edge removals. Suppose e is the last edge that was removed by $\mathcal{A}'$. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be the two subtrees emanating from e. Let $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ be the sets of variables contained in $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. $\mathcal{A}'$ first calculates the potentials on $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ using sequences $\mathcal{A}'_1$ and $\mathcal{A}'_2$ respectively. Then it merges the two potentials to get the potential on $\boldsymbol{V}$. hence, the total size of intermediate cliques is,

$$S(\mathcal{A}') = S(\mathcal{A}'_1) + S(\mathcal{A}'_2) + |dom(\boldsymbol{V})| \tag{4.9}$$

We shall now consider two cases. In the first case we assume that one of the two subtrees, wlog $\mathcal{T}_1$, contains only on clique. In this case there is at least one variable contained in $\mathcal{T}_1$ but is not contained in $\mathcal{T}_2$. Therefore $\boldsymbol{V}_2 \subset \boldsymbol{V}$ and we have,

$$\begin{aligned} S(\mathcal{A}') &= S(\mathcal{A}'_1) + S(\mathcal{A}'_2) + |dom(\boldsymbol{V})| \\ &\leq 0 + 2 \cdot |dom(\boldsymbol{V}_2)| + |dom(\boldsymbol{V})| \\ &\leq 2 \cdot |dom(\boldsymbol{V})| \end{aligned}$$

Suppose now that both subtrees contain more than one clique. The running intersection property and the assumption we made that no clique is contained in a neighbor clique, implies that if $\mathcal{T}'$ is a subtree of $\mathcal{T}$, then the number of variables in $\boldsymbol{V} \setminus \boldsymbol{V}'$ is at least the number of cliques that are in $\mathcal{T}$ but in $\mathcal{T}'$. Consequently $|\boldsymbol{V}_1| \leq |\boldsymbol{V}| - 2$ and $|\boldsymbol{V}_2| \leq |\boldsymbol{V}| - 2$. Thus,

$$\begin{aligned} S(\mathcal{A}') &= S(\mathcal{A}'_1) + S(\mathcal{A}'_2) + |dom(\boldsymbol{V})| \\ &\leq 2 \cdot |dom(\boldsymbol{V}_1)| + 2 \cdot |dom(\boldsymbol{V}_2)| + |dom(\boldsymbol{V})| \\ &\leq 2 \cdot \frac{1}{4}|dom(\boldsymbol{V})| + 2 \cdot \frac{1}{4}|dom(\boldsymbol{V})| + |dom(\boldsymbol{V})| \\ &= 2 \cdot |dom(\boldsymbol{V})| \end{aligned}$$

▮

We can now prove the general case,

Claim 4.2.1: Let $\mathcal{T}$ be a junction tree. Let $\boldsymbol{U}$ be a subset of $\boldsymbol{V}$. Suppose $\mathcal{A}(\mathcal{T}, \boldsymbol{U})$ is a sequence of operations computing $P(\boldsymbol{U})$. Then, there exists a sequence of local clique mergings $\mathcal{A}'(\mathcal{T}, \boldsymbol{U})$ such that,

$$S(\mathcal{A}') \leq 2 \cdot S(\mathcal{A})$$

Proof: We will prove this result by induction on the number of variables that have to be marginalized, $n = |\boldsymbol{V} \setminus \boldsymbol{U}|$. For $n = 0$ we get the case of claim 4.5.1, and we are done.

Suppose now that $|\boldsymbol{V} \setminus \boldsymbol{U}| = n$. Let X be the first variable that is marginalized out by $\mathcal{A}$. We can assume that the operations performed by $\mathcal{A}$ prior to the marginalization of X are multiplication and division of all the cliques that contain X. Indeed, in order to marginalize out X we have to multiply all the potentials that contain X. Also, if we multiplied a clique that does not contain X, we can change the order of multiplication and multiply that clique after the marginalization of X. It can be shown by direct calculation, using the fact that the domain size of each variable is at least 2, that this will result in a lower sum of intermediate clique size.

The set of cliques containing X constitutes a subtree of $\mathcal{T}$. We shall denote this subtree by $\mathcal{T}_X$ and the sequence of operations done by $\mathcal{A}$ before marginalization of X by $\mathcal{A}(\mathcal{T}_X, \boldsymbol{V}_X)$, where $\boldsymbol{V}_X$ is the set of variables contained in $\mathcal{T}_X$. According to claim 4.5.1 we can found a sequence $\mathcal{A}'(\mathcal{T}_X, \boldsymbol{V}_X)$ of local clique mergings on $\mathcal{T}_X$ so that,

$$S(\mathcal{A}'(\mathcal{T}_X, \boldsymbol{V}_X)) \leq 2 \cdot S(\mathcal{A}(\mathcal{T}_X, \boldsymbol{V}_X)) \tag{4.10}$$

After marginalizing X we are left with a clique tree $\mathcal{T}^-$ from which we would like to marginalize out $n-1$ variables. This is done by the suffix of the sequence $\mathcal{A}$, denoted by $\mathcal{A}(\mathcal{T}^-, \boldsymbol{U})$. Using the induction hypothesis we can find a sequence of clique mergings $\mathcal{A}'(\mathcal{T}^{-}, \boldsymbol{U})$ in which,

$$S(\mathcal{A}'(\mathcal{T}^{-}, \boldsymbol{U})) \leq 2 \cdot S(\mathcal{A}(\mathcal{T}^{-}, \boldsymbol{U})) \tag{4.11}$$

Combining 4.10 and 4.11 we get,

$$S(\mathcal{A}'(\mathcal{T}, \boldsymbol{U})) \leq 2 \cdot S(\mathcal{A}(\mathcal{T}, \boldsymbol{U})) \tag{4.12}$$

▮

Chapter 5

Conclusions and further research

The necessity of efficient and accurate approximate inference schemes arises from the complexity of exact inference in many probabilistic models of real life phenomena. The variational approximation methods is a thoroughly studied approximate inference tool that has been shown to provide a good tradeoff between complexity and accuracy. In the first part of the thesis we presented to extensions of structured variational methods–based on chain graphs and additional hidden variables. Each extension exploits a representational structure that allows to better match a tractable approximating network to the posterior. Such extensions can potentially provide better tradeoff between network complexity on one hand and the accuracy of approximation on the other.

The introduction of extra hidden variables have led to an intractable optimization problem of finding parameters that would minimize the KL-distance between the true distribution and approximating distribution. In order to utilize the potential of hidden variables we used an upper bound on the KL-distance as a target function for optimization. Experimental results suggest that addition of hidden variables improve accuracy significantly. The upper bound on KL-distance gained by adding extra hidden variables was in most cases lower than the actual KL-distance gained by employing conventional structured approximations. The complexity increment that arises from adding these variables is proportional to the clique size of the approximating network and quadratic in the number of hidden values in the clique. Therefore, it is easy to adjust the complexity of approximation.

We put emphasis on presenting uniform machinery in the derivations of the three variants we considered. This uniform presentation allows for better insights into the workings of such approximations and simplifies the process of deriving new variants for other representations. Many real life models involve other types of distributions not considered here. The principles presented here can be applied easily to other distributions from the exponential families. Applying these principles to additional types of distributions (such as sigmoid or noisy-or) requires more effort.

While addressing the problem of efficient computations of marginal probabilities required to update every *single* parameter, we did not address the problem of efficient *series* of updates. However, much computation (up to a quadratic factor) can be saved by cautious planning of order of asynchronous updates.

The grand challenge for applications of such variational methods is to build automatic procedures that can determine what structures matches best a given network with a given query. This is a non-trivial problem. We hope that some of the insights we got from our derivations can provide initial clues that will lead to development of such methods.

In the second part of the thesis we confronted the problem of computing marginal distributions from junction trees of sets of variables that are not contained within a single clique. We investigated methods to reduce the time and space complexity of this task in a preprocessing phase. This assignment is an essential subroutine in the variational approximation tasks presented in the first part. An additional task that uses this subroutine is structure learning of Bayesian networks with missing values [11]. In the later problem, the computation of a specific subset of variables is repeated many times with different values. Therefore applying an optimization algorithm for this task in the preprocessing phase might reduce computation time significantly.

We purposed a dynamic programming algorithm, exploiting the tree structure, and showed that it yields almost an optimal elimination order. Since the algorithm is too complex we recoursed to greedy and search methods. Investigating several greedy methods we saw that the best greedy method is the greedy variable elimination with the heuristic purposed by [13] (See section 4.1.2). This method is fast compared to the marginalization task and is useful for small networks having dozens of nodes. Plugging in the greedy algorithm to an AO^* search procedure results in a further improvement, especially in larger networks with hundreds of random variables.

Further improvements to this algorithm might be done by using different search methods, similar to different algorithm used for graph triangulation [20]. An open question remains, if the dynamic programming algorithm can be improved by reducing the number of subtrees that are examined by the algorithm.

Bibliography

[1] S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck, and F. Jensen. MUNIN — an expert EMG assistant. In John E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam, 1989.

[2] D. Barber and W. Wiegerinck. Tractable variational structures for approximating graphical models. In S. Solla M. Kearns and D. Cohn, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 11, pages 183–189, 1999.

[3] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proc. 2'nd European Conf. on AI and Medicine.* Springer-Verlag, 1989.

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

[5] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & sons, 1991.

[6] Robert G. Cowell, A. Philip Dawid, Steffen L. Lauritzen, and David J. Speigelhalter. *Probabilistic Networks and Expert Systems.* Springer, 1999.

[7] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, pages 141–153, 1993.

[8] P. Dagum and M. Luby. An optimal approximation algorithm for bayesian inference. *Artificial Intelligence*, pages 1–27, 1997.

[9] H. J. Suermondt E. J. Horvitz and G. F. Couper. Bounded conditioning: Flexible inference for decisions under scarce resourses. In *Uncertainty in Artificial Inteligence: Proceedings of the Fifth Conference*, 1989.

[10] T. El-Hay and N. Friedman. Incorporating expressive graphical models in variational approximations: Chain-graphs and hidden variables. In *Proc. Seventeenth Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2001.

[11] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Fourteenth Inter. Conf. on Machine Learning (ICML).*, 1997.

[12] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–274, 1997.

[13] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, October 1996.

[14] W. T. Freeman J. S. Yedidia and Y. Weiss. Generalized belief propagation. In *NIPS*, 2000.

[15] T. Jaakkola and M. Jordan. Computing upper and lower bounds on likelihoods in intractable networks. In *Proceedings of the Twelfth Annual Confenence on Uncertainty in Artifical Intelligence (UAI-96)*, pages 340–348, 1996.

[16] T. S. Jaakkola. Tutorial on variational approximation methods. In *Advanced mean field methods: theory and practice*. MIT Press, 2000. http://www.ai.mit.edu/ tommi/papers.html.

[17] T. S. Jaakkola and M. I. Jordan. Improving the mean field approximation via the use of mixture models. In Michael I. Jordan, editor, *Proceedings of the NATO ASI on Learning in Graphical Models*. Kluwer, 1997.

[18] F. V. Jensen. *An Introduction to Bayesian Networks*. University College London Press, 1996.

[19] M. I. Jordan, Z. Ghahramani, T. Jaakkola, and L. K. Saul. An introduction to variational approximations methods for graphical models. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands, 1998.

[20] U. Kjærulff. Triangulation of graphs — algorithms giving small total state space. Technical report, Dept. of Mathematics and Computer Science, Aalborg University, 1990.

[21] D.J.C. MacKay. Introduction to monte carlo methods. In M. I. Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1998.

[22] R. Neal and G. Hinton. A view of the em algorithm taht justifies incremental, sparse, and other variants. In Michael I. Jordan, editor, *Proceedings of the NATO ASI on Learning in Graghical Models*. Kluwer, 1997.

[23] N. J. Nillson. *Principles of Artificial Intelligence*. Morgan Kaufman, 1980.

[24] J. Pearl. *Probabilistic Reasoning in Inelligent Systems*. Morgan Kaufman, 1988.

[25] C. Peterson and J.R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[26] E. Rich and K. Knight. *Artificial Intelligence*. McGraw-Hill, 1991.

[27] R. T. Rockafellar and R. J-B. Wets. *Variational Analysis*. Springer, 1998.

[28] D. G. Corneil S. Arnborg and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.

[29] L. K. Saul and M. I. Jordan. Exploiting tractable substructures in intractable networks. In Michael C. Mozer David S. Touretzky and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 486–492. MIT Press, 1996.

[30] W. Wiegerinck. Variational approximations between mean field theory and the junction tree algorithm. In *Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI 2000)*, 2000.

[31] Hong Xu. Computing marginals from the marginal representation in markov trees. *Artificial Intelligence*, 74:177–189, 1995.