2 Graphical Models in a Nutshell

Daphne Koller, Nir Friedman, Lise Getoor and Ben Taskar

Probabilistic graphical models are an elegant framework which combines uncertainty (probabilities) and logical structure (independence constraints) to compactly represent complex, real-world phenomena. The framework is quite general in that many of the commonly proposed statistical models (Kalman filters, hidden Markov models, Ising models) can be described as graphical models. Graphical models have enjoyed a surge of interest in the last two decades, due both to the flexibility and power of the representation and to the increased ability to effectively learn and perform inference in large networks.

2.1 Introduction

Graphical models [11, 3, 5, 9, 7] have become an extremely popular tool for modeling uncertainty. They provide a principled approach to dealing with uncertainty through the use of probability theory, and an effective approach to coping with complexity through the use of graph theory. The two most common types of graphical models are Bayesian networks (also called belief networks or causal networks) and Markov networks (also called Markov random fields (MRFs)).

At a high level, our goal is to efficiently represent a joint distribution P over some set of random variables $\mathcal{X} = \{X_1, \ldots, X_n\}$. Even in the simplest case where these variables are binary-valued, a joint distribution requires the specification of 2^n numbers — the probabilities of the 2^n different assignments of values $x_1, \ldots, x_n$. However, it is often the case that there is some structure in the distribution that allows us to factor the representation of the distribution into modular components. The structure that graphical models exploit is the independence properties that exist in many real-world phenomena.

The independence properties in the distribution can be used to represent such high-dimensional distributions much more compactly. Probabilistic graphical models provide a general-purpose modeling language for exploiting this type of structure in our representation. Inference in probabilistic graphical models provides us with

the mechanisms for gluing all these components back together in a probabilistically coherent manner. Effective learning, both parameter estimation and model selection, in probabilistic graphical models is enabled by the compact parameterization.

This chapter provides a compact graphical models tutorial based on [8]. We cover representation, inference, and learning. Our tutorial is not comprehensive; for more details see [8, 11, 3, 5, 9, 4, 6].

2.2 Representation

The two most common classes of graphical models are *Bayesian networks* and *Markov networks*. The underlying semantics of Bayesian networks are based on directed graphs and hence they are also called *directed graphical models*. The underlying semantics of Markov networks are based on undirected graphs; Markov networks are also called *undirected graphical models*. It is possible, though less common, to use a mixed directed and undirected representation (see, for example, the work on chain graphs [10, 2]); however, we will not cover them here.

Basic to our representation is the notion of *conditional independence*:

Definition 2.1
Let $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ be sets of random variables. $\boldsymbol{X}$ is *conditionally independent* of $\boldsymbol{Y}$ given $\boldsymbol{Z}$ in a distribution P if

$$P(\boldsymbol{X} = \boldsymbol{x}, \boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{Z} = \boldsymbol{z}) = P(\boldsymbol{X} = \boldsymbol{x} \mid \boldsymbol{Z} = \boldsymbol{z}) P(\boldsymbol{Y} = \boldsymbol{y} \mid \boldsymbol{Z} = \boldsymbol{z})$$

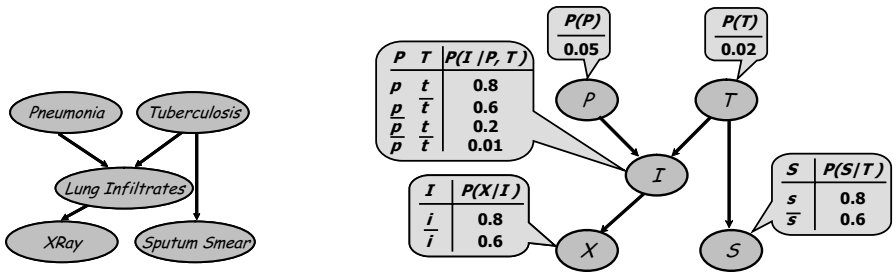
for all values $\boldsymbol{x} \in Val(\boldsymbol{X})$, $\boldsymbol{y} \in Val(\boldsymbol{Y})$ and $\boldsymbol{z} \in Val(\boldsymbol{Z})$. ∎

In the case where P is understood, we use the notation $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z})$ to say that $\boldsymbol{X}$ is conditionally independent of $\boldsymbol{Y}$ given $\boldsymbol{Z}$. If it is clear from the context, sometimes we say "independent" when we really mean "conditionally independent".

2.2.1 Bayesian Networks

The core of the Bayesian network representation is a directed acyclic graph (DAG) $\mathcal{G}$. The nodes of $\mathcal{G}$ are the random variables in our domain and the edges correspond, intuitively, to direct influence of one node on another. One way to view this graph is as a data structure that provides the skeleton for representing the joint distribution compactly in a factorized way.

Let $\mathcal{G}$ be a BN graph over the variables $X_1, \ldots, X_n$. Each random variable X_i in the network has an associated *conditional probability distribution (CPD)* or *local probabilistic model*. The CPD for X_i, given its parents in the graph (denoted $\mathbf{Pa}_{X_i}$), is $P(X_i \mid \mathbf{Pa}_{X_i})$. It captures the conditional probability of the random variable, given its parents in the graph. CPDs can be described in a variety of ways. A common, but not necessarily compact, representation for a CPD is a table which contains a row for each possible set of values for the parents of the node describing



(a) (b)

Figure 2.1 (a) A simple Bayesian network showing two potential diseases, *Pneumonia* and *Tuberculosis*, either of which may cause a patient to have *Lung Infiltrates*. The lung infiltrates may show up on an *XRay*; there is also a separate *Sputum Smear* test for tuberculosis. All of the random variables are Boolean. (b) The same Bayesian network, together with the conditional probability tables. The probabilities shown are the probability that the random variable takes the value true (given the values of its parents); the conditional probability that the random variable is false is simply 1 minus the probability that it is true.

the probability of different values for X_i. These are often referred to as *table CPD*s, and are tables of multinomial distributions. Other possibilities are to represent the distributions via a tree structure (called, appropriately enough, *tree-structured CPDs*), or using an even more compact representation such as a *noisy-OR* or *noisy-MAX*.

Example 2.1

Consider the simple Bayesian network shown in figure 2.1. This is a toy example indicating the interactions between two potential diseases, pneumonia and tuberculosis. Both of them may cause a patient to have lung infiltrates. There are two tests that can be performed. An x-ray can be taken, which may indicate whether the patient has lung infiltrates. There is a separate sputum smear test for tuberculosis. figure 2.1(a) shows the dependency structure among the variables. All of the variables are assumed to be Boolean. figure 2.1(b) shows the conditional probability distributions for each of the random variables. We use initials P, T, I, X, and S for shorthand. At the roots, we have the prior probability of the patient having each disease. The probability that the patient does not have the disease a priori is simply 1 minus the probability he or she has the disease; for simplicity only the probabilities for the true case are shown. Similarly, the conditional probabilities for the non-root nodes give the probability that the random variable is true, for different possible instantiations of the parents.

Definition 2.2
Let $\mathcal{G}$ be a Bayesinan network graph over the variables $X_1, \ldots, X_n$. We say that a distribution $P_{\mathcal{B}}$ over the same space *factorizes according to* $\mathcal{G}$ if $P_{\mathcal{B}}$ can be expressed as a product

$$P_{\mathcal{B}}(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \mathbf{Pa}_{X_i}). \tag{2.1}$$

A *Bayesian network* is a pair $(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ where $P_{\mathcal{B}}$ factorizes over $\mathcal{G}$, and where $P_{\mathcal{B}}$ is specified as set of CPDs associated with $\mathcal{G}$'s nodes, denoted $\boldsymbol{\theta}_{\mathcal{G}}$. ∎

The equation above is called the *chain rule for Bayesian networks*. It gives us a method for determining the probability of any complete assignment to the set of random variables: any entry in the joint can be computed as a product of factors, one for each variable. Each factor represents a conditional probability of the variable given its parents in the network.

Example 2.2
The Bayesian network in figure 2.1(a) describes the following factorization:

$$P(P, T, I, X, S) = P(P)P(T)P(I \mid P, T)P(X \mid I)P(S \mid T).$$

Sometimes it is useful to think of the Bayesian network as describing a generative process. We can view the graph as encoding a generative sampling process executed by nature, where the value for each variable is selected by nature using a distribution that depends only on its parents. In other words, each variable is a stochastic function of its parents.

2.2.2 Conditional Independence Assumptions in Bayesian Networks

Another way to view a Bayesian network is as a compact representation for a set of conditional independence assumptions about a distribution. These conditional independence assumptions are called the *local Markov assumptions*. While we won't go into the full details here, this view is, in a strong sense, equivalent to the view of the Bayesian network as providing a factorization of the distribution.

Definition 2.3
Given a BN network structure $\mathcal{G}$ over random variables $X_1, \ldots, X_n$, let $NonDescendants_{X_i}$ denote the variables in the graph that are not descendants of X_i. Then $\mathcal{G}$ encodes the following set of conditional independence assumptions, called the *local Markov assumptions*:

> *For each variable X_i, we have that*
>
> $$(X_i \perp \text{NonDescendants}_{X_i} \mid \mathbf{Pa}_{X_i}),$$
>
> *In other words, the local Markov assumptions state that each node X_i is independent of its nondescendants given its parents.* ∎

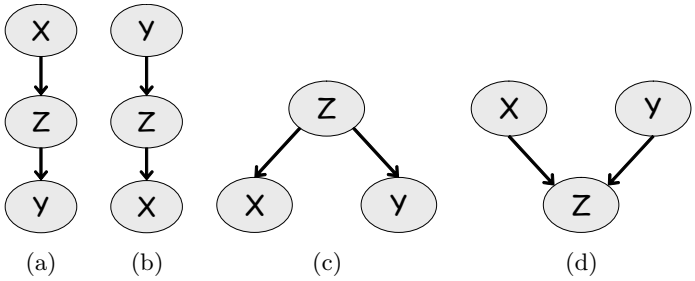


Figure 2.2 (a) An indirect causal effect; (b) an indirect evidential effect; (c) a common cause; (d) a common effect.

Example 2.3
The BN in figure 2.1(a) describes the following local Markov assumptions: $(P \perp T \mid \emptyset), (T \perp P \mid \emptyset), (X \perp \{P, T, S\} \mid I)$, and $(S \perp \{P, I, X\} \mid T)$.

These are not the only independence assertions that are encoded by a network. A general procedure called *d-separation* (which stands for directed separation) can answer whether an independence assertion *must* hold in *any* distribution consistent with the graph $\mathcal{G}$. However, note that other independencies may hold in *some* distributions consistent with $\mathcal{G}$; these are due to flukes in the particular choice of parameters of the network (and this is why they hold in some of the distributions).

Returning to our definition of d-separation, it is useful to view probabilistic influence as a flow in the graph. Our analysis here tells us when influence from X can "flow" through Z to affect our beliefs about Y. We will consider flow allows (undirected) paths in the graph.

Consider a simple three-node path X—Y—Z If influence can flow from X to Y via Z, we say that the path X—Z—Y is *active*. There are four cases:

- **Causal path** $X \rightarrow Z \rightarrow Y$**:** active if and only if Z is not observed.
- **Evidential path** $X \leftarrow Z \leftarrow Y$**:** active if and only if Z is not observed.
- **Common cause** $X \leftarrow Z \rightarrow Y$**:** active if and only if Z is not observed.
- **Common effect** $X \rightarrow Z \leftarrow Y$**:** active if and only if either Z or one of Z's descendants is observed.

A structure where $X \rightarrow Z \leftarrow Y$ (as in figure 2.2(d)) is also called a *v-structure*.

Example 2.4
In the BN from figure 2.1(a), the path from $P \rightarrow I \rightarrow X$ is active if I is not observed. On the other hand, the path from $P \rightarrow I \leftarrow T$ is active if I *is* observed.

Now consider a longer path X_1—$\cdots$—X_n. Intuitively, for influence to "flow" from X_1 to X_n, it needs to flow through every single node on the trail. In other words, X_1 can influence X_n if every two-edge path X_{i-1}—X_i—X_{i+1} along the trail allows influence to flow. We can summarize this intuition in the following definition:

Definition 2.4
Let $\mathcal{G}$ be a BN structure, and $X_1 — \ldots — X_n$ a path in $\mathcal{G}$. Let $\boldsymbol{E}$ be a subset of nodes of $\mathcal{G}$. The path $X_1 — \ldots — X_n$ is *active* given evidence $\boldsymbol{E}$ if

- whenever we have 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 $\boldsymbol{E}$. ∎

Our flow intuition carries through to graphs in which there is more than one path between two nodes: one node can influence another if there is any path along which influence can flow. Putting these intuitions together, we obtain the notion of *d-separation*, which provides us with a notion of separation between nodes in a directed graph (hence the term d-separation, for directed separation):

Definition 2.5
Let $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$ be three sets of nodes in $\mathcal{G}$. We say that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are *d-separated given* $\boldsymbol{Z}$, denoted $d\text{-}sep_{\mathcal{G}}(\boldsymbol{X}; \boldsymbol{Y} \mid \boldsymbol{Z})$, if there is no active path between any node $X \in \boldsymbol{X}$ and $Y \in \boldsymbol{Y}$ given $\boldsymbol{Z}$. ∎

Finally, an important theorem which relates the independencies which hold in a distribution to the factorization of a distribution is the following:

Theorem 2.6
Let $\mathcal{G}$ be a BN graph over a set of random variables $\mathcal{X}$ and let P be a joint distribution over the same space. If all the local Markov properties associated with $\mathcal{G}$ hold in P, then P factorizes according to $\mathcal{G}$.

Theorem 2.7
Let $\mathcal{G}$ be a BN graph over a set of random variables $\mathcal{X}$ and let P be a joint distribution over the same space. If P factorizes according to $\mathcal{G}$, then all the local Markov properties associated with $\mathcal{G}$ hold in P.

2.2.3 Markov Networks

The second common class of probabilistic graphical models is called a *Markov network* or a *Markov random field*. The models are based on undirected graphical models. These models are useful in modeling a variety of phenomena where one cannot naturally ascribe a directionality to the interaction between variables. Furthermore, the undirected models also offer a different and often simpler perspective on directed models, both in terms of the independence structure and the inference task.

A representation that implements this intuition is that of an undirected graph. As in a Bayesian network, the nodes in the graph of a *Markov network graph* $\mathcal{H}$ represent the variables, and the edges correspond to some notion of direct probabilistic interaction between the neighboring variables.

The remaining question is how to parameterize this undirected graph. The graph structure represents the qualitative properties of the distribution. To represent the

distribution, we need to associate the graph structure with a set of parameters, in the same way that CPDs were used to parameterize the directed graph structure. However, the parameterization of Markov networks is not as intuitive as that of Bayesian networks, as the factors do not correspond either to probabilities or to conditional probabilities.

The most general parameterization is a *factor*:

Definition 2.8
Let $\boldsymbol{D}$ be a set of random variables. We define a *factor* to be a function from $Val(\boldsymbol{D})$ to $\mathbb{R}^+$. ∎

Definition 2.9
Let $\mathcal{H}$ be a Markov network structure. A distribution $P_{\mathcal{H}}$ *factorizes* over $\mathcal{H}$ if it is associated with

- a set of subsets $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_m$, where each $\boldsymbol{D}_i$ is a complete subgraph of $\mathcal{H}$;
- factors $\pi_1[\boldsymbol{D}_1], \ldots, \pi_m[\boldsymbol{D}_m]$,

such that

$$P_{\mathcal{H}}(X_1, \ldots, X_n) = \frac{1}{Z} P'(X_1, \ldots, X_n),$$

where

$$P'_{\mathcal{H}}(X_1, \ldots, X_n) = \pi_i[\boldsymbol{D}_1] \times \pi_2[\boldsymbol{D}_2] \times \cdots \times \pi_m[\boldsymbol{D}_m]$$

is an unnormalized measure and

$$Z = \sum_{X_1, \ldots, X_n} P'_{\mathcal{H}}(X_1, \ldots, X_n)$$

is a normalizing constant called the *partition function*. A distribution P that factorizes over $\mathcal{H}$ is also called a *Gibbs distribution* over $\mathcal{H}$. (The naming convention has roots in statistical physics.) ∎

Note that this definition is quite similar to the factorization definition for Bayesian networks: There, we decomposed the distribution as a product of CPDs. In the case of Markov networks, the only constraint on the parameters in the factor is non-negativity.

As every complete subgraph is a subset of some clique, we can simplify the parameterization by introducing factors only for cliques, rather than for subcliques. More precisely, let $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k$ be the cliques in $\mathcal{H}$. We can parameterize P using a set of factors $\pi_1[\boldsymbol{C}_1], \ldots, \pi_k[\boldsymbol{C}_k]$. These factors are called *clique potentials* (in the context of the Markov network $\mathcal{H}$). It is tempting to think of the clique potentials as representing the marginal probabilities of the variables in their scope. However, this is incorrect. It is important to note that, although conceptually somewhat simpler, the parameterization using clique potentials can obscure the structure that

is present in the original parameterization, and can possibly lead to an exponential increase in the size of the representation.

It is often useful to consider a slightly different way of specifying potentials, by using a logarithmic transformation. In particular, we can rewrite a factor $\pi[\boldsymbol{D}]$ as

$$\pi[\boldsymbol{D}] = \exp(-\epsilon[\boldsymbol{D}]),$$

where $\epsilon[\boldsymbol{D}] = -\ln \pi[\boldsymbol{D}]$ is often called an *energy function*. The use of the word "energy" derives from statistical physics, where the probability of a physical state (e.g., a configuration of a set of electrons), depends inversely on its energy.

In this logarithmic representation, we have that

$$P_{\mathcal{H}}(X_1, \ldots, X_n) \propto \exp\left[-\sum_{i=1}^{m} \epsilon_i[\boldsymbol{D}_i]\right].$$

The logarithmic representation ensures that the probability distribution is positive. Moreover, the logarithmic parameters can take any real value.

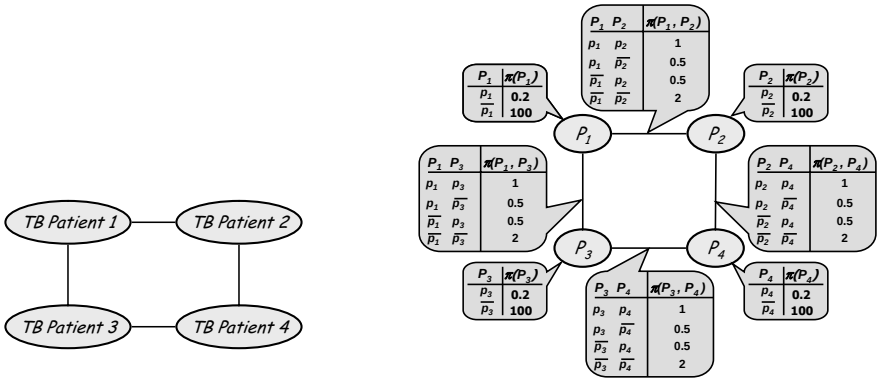
A subclass of Markov networks that arises in many contexts is that of *pairwise Markov networks*, representing distributions where all of the factors are over single variables or pairs of variables. More precisely, a pairwise Markov network over a graph $\mathcal{H}$ is associated with a set of *node potentials* $\{\pi[X_i] : i = 1, \ldots, n\}$ and a set of *edge potentials* $\{\pi[X_i, X_j] : (X_i, X_j) \in \mathcal{H}\}$. The overall distribution is (as always) the normalized product of all of the potentials (both node and edge). Pairwise MRFs are attractive because of their simplicity, and because interactions on edges are an important special case that often arises in practice.

Example 2.5

Figure 2.3(a) shows a simple Markov network. This toy example has random variables describing the tuberculosis status of four patients. Patients that have been in contact are linked by undirected edges. The edges indicate the possibilities for the disease transmission. For example, *Patient* 1 has been in contact with *Patient* 2 and *Patient* 3, but has not been in contact with *Patient* 4. figure 2.3(b) shows the same Markov network, along with the node and edge potentials. We use $P1$, $P2$, $P3$, and $P4$ for shorthand. In this case, all of the node and edge potentials are the same, but this is not a requirement. The node potentials show that the patients are much more likely to be uninfected. The edge potentials capture the intuition that it is most likely for two people to have the same infection state — either both infected, or both not. Furthermore, it is more likely that they are both not infected.

2.2.4 Independencies in Markov Networks

As in the case of Bayesian networks, the graph structure in a Markov network can be viewed as encoding a set of independence assumptions. Intuitively, in Markov networks, probabilistic influence "flows" along the undirected paths in the graph, but is blocked if we condition on the intervening nodes. We can define two sets



(a) (b)

Figure 2.3 (a) A simple Markov network describing the tuberculosis status of four patients. The links between patients indicate which patients have been in contact with each other. (b) The same Markov network, together with the node and edge potentials.

of independence assumptions, the local Markov properties and the global Markov properties.

The local Markov properties are associated with each node in the graph and are based on the intuition that we can block all influences on a node by conditioning on its immediate neighbors.

Definition 2.10
Let $\mathcal{H}$ be an undirected graph. Then for each node $X \in \mathcal{X}$, the Markov blanket of X, denoted $\mathcal{N}_{\mathcal{H}}(X)$, is the set of neighbors of X in the graph (those that share an edge with X). We define the *local Markov independencies* associated with $\mathcal{H}$ to be

$$\mathcal{I}_{\ell}(\mathcal{H}) = \{(X \perp \mathcal{X} - \{X\} - \mathcal{N}_{\mathcal{H}}(X) \mid \mathcal{N}_{\mathcal{H}}(X)) \ : \ X \in \mathcal{X}\}.$$

In other words, the Markov assumptions state that X is independent of the rest of the nodes in the graph given its immediate neighbors. ∎

Example 2.6
The MN in figure 2.3(a) describes the following local Markov assumptions: $(P_1 \perp P_4 \mid \{P_2, P_3\})$, $(P_2 \perp P_3 \mid \{P_1, P_4\})$, $(P_3 \perp P_2 \mid \{P_1, P_4\})$, $(P_4 \perp P_1 \mid \{P_2, P_3\})$.

To define the global Markov properties, we begin by defining active paths in undirected graphs.

Definition 2.11
Let $\mathcal{H}$ be a Markov network structure, and $X_1 — \ldots — X_k$ be a path in $\mathcal{H}$. Let $\boldsymbol{E} \subseteq \mathcal{X}$ be a set of *observed variables*. The path $X_1 — \ldots — X_k$ is *active* given $\boldsymbol{E}$ if none of the X_i's, $i = 1, \ldots, k$, is in $\boldsymbol{E}$. ∎

Using this notion, we can define a notion of *separation* in the undirected graph. This is the analogue of *d-separation*; note how much simpler it is.

Definition 2.12
We say that a set of nodes $\boldsymbol{Z}$ *separates* $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathcal{H}$, denoted $sep_{\mathcal{H}}(\boldsymbol{X}; \boldsymbol{Y} \mid \boldsymbol{Z})$, if there is no active path between any node $X \in \boldsymbol{X}$ and $Y \in \boldsymbol{Y}$ given $\boldsymbol{Z}$. We define the *global Markov assumptions* associated with $\mathcal{H}$ to be

$$\mathcal{I}(\mathcal{H}) = \{(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z}) \ : \ sep_{\mathcal{H}}(\boldsymbol{X}; \boldsymbol{Y} \mid \boldsymbol{Z})\}. \ \blacksquare$$

As in the case of Bayesian networks, we can make a connection between the local Markov properties and the global Markov properties. The assumptions are in fact equivalent, but only for positive distributions. (Informally, a distribution is positive if every possible joint instantiation has probability > 0.)

We begin with the analogue to theorem 2.7, which asserts that a Gibbs distribution satisfies the global independencies associated with the graph.

Theorem 2.13
Let P be a distribution over $\mathcal{X}$, and $\mathcal{H}$ a Markov network structure over $\mathcal{X}$. If P is a Gibbs distribution over $\mathcal{H}$, then all the local Markov properties associated with $\mathcal{H}$ hold in P.

The other direction, which goes from the global independence properties of a distribution to its factorization, is known as the *Hammersley-Clifford theorem*. Unlike for Bayesian networks, this direction does not hold in general. It only holds under the additional assumption that P is a positive distribution.

Theorem 2.14
Let P be a positive distribution over $\mathcal{X}$, and $\mathcal{H}$ a Markov network graph over $\mathcal{X}$. If all of the independence constraints implied by $\mathcal{H}$ hold in P, then P is a Gibbs distribution over $\mathcal{H}$.

This result shows that, for positive distributions, the global Markov property implies that the distribution factorizes according to the network structure. Thus, for this class of distributions, we have that a distribution P factorizes over a Markov network $\mathcal{H}$ if and only if all of the independencies implied by $\mathcal{H}$ hold in P. The positivity assumption is necessary for this result to hold.

2.3 Inference

Both directed and undirected graphical models represent a full joint probability distribution over $\mathcal{X}$. We describe some of the main query types one might expect to answer with a joint distribution, and discuss the computational complexity of answering such queries using a graphical model.

The most common query type is the standard *conditional probability query*, $P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e})$. Such a query consists of two parts: the *evidence*, a subset $\boldsymbol{E}$ of

random variables in the network, and an instantiation $\boldsymbol{e}$ to these variables; and the *query*, a subset $\boldsymbol{Y}$ of random variables in the network. Our task is to compute $P(\boldsymbol{Y} \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{P(\boldsymbol{Y}, \boldsymbol{e})}{P(\boldsymbol{e})}$, i.e., the probability distribution over the values $\boldsymbol{y}$ of $\boldsymbol{Y}$, conditioned on the fact that $\boldsymbol{E} = \boldsymbol{e}$.

Another type of query that often arises is that of finding the *most probable assignment* to some subset of variables. As with conditional probability queries, we have evidence $\boldsymbol{E} = \boldsymbol{e}$. In this case, however, we are trying to compute the most likely assignment to some subset of the remaining variables. This problem has two variants, where the first variant is an important special case of the second. The simplest variant of this task is the *most probable explanation (MPE)* queries. An MPE query tries to find the most likely assignment to all of the (non-evidence) variables. More precisely, if we let $\boldsymbol{W} = \mathcal{X} - \boldsymbol{E}$, our task is to find the most likely assignment to the variables in $\boldsymbol{W}$ given the evidence $\boldsymbol{E} = \boldsymbol{e}$: $\text{argmax}_{\boldsymbol{w}} P(\boldsymbol{w}, \boldsymbol{e})$, where, in general, $\text{argmax}_x f(x)$ represents the value of x for which $f(x)$ is maximal. Note that there might be more than one assignment that has the highest posterior probability. In this case, we can either decide that the MPE task is to return the set of possible assignments, or to return an arbitrary member of that set.

In the second variant, the *maximum a posteriori (MAP)* query, we have a subset of variables $\boldsymbol{Y}$ which forms our query. The task is to find the most likely assignment to the variables in $\boldsymbol{Y}$ given the evidence $\boldsymbol{E} = \boldsymbol{e}$: $\text{argmax}_{\boldsymbol{y}} P(\boldsymbol{y} \mid \boldsymbol{e})$. This class of queries is clearly more general than MPE queries, so it might not be clear why the class of MPE queries is sufficiently interesting to consider as a special case. The difference becomes clearer if we explicitly write out the expression for a general MAP query. If we let $\boldsymbol{Z} = \mathcal{X} - \boldsymbol{Y} - \boldsymbol{E}$, the MAP task is to compute: $\text{argmax}_{\boldsymbol{Y}} \sum_{\boldsymbol{Z}} P(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{e})$. MAP queries contain both summations and maximizations; in a way, they contain elements of both a conditional probability query and an MPE query. This combination makes the MAP task harder than either of these other tasks. In particular, there are techniques and analysis for the MPE task that do not generalize to the MAP task. This observation, combined with the fact that the MPE case is reasonably common, makes it worthwhile to consider MPE as a separate task. Note that in statistics literature, as well as in some work on graphical models, the term MAP is often used to mean MPE, but the distinction can be made clear from the context.

In principle, a graphical model can be used to answer all of the query types described above. We simply generate the joint distribution, and exhaustively sum out the joint (in the case of a conditional probability query), search for the most likely entry (in the case of an MPE query), or both (in the case of an MAP query). However, this approach to the inference problem is not very satisfactory, as it results in the exponential blowup of the joint distribution that the graphical model representation was precisely designed to avoid.

We assume that we are dealing with a set of factors $\mathcal{F}$ over a set of variables $\mathcal{X}$. This set of factors defines a possibly unnormalized function

$$P_{\mathcal{F}}(\mathcal{X}) = \prod_{\phi \in \mathcal{F}} \phi. \tag{2.2}$$

For a Bayesian network without evidence, the factors are simply the CPDs, and the distribution $P_{\mathcal{F}}$ is a normalized distribution. For a Bayesian network $\mathcal{B}$ with evidence $\boldsymbol{E} = \boldsymbol{e}$, the factors are the CPDs restricted to $\boldsymbol{e}$, and $P_{\mathcal{F}}(\mathcal{X}) = P_{\mathcal{B}}(\mathcal{X}, \boldsymbol{e})$. For a Markov network $\mathcal{H}$ (with or without evidence), the factors are the (restricted) compatibility potentials, and $P_{\mathcal{F}}$ is the unnormalized distribution $P'_{\mathcal{H}}$ before dividing by the partition function. It is important to note, however, that most of the operations that one can perform on a normalized distribution can also be performed on an unnormalized one. Thus, we can marginalize $P_{\mathcal{F}}$ on a subset of the variables by summing out the others. We can also consider a conditional probability $P_{\mathcal{F}}(\boldsymbol{X} \mid \boldsymbol{Y}) = P_{\mathcal{F}}(\boldsymbol{X}, \boldsymbol{Y})/P_{\mathcal{F}}(\boldsymbol{Y})$. Thus, for the purposes of this section, we treat $P_{\mathcal{F}}$ as a distribution, ignoring the fact that it may not be normalized.

In the worst case, the complexity of probabilistic inference is unavoidable. Below, we assume that the set of factors $\{\phi \in \mathcal{F}\}$ of the graphical model defining the desired distribution can be specified in a polynomial number of bits (in terms of the number of variables).

Theorem 2.15
The following decision problems are $\mathcal{NP}$-complete:

- Given a distribution $P_{\mathcal{F}}$ over $\mathcal{X}$, a variable $X \in \mathcal{X}$, and a value $x \in \mathit{Val}(X)$, decide whether $P_{\mathcal{F}}(X = x) > 0$.
- Given a distribution $P_{\mathcal{F}}$ over $\mathcal{X}$ and a number τ, decide whether there exists an assignment $\boldsymbol{x}$ to $\mathcal{X}$ such that $P_{\mathcal{F}}(\boldsymbol{x}) > \tau$.

The following problem is $\#\mathcal{P}$-complete:

- Given a distribution $P_{\mathcal{F}}$ over $\mathcal{X}$, a variable $X \in \mathcal{X}$, and a value $x \in \mathit{Val}(X)$, compute $P_{\mathcal{F}}(X = x)$.

These results seem like very bad news: every type of inference in graphical models is $\mathcal{NP}$-hard or harder. In fact, even the simple problem of computing the distribution over a single binary variable is $\mathcal{NP}$-hard. Assuming (as seems increasingly likely) that the best computational performance we can achieve for $\mathcal{NP}$-hard problems is exponential in the worst case, there seems to be no hope for efficient algorithms for even the simplest type of inference. However, as we discuss below, the worst-case blowup can often be avoided. For all other models, we will resort to approximate inference techniques. Note that the worst-case results for approximate inference are also negative:

Theorem 2.16
The following problem is $\mathcal{NP}$-hard for any $\epsilon \in (0, 1/2)$: Given a distribution $P_{\mathcal{F}}$ over $\mathcal{X}$, a variable $X \in \mathcal{X}$, and a value $x \in Val(X)$, find a number τ, such that $|P_{\mathcal{F}}(X = x) - \tau| \leq \epsilon$.

Fortunately, many types of exact inference can be performed efficiently for a very important class of graphical models (low treewidth) we define below. For a large number of models, however, exact inference is intractable and we resort to approximations. Broadly speaking, there are two major frameworks for probabilistic inference: optimization-based and sampling-based. Exact inference algorithms have been historically derived from the dynamic programming perspective, by carefully avoiding repeated computations. We take a somewhat unconventional approach here by presenting exact and approximate inference in a unified optimization framework. We thus start out by considering approximate inference and then present conditions under which it yields exact results.

2.3.1 Inference as Optimization

The methods that fall into an optimization framework are based on a simple conceptual principle: define a target class of "easy" distributions $\boldsymbol{Q}$, and then search for a particular instance Q within that class which is the "best" approximation to $P_{\mathcal{F}}$. Queries can then be answered using inference on Q rather than on $P_{\mathcal{F}}$. The specific algorithms that have been considered in the literature differ in many details. However, most of them can be viewed as optimizing a target function for measuring the quality of approximation.

Suppose that we want to approximate $P_{\mathcal{F}}$ with another distribution Q. Intuitively, we want to choose the approximation Q to be close to $P_{\mathcal{F}}$. There are many possible ways to measure the distance between two distributions, such as the Euclidean distance (L_2), or the L_1 distance. Our main challenge, however, is that our aim is to avoid performing inference with the distribution $P_{\mathcal{F}}$; in particular, we cannot effectively compute marginal distributions in $P_{\mathcal{F}}$. Hence, we need methods that allow us to optimize the distance (technically, divergence) between Q and $P_{\mathcal{F}}$ without answering hard queries in $P_{\mathcal{F}}$. A priori, this requirement may seem impossible to satisfy. However, it turns out that there exists a distance measure — the relative entropy (or KL-divergence) — that allows us to exploit the structure of $P_{\mathcal{F}}$ without performing reasoning with it.

Recall that the relative entropy between P_1 and P_2 is defined as $\boldsymbol{D}(P_1 \| P_2) = \boldsymbol{E}_{P_1}\left[\ln \frac{P_1(\mathcal{X})}{P_2(\mathcal{X})}\right]$. The relative entropy is always non-negative, and equal to 0 if and only if $P_1 = P_2$. Thus, we can use it as a distance measure, and choose to find an approximation Q to $P_{\mathcal{F}}$ that minimizes the relative entropy. However, the relative entropy is not symmetric — $\boldsymbol{D}(P_1 \| P_2) \neq \boldsymbol{D}(P_2 \| P_1)$. A priori, it might appear that $\boldsymbol{D}(P_{\mathcal{F}} \| Q)$ is a more appropriate measure for approximate inference, as one of the main information-theoretic justifications for relative entropy is the number of bits lost when coding a true message distribution $P_{\mathcal{F}}$ using an (approximate) estimate Q.

However, computing the so-called M-projection Q of $P_{\mathcal{F}}$ — the $\text{argmin}_Q \boldsymbol{D}(P_{\mathcal{F}} \| Q)$ — is actually equivalent to running inference in $P_{\mathcal{F}}$. Somewhat surprisingly, as we show in the subsequent discussion, this does not apply to the so-called I-projection: we can exploit the structure of $P_{\mathcal{F}}$ to optimize $\text{argmin}_Q \boldsymbol{D}(Q \| P_{\mathcal{F}})$ efficiently, *without* running inference in $P_{\mathcal{F}}$.

An additional reason for using relative entropy as our distance measure is based on the following result, which relates the relative entropy $\boldsymbol{D}(Q \| P_{\mathcal{F}})$ with the partition function Z:

Proposition 2.17

$$\ln Z = F[P_{\mathcal{F}}, Q] + \boldsymbol{D}(Q \| P_{\mathcal{F}}), \tag{2.3}$$

where $F[P_{\mathcal{F}}, Q]$ is the *energy functional* $F[P_{\mathcal{F}}, Q] = \sum_{\phi \in \mathcal{F}} \boldsymbol{E}_Q[\ln \phi] + \boldsymbol{H}_Q(\mathcal{X})$.

This proposition has several important ramifications. Note that the term $\ln Z$ does not depend on Q. Hence, minimizing the relative entropy $\boldsymbol{D}(Q \| P_{\mathcal{F}})$ is equivalent to maximizing the energy functional $F[P_{\mathcal{F}}, Q]$. This latter term relates to concepts from statistical physics, and it is the negative of what is referred to in that field as the *Helmholtz free energy*. While explaining the physics-based motivation for this term is out of the scope of this chapter, we continue to use the standard terminology of energy functional.

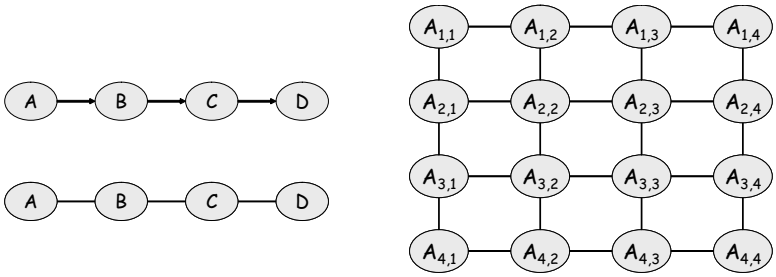
In the remainder of this section, we pose the problem of finding a good approximation Q as one of maximizing the energy functional, or, equivalently, minimizing the relative entropy. Importantly, the energy functional involves expectations in Q. As we show, by choosing approximations Q that allow for efficient inference, we can both evaluate the energy functional and optimize it effectively.

Moreover, as $\boldsymbol{D}(Q \| P_{\mathcal{F}}) \geq 0$, we have that $\ln Z \geq F[P_{\mathcal{F}}, Q]$. That is, the energy functional is a *lower bound* on the value of the logarithm of the partition function Z, for any choice of Q. Why is this fact significant? Recall that, in directed models, the partition function Z is the probability of the evidence. Computing the partition function is often the hardest part of inference. And so this theorem shows that if we have a good approximation (that is, $\boldsymbol{D}(Q \| P_{\mathcal{F}})$ is small), then we can get a good lower bound approximation to Z. The fact that this approximation is a lower bound plays an important role in learning parameters of graphical models.

2.3.2 Exact Inference as Optimization

Before considering approximate inference methods, we illustrate the use of a variational approach to derive an exact inference procedure. The concepts we introduce here will serve in discussion of the following approximate inference methods.

The goal of exact inference here will be to compute marginals of the distribution. To achieve this goal, we will need to make sure that the set of distributions $\boldsymbol{Q}$ is expressive enough to represent the target distribution $P_{\mathcal{F}}$. Instead of approximating $P_{\mathcal{F}}$, the solution of the optimization problem transforms the representation of the



(a) (b)

Figure 2.4 (a) Chain-structured Bayesian network and equivalent Markov network (b) Grid-structured Markov network.

distribution from a product of factors into a more useful form Q that directly yields the desired marginals.

To accomplish this, we will need to optimize over the set of distributions $\boldsymbol{Q}$ that include $P_{\mathcal{F}}$. Then, if we search over this set, we are guaranteed to find a distribution Q^* for which $\boldsymbol{I\!D}(Q^* \| P_{\mathcal{F}}) = 0$, which is therefore the unique global optimum of our energy functional. We will represent this set using an undirected graphical model called the clique tree, for reasons that will be clear below.

Consider the undirected graph corresponding to the set of factors $\mathcal{F}$. In this graph, nodes are connected if they appear together in a factor. Note that if a factor is the CPD of a directed graphical model, then the family will be a clique in the graph, so its connectivity is denser then the original directed graph since parents have been connected (moralized). The key property for exact inference in the graph is chordality:

Definition 2.18
Let X_1—X_2—$\cdots$—X_k—X_1 be a loop in the graph; a *chord* in the loop is an edge connecting X_i and X_j for two nonconsecutive nodes X_i, X_j. An undirected graph $\mathcal{H}$ is said to be *chordal* if any loop X_1—X_2—$\cdots$—X_k—X_1 for $k \geq 4$ has a chord. ∎

In other words, the longest "minimal loop" (one that has no shortcut) is a triangle. Thus, chordal graphs are often also called *triangulated*.

The simplest (and most commonly used) chordal graphs are chain-structured (see figure 2.4(a)). What if the graph is not chordal? For example, grid-structured graphs are commonly used in computer vision for pixel-labeling problems (see figure 2.4(b)). To make a graph chordal (triangulate it), *fill-in* edges are added to short-circuit loops. There are generally many ways to do this and finding the least number of edges to fill is $\mathcal{NP}$-hard. However, good heuristic algorithms for this problem exist [12, 1].

We now define a *cluster graph* — the backbone of the graphical data structure needed to perform inference. Each node in the cluster graph is a *cluster*, which

is associated with a subset of variables; the graph contains undirected edges that connect clusters whose scopes have some nonempty intersection.

Definition 2.19
A *cluster graph* $\mathcal{K}$ for a set of factors $\mathcal{F}$ over $\mathcal{X}$ is an undirected graph, each of whose nodes i is associated with a subset $\boldsymbol{C}_i \subseteq \mathcal{X}$. A cluster graph must be *family-preserving* — each factor $\phi \in \mathcal{F}$ must be associated with a cluster $\boldsymbol{C}$, denoted $\alpha(\phi)$, such that $Scope[\phi] \subseteq \boldsymbol{C}_i$. Each edge between a pair of clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ is associated with a *sepset* $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$. A singly connected cluster graph (a tree) is called a *cluster tree.* ∎

Definition 2.20
Let $\mathcal{T}$ be a cluster tree over a set of factors $\mathcal{F}$. We say that $\mathcal{T}$ has the *running intersection property* if, whenever there is a variable X such that $X \in \boldsymbol{C}_i$ and $X \in \boldsymbol{C}_j$, then X is also in every cluster in the (unique) path in $\mathcal{T}$ between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$. A cluster tree that satisfies the running intersection property is called a *clique tree.* ∎

Theorem 2.21
Every chordal graph $\mathcal{G}$ has a clique tree $\mathcal{T}$.

Constructing the clique tree from a chordal graph is actually relatively easy: (1) find maximal cliques of the graph (this is easy in chordal graphs) and (2) run a maximum spanning tree algorithm on the appropriate clique graph. More specifically, we build an undirected graph whose nodes are the maximal cliques, and where every pair of nodes $\boldsymbol{C}_i, \boldsymbol{C}_j$ is connected by an edge whose *weight* is $|\boldsymbol{C}_i \cap \boldsymbol{C}_j|$.

Because of this correspondence, we can define a very important characteristic of a graph, which is critical to the complexity of exact inference:

Definition 2.22
The *treewidth* of a chordal graph is the size of the largest clique minus 1. The *treewidth* of an untriangulated graph is the minimum treewidth of all of its triangulations. ∎

Note that the treewidth of a chain in figure 2.4(a) is 1 and the treewidth of the grid in figure 2.4(b) is 4.

2.3.2.1 The Optimization Problem

Suppose we are given a clique tree $\mathcal{T}$ for $P_{\mathcal{F}}$. That is, $\mathcal{T}$ satisfies the running intersection property and the family preservation property. Moreover, suppose we are given a set of potentials $\boldsymbol{Q} = \{\pi_i\} \cup \{\mu_{i,j} : (\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}\}$, where $\boldsymbol{C}_i$ denotes clusters in $\mathcal{T}$, $\boldsymbol{S}_{i,j}$ denote separators along edges in $\mathcal{T}$, π_i is a potential over $\boldsymbol{C}_i$, and $\mu_{i,j}$ is a potential over $\boldsymbol{S}_{i,j}$. The set of potentials defines a distribution Q according

to $\mathcal{T}$ by the formula

$$Q(\mathcal{X}) = \frac{\prod_{\boldsymbol{C}_i \in \mathcal{T}} \pi_i}{\prod_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}} \mu_{i,j}}. \tag{2.4}$$

Note that by construction, $\boldsymbol{Q}$ can represent $P_{\mathcal{F}}$ by simply letting the appropriate potentials π_i equal factors ϕ_i and letting $\mu_{i,j}$ equal 1. However, we will consider a different, more useful, representation.

Definition 2.23
The set of potentials $\boldsymbol{Q}$ is *calibrated* when for each $(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}$ the potential $\mu_{i,j}$ on $\boldsymbol{S}_{i,j}$ is the marginal of π_i (and π_j). ∎

Proposition 2.24
Let $\boldsymbol{Q}$ be a set of calibrated potentials for $\mathcal{T}$, and let Q be the distribution defined by (2.4). Then $\pi_i[\boldsymbol{c}_i] = Q(\boldsymbol{c}_i)$ and $\mu_{i,j}[\boldsymbol{s}_{i,j}] = Q(\boldsymbol{s}_{i,j})$.

In other words, the potentials correspond to marginals of the distribution Q defined by (2.4). Now if $\boldsymbol{Q}$ is a set of *uncalibrated* potentials for $\mathcal{T}$, and Q is the distribution defined by (2.4), we can construct $\boldsymbol{Q}'$, a set of *calibrated potentials* which represent Q by simply using the appropriate marginals of Q.

Once we decide to focus our attention on calibrated clique trees, we can rewrite the energy functional in a factored form, as a sum of terms each of which depends directly only on one of the potentials in $\boldsymbol{Q}$. This form reveals the structure in the distribution, and is therefore a much better starting point for further analysis. As we shall see, this form will also be the basis for our approximations in subsequent sections.

Definition 2.25
Given a clique tree $\mathcal{T}$ with a set of potentials, $\boldsymbol{Q}$, and an assignment α that maps factors in $P_{\mathcal{F}}$ to clusters in $\mathcal{T}$, we define the *factored energy functional*

$$\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}] = \sum_i \boldsymbol{E}_{\pi_i}\left[\ln \pi_i^0\right] + \sum_{\boldsymbol{C}_i \in \mathcal{T}} \boldsymbol{H}_{\pi_i}(\boldsymbol{C}_i) - \sum_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}), \tag{2.5}$$

where $\pi_i^0 = \prod_{\phi, \alpha(\phi)=i} \phi$. ∎

Before we prove that the energy functional is equivalent to its factored form, let us first understand its form. The first term is a sum of terms of the form $\boldsymbol{E}_{\pi_i}\left[\ln \pi_i^0\right]$. Recall that π_i^0 is a factor (not necessarily a distribution) over the scope $\boldsymbol{C}_i$, that is, a function from $Val(\boldsymbol{C}_i)$ to $\mathbb{R}^+$. Its logarithm is therefore a function from $Val(\boldsymbol{C}_i)$ to $\mathbb{R}$. The clique potential π_i is a distribution over $Val(\boldsymbol{C}_i)$. We can therefore compute the expectation, $\sum_{\boldsymbol{c}_i} \pi_i[\boldsymbol{c}_i] \ln \pi_i^0$. The last two terms are entropies of the distributions — the potentials and messages — associated with the clusters and sepsets in the tree.

Proposition 2.26
If $\boldsymbol{Q}$ is a set of calibrated potentials for $\mathcal{T}$, and Q is defined by by (2.4), then

$$\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}] = F[P_{\mathcal{F}}, Q].$$

Using this form of the energy, we can now define the optimization problem. We first need to define the space over which we are optimizing. If Q is factorized according to $\mathcal{T}$, we can represent it by a set of calibrated potentials. Calibration is essentially a constraint on the potentials, as a clique tree is calibrated if neighboring potentials agree on the marginal distribution on their joint subset. Thus, we pose the following constrained optimization procedure:

CTree-Optimize

Find $\boldsymbol{Q}$

that maximize $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$

subject to

$$\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \pi_i = \mu_{i,j}, \quad \forall (\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}; \tag{2.6}$$

$$\sum_{\boldsymbol{C}_i} \pi_i = 1, \qquad \forall \boldsymbol{C}_i \in \mathcal{T}. \tag{2.7}$$

The constraints (2.6) and (2.7) ensure that the potentials in $\boldsymbol{Q}$ are calibrated and represent legal distributions. It can be shown that the objective function is strictly concave in the variables π, μ. The constraints define a convex set (linear subspace), so this optimization problem has a unique maximum. Since $\boldsymbol{Q}$ can represent $P_{\mathcal{F}}$, this maximum is attained when $\boldsymbol{D}(Q \| P_{\mathcal{F}}) = 0$.

2.3.2.2 Fixed-Point Characterization

We can now prove that the *stationary points* of this constrained optimization function — the points at which the gradient is orthogonal to all the constraints — can be characterized by a set of *self-consistent equations*.

Recall that a stationary point of a function is either a local maximum, a local minimum, or a saddle point. In this optimization problem, there is a single global maximum. Although we do not show it here, we can show that it is also the single stationary point. We can therefore define the global optimum declaratively, as a set of equations, using standard methods based on Lagrange multipliers. As we now show, this declarative formulation gives rise to a set of equations which precisely corresponds to message-passing steps in the clique tree, a standard inference procedure usually derived via dynamic programming.

Theorem 2.27
A set of potentials $\boldsymbol{Q}$ is a stationary point of CTree-Optimize if and only if there exists a set of factors $\{\delta_{i\to j}[\boldsymbol{S}_{i,j}] : \boldsymbol{C}_i — \boldsymbol{C}_j \in \mathcal{T}\}$ such that

$$\delta_{i\to j} \propto \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \pi_i^0 \left(\prod_{k \in \mathcal{N}_{\boldsymbol{C}_i} - \{j\}} \delta_{k\to i} \right) \tag{2.8}$$

$$\pi_i \propto \pi_i^0 \left(\prod_{j \in \mathcal{N}_{\boldsymbol{C}_i}} \delta_{j\to i} \right) \tag{2.9}$$

$$\mu_{i,j} = \delta_{j\to i} \times \delta_{i\to j}, \tag{2.10}$$

where $\mathcal{N}_{\boldsymbol{C}_i}$ are the neighboring cliques of $\boldsymbol{C}_i$ in $\mathcal{T}$.

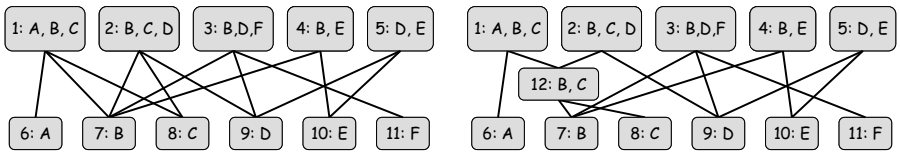
Theorem 2.27 illustrates themes that appear in many approaches that turn variational problems into message-passing schemes. It provides a characterization of the solution of the optimization problem in terms of *fixed-point equations* that must hold when we find a maximal Q. These fixed-point equations define the relationships that must hold between the different parameters involved in the optimization problem. Most importantly, (2.8) defines each $\delta_{i\to j}$ in terms of $\delta_{k\to j}$ *other than* $\delta_{i\to j}$. The other parameters are all defined in a noncyclic way in terms of the $\delta_{i\to j}$'s.

The form of the equations resulting from the theorem suggest an iterative procedure for finding a fixed point, in which we view the equations as assignments, and iteratively apply equations to the current values of the left-hand side to define a new value for the right-hand side. We initialize all of the $\delta_{i\to j}$'s to 1, and then iteratively apply (2.8), computing the left-hand side $\delta_{i\to j}$ of each equality in terms of the right-hand side (essentially converting each equality sign to an assignment). Clearly, a single iteration of this process does not usually suffice to make the equalities hold; however, under certain conditions (which hold in this particular case) we can guarantee that this process converges to a solution satisfying all of the equations in (2.8); the other equations are now easy to satisfy.

2.3.3 Loopy Belief Propagation in Pairwise Markov Networks

We focus on the class of pairwise Markov networks. In these networks, we have a univariate potential $\phi_i[X_i]$ over each variable X_i, and in addition a pairwise potential $\phi_{(i,j)}[X_i, X_j]$ over some pairs of variables. These pairwise potentials correspond to edges in the Markov network. Examples of such networks include our simple tuberculosis example in figure 2.3 and the grid networks we discussed above.

The transformation of such a network into a cluster graph is fairly straightforward. For each potential, we introduce a corresponding cluster, and put edges between the clusters that have overlapping scope. In other words, there is an edge



(a) $\mathcal{K}_3$ (b) $\mathcal{K}_4$

Figure 2.5 Two additional examples of generalized cluster graphs for a Markov network with potentials over $\{A, B, C\}$, $\{B, C, D\}$, $\{B, D, F\}$, $\{B, E\}$, and $\{D, E\}$. (a) Bethe factorization. (b) Capturing interactions between $\{A, B, C\}$ and $\{B, C, D\}$.

between the cluster $\boldsymbol{C}_{(i,j)}$ that corresponds to the edge X_i—X_j and the clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ that correspond to the univariate factors over X_i and X_j.

As there is a direct correspondence between the clusters in the cluster graphs and variables or edges in the original Markov network, it is often convenient to think of the propagation steps as operations on the original network. Moreover, as each pairwise cluster has only two neighbors, we consider two propagation steps along the path $\boldsymbol{C}_i$—$\boldsymbol{C}_{(i,j)}$—$\boldsymbol{C}_j$ as propagating information between X_i and X_j. Indeed, early versions of generalized belief propagation were stated in these terms. This algorithm is known as *loopy belief propagation*, as it uses propagation steps used by algorithms for Markov trees, except that it was applied to networks with loops.

A natural question is how to extend this method to networks that are more complex than pairwise Markov networks. Once we have larger potentials, they may overlap in ways that result in complex interactions among them.

One simple construction creates a bipartite graph. The first layer consists of "large" clusters, with one cluster for each factor ϕ in $\mathcal{F}$, whose scope is $Scope[\phi]$. These clusters ensure that we satisfy the family-preservation property. The second layer consists of "small" univariate clusters, one for each random variable. Finally, we place an edge between each univariate cluster X on the second layer and each cluster in the first layer that includes X; the scope of this edge is X itself. For a concrete example, see figure 2.5(a).

We can easily verify that this is a proper cluster graph. First, by construction it satisfies the family-preserving property. Second, the edges that mention a variable X form a star-shaped subgraph with edges from the univariate cluster with scope X to all the large clusters that contain X. We will call this construction the *Bethe approximation* (for reasons that will be clarified below). The construction of this cluster graph is simple and can easily be automated.

So far, our discussion of belief propagation has been entirely procedural, and motivated purely by similarity to message-passing algorithms for cluster trees. Is there any formal justification for this approach? Is there a sense in which we can view this algorithm as providing an approximation to the exact inference task? In this section, we show that belief propagation can be justified using the energy function formulation. Specifically, the messages passed by generalized belief propagation can be derived from fixed-point equations for the stationary points of an approximate

version of the energy functional of (2.3). As we shall see, this formulation provides significant insight into the generalized belief propagation algorithm. It allows us to better understand the convergence properties of generalized belief propagation, and to characterize its convergence points. It also suggests generalizations of the algorithm which have better convergence properties, or that optimize a more "accurate" approximation to the energy functional.

Our construction will be similar to the one in section 2.3.2 for exact inference. However, there are some differences. As we saw, the calibrated cluster graph maintains the information in $P_{\mathcal{F}}$. However, the resulting cluster potentials are not, in general, the marginals of $P_{\mathcal{F}}$. In fact, these cluster potentials may not represent the marginals of any single coherent joint distribution over $\mathcal{X}$. Thus, we can think of generalized belief propagation as constructing a set of *pseudo-marginal distributions*, each one over the variables in one cluster. These pseudo-marginals are calibrated, and therefore locally consistent with each other, but are not necessarily marginals of a single underlying joint distribution.

The energy functional $F[P_{\mathcal{F}}, Q]$ has terms involving the entropy of an entire joint distribution; thus, it cannot be used to evaluate the quality of an approximation defined in terms of (possibly incoherent) pseudo-marginals. However, the factored free energy functional $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$ is defined in terms of entropies of clusters and messages, and is therefore well-defined for pseudo-marginals $\boldsymbol{Q}$. Thus, we can write down an optimization problem as before:

CGraph-Optimize

Find $\boldsymbol{Q}$

that maximize $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$

subject to

$$\sum_{\boldsymbol{C}_i \setminus \boldsymbol{S}_{i,j}} \pi_i = \mu_{i,j}, \quad \forall (\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{T}; \tag{2.11}$$

$$\sum_{\boldsymbol{C}_i} \pi_i = 1, \qquad \forall \boldsymbol{C}_i \in \mathcal{T}. \tag{2.12}$$

Importantly, however, unlike for clique trees, $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$ is no longer simply a reformulation of the free energy, but rather an approximation of it. Thus, our optimization problem contains two approximations: we are using an approximation, rather than an exact, energy functional; and we are optimizing it over the space of pseudo-marginals, which is a relaxation (a superspace) of the space of all coherent probability distributions that factorize over the cluster graph. The approximate energy functional in this case is a restricted form of an approximation known as the *Kikuchi free energy* in statistical physics.

We noted that the energy functional is a lower bound of the log-partition function; thus, by maximizing it, we get better approximations of $P_{\mathcal{F}}$. Unfortunately, the factored energy functional, which is only an approximation to the true energy functional, is not necessarily also a lower bound. Nonetheless, it is still a reasonable strategy to maximize the approximate energy functional.

Our maximization problem is the natural analogue of CTree-Optimize to the case of cluster graphs. Not surprisingly, we can derive a similar analogue to theorem 2.27.

Theorem 2.28
A set of potentials $\boldsymbol{Q}$ is a stationary point of CGraph-Optimize if and only if for every edge $(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{K}$ there are auxiliary potentials $\delta_{i \to j}(\boldsymbol{S}_{i,j})$ and $\delta_{j \to i}(\boldsymbol{S}_{j,i})$ so that

$$\delta_{i \to j} \propto \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \pi_i^0 \times \prod_{k \in \mathcal{N}_{\boldsymbol{C}_i} - \{j\}} \delta_{k \to i} \tag{2.13}$$

$$\pi_i \propto \pi_i^0 \times \prod_{j \in \mathcal{N}_{\boldsymbol{C}_i}} \delta_{j \to i} \tag{2.14}$$

$$\mu_{i,j} = \delta_{j \to i} \times \delta_{i \to j}. \tag{2.15}$$

This theorem shows that we can characterize convergence points of the energy function in terms of the original potentials and messages between clusters. We can, once again, define a procedural variant, in which we initialize $\delta_{i \to j}$, and then iteratively use (2.13) to redefine each $\delta_{i \to j}$ in terms of the current values of other $\delta_{k \to i}$. theorem 2.28 shows that convergence points of this procedure are related to stationary points of $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$.

It is relatively easy to verify that $\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}]$ is bounded from above. And thus, this function must have a maximum. There are two cases. The maximum is either an interior point or a boundary point (some of the probabilities in $\boldsymbol{Q}$ are 0). In the former case the maximum is also a stationary point, which implies that it satisfies the condition of theorem 2.28. In the latter case, the maximum is not necessarily a stationary point. This situation, however, is very rare in practice, and can be guaranteed not to arise if we make some fairly benign assumptions.

It is important to understand what these results imply, and what they do not. The results imply only that the convergence points of generalized belief propagation are stationary points of the free energy function They do *not* imply that we can reach these convergence points by applying belief propagation steps. In fact, there is no guarantee that the message-passing steps of generalized belief propagation necessarily improve the free energy objective: a message passing step may increase or decrease the energy functional. (In fact, if generalized belief propagation was guaranteed to monotonically improve the functional, then it would necessarily always converge.)

What are the implications of this result? First, it provides us with a declarative semantics for generalized belief propagation in terms of optimization of a target functional. This declarative semantics opens the way to investigate other computational approaches for optimizing the same functional. We discuss some of these approaches below.

This result also allows us to understand what properties are important for this type of approximation, and subsequently to design other approximations that may be more accurate, or better in some other way. As a concrete example, recall that, in our discussion of generalized cluster graphs, we required the running intersection

property. This property has two important implications. First, that the set of clusters that contain some variable X are connected; hence, the marginal over X will be the same in all of these clusters at the calibration point. Second, that there is no cycle of clusters and sepsets all of which contain X. We can motivate this assumption intuitively, by noting that it prevents us from allowing information about X to cycle endlessly through a loop. The free energy function analysis provides a more formal justification. To understand it, consider first the form of the factored free energy functional when our cluster graph $\mathcal{K}$ has the form of the Bethe approximation Recall that in the Bethe approximation graph there are two layers: one consisting of clusters that correspond to factors in $\mathcal{F}$, and the other consisting of univariate clusters. When the cluster graph is calibrated, these univariate clusters have the same distribution as the separators between them and the factors in the first layer. As such, we can combine together the entropy terms for all the separators labeled by X and the associated univariate cluster and rewrite the free energy, as follows:

Proposition 2.29
If $\boldsymbol{Q} = \{\pi_\phi : \phi \in \mathcal{F}\} \cup \{\pi_i(X_i)\}$ is a calibrated set of potentials for $\mathcal{K}$ for a Bethe approximation cluster graph with clusters $\{\boldsymbol{C}_\phi \ : \ \phi \in \mathcal{F}\} \cup \{X_i \ : \ X_i \in \mathcal{X}\}$, then

$$\tilde{F}[P_{\mathcal{F}}, \boldsymbol{Q}] = \sum_{\phi \in \mathcal{F}} \boldsymbol{E}_{\pi_\phi}[\ln \phi] + \sum_{\phi \in \mathcal{F}} \boldsymbol{H}_{\pi_\phi}(\boldsymbol{C}_\phi) - \sum_i (d_i - 1)\boldsymbol{H}_{\pi_i}(X_i), \tag{2.16}$$

where $d_i = |\{\phi : X_i \in \mathit{Scope}[\phi]\}|$ is the number of factors that contain X_i.

Note that (2.16) is equivalent to the factored free energy only when $\boldsymbol{Q}$ is calibrated. However, as we are interested only in such cases, we can freely alternate between the two forms for the purpose of finding fixed points of the factored free energy functional. Equation (2.16) is known as the *Bethe free energy*, and again has a history in statistical mechanics. The Bethe approximation we discussed above is a construction in terms of cluster graphs that is designed to match the Bethe free energy.

As we can see in this alternative form, if the variable X_i appears in d_i clusters in the cluster graph, then it appears in an entropy term with a positive sign exactly d_i times. Due to the running intersection property, the number of separators that contain X_i is $d_i - 1$ (the number of edges in a tree with k vertices is $k - 1$), so that X_i appears in an entropy term with a negative sign exactly $d_i - 1$ times. In this case, we say that the *counting number* of X_i is 1. Thus, our approximation does not over- or undercount the entropy of X_i. It is not difficult to show that the counting number result holds for any approximation that satisfies the running intersection property. Thus, one motivation for the running intersection property is that cluster graphs satisfying it provide a better approximation to the free energy functional.

This intuition forms the basis for improved approximations. Specifically, we can construct energy functionals (called *Kikuchi free energy* approximations) that resemble (2.5), in which we introduce additional entropy terms, with both positive and negative signs, in a way that ensures that the counting number for all variables

is 1. Somewhat remarkably, the same analysis we performed in this section — defining a set of fixed-point equations for stationary points of the approximate free energy — also leads to message-passing algorithms for these richer approximations. The propagation rules for these approximations, which also fall under the heading of generalized belief propagation, are more elaborate, and we do not discuss them here.

2.3.4 Sampling-Based Approximate Inference

As we discussed above, another approach to dealing with the worst-case combinatorial explosion of exact inference in graphical models is via *sampling-based methods*. In these methods, we approximate the joint distribution as a set of instantiations to all or some of the variables in the network. These instantiations, often called *samples*, represent part of the probability mass.

The general framework for most of the discussion is as follows. Consider some distribution $P(\mathcal{X})$, and assume we want to estimate the probability of some event $\boldsymbol{Y} = \boldsymbol{y}$ relative to P, for some $\boldsymbol{Y} \subseteq \mathcal{X}$ and $\boldsymbol{y} \in Val(\boldsymbol{Y})$. More generally, we might want to estimate the expectation of some function $f(\mathcal{X})$ relative to P; this task is a generalization, as we can choose $f(\xi) = \mathbf{1}\{\xi\langle\boldsymbol{Y}\rangle = \boldsymbol{y}\}$. We approximate this expectation by generating a set of M samples, estimating the value of the function or its expectation relative to each of the generated samples, and then aggregating the results.

2.3.4.1 Markov Chain Monte Carlo Methods

Markov chain Monte Carlo (abbreviated *MCMC*) is an approach for generating samples from the posterior distribution. As we discussed, we cannot typically sample from the posterior directly; however, we can construct a process which gradually samples from distributions that are closer and closer to the posterior. Intuitively, we define a state graph whose nodes are the states of the system, i.e., possible instantiations $Val(\mathcal{X})$. (This graph is very different from the graphical model that defines the distribution $P(\mathcal{X})$, whose nodes correspond to variables.) We then define a process that randomly traverses this graph, moving from one state to another. This process is defined so that, ultimately (after enough steps), the probability of being in any particular state is the desired posterior distribution.

We begin by describing the general framework of Markov chains, and then describe their application to approximate inference in graphical models. We note that, unlike forward sampling methods (including likelihood weighting), Markov chain methods apply equally well to directed and to undirected models.

A *Markov chain* is defined in terms of a set of states, and a transition model from one state to another. The chain defines a process that evolves stochastically from state to state.

Definition 2.30
A *Markov chain* is defined via a state space $Val(\boldsymbol{X})$ and a *transition probability model*, which defines, for every state $\boldsymbol{x} \in Val(X)$ a *next-state* distribution over $Val(X)$. The *transition probability* of going from $\boldsymbol{x}$ to $\boldsymbol{x}'$ is denoted $\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}')$. This transition probability applies whenever the chain is in state $\boldsymbol{x}$. ∎

We note that, in this definition and in the subsequent discussion, we restrict attention to *homogeneous* Markov chains, where the system dynamics do not change over time.

We can imagine a random sampling process that defines a sequence of states $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$. As the transition model is random, the state of the process at step t can be viewed as a random variable $\boldsymbol{X}^{(t)}$. We assume that the initial state $\boldsymbol{X}^{(0)}$ is distributed according to some initial state distribution $P^{(0)}(\boldsymbol{X}^{(0)})$. We can now define distributions over the subsequent states $P^{(1)}(\boldsymbol{X}^{(1)}), P^{(2)}(\boldsymbol{X}^{(2)}), \ldots$ using the chain dynamics:

$$P^{(t+1)}(\boldsymbol{X}^{(t+1)} = \boldsymbol{x}') = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} P^{(t)}(\boldsymbol{X}^{(t)} = \boldsymbol{x})\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}'). \tag{2.17}$$

Intuitively, the probability of being at state $\boldsymbol{x}'$ at time $t+1$ is the sum over all possible states $\boldsymbol{x}$ that the chain could have been in at time t of the probability being in state $\boldsymbol{x}$ times the probability that the chain took a transition from $\boldsymbol{x}$ to $\boldsymbol{x}'$.

As the process converges, we would expect $P^{(t+1)}$ to be close to $P^{(t)}$. Using (2.17), we obtain

$$P^{(t)}(\boldsymbol{x}') \approx P^{(t+1)}(\boldsymbol{x}') = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} P^{(t)}(\boldsymbol{x})\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}').$$

At convergence, we would expect the resulting distribution $\pi(\boldsymbol{X})$ to be an equilibrium relative to the transition model; i.e., the probability of being in a state is the same as the probability of transitioning into it from a randomly sampled predecessor. Formally:

Definition 2.31
A distribution $\pi(\boldsymbol{X})$ is a *stationary distribution* for a Markov chain $\mathcal{T}$ if it satisfies

$$\pi(\boldsymbol{X} = \boldsymbol{x}') = \sum_{\boldsymbol{x} \in Val(\boldsymbol{X})} \pi(\boldsymbol{X} = \boldsymbol{x})\mathcal{T}(\boldsymbol{x} \to \boldsymbol{x}'). \quad ∎ \tag{2.18}$$

We wish to restrict attention to Markov chains that have a unique stationary distribution, which is reached from any starting distribution $P^{(0)}$. There are various conditions that suffice to guarantee this property. The condition most commonly used is a fairly technical one, that the chain be *ergodic*. In the context of Markov chains where the state space $Val(\boldsymbol{X})$ is finite, the following condition is equivalent to this requirement:

Definition 2.32
A Markov chain is said to be *regular* if there exists some number k such that, for every $\boldsymbol{x}, \boldsymbol{x}' \in Val(\boldsymbol{X})$, the probability of getting from $\boldsymbol{x}$ to $\boldsymbol{x}'$ in exactly k steps is greater than 0. ∎

The following result can be shown to hold:

Theorem 2.33
A finite-state Markov chain $\mathcal{T}$ has a unique stationary distribution if and only if it is regular.

Ensuring regularity is usually straightforward. Two simple conditions that guarantee regularity in finite-state Markov chains are:

- It is possible to get from any state to any state using a positive probability path in the state graph.
- For each state $\boldsymbol{x}$, there is a positive probability of transitioning from $\boldsymbol{x}$ to $\boldsymbol{x}$ in one step (a self-loop).

These two conditions together are sufficient but not necessary to guarantee regularity. However, they often hold in the chains used in practice.

2.3.4.2 Markov Chains for Graphical Models

The theory of Markov chains provides a general framework for generating samples from a target distribution π. In this section, we discuss the application of this framework to the sampling tasks encountered in probabilistic graphical models. In this case, we typically wish to generate samples from the posterior distribution $P(\mathcal{X} \mid \boldsymbol{E} = \boldsymbol{e})$. Thus, we wish to define a chain for which $P(\mathcal{X} \mid \boldsymbol{e})$ is the stationary distribution. Clearly, there are many ways of defining such a chain. We focus on the most common approaches.

In graphical models, we define the states of the Markov chain to be instantiations ξ to $\mathcal{X}$, which are compatible with $\boldsymbol{e}$; i.e., all of the states ξ in the Markov chain satisfy $\xi\langle\boldsymbol{E}\rangle = \boldsymbol{e}$. The states in our Markov chain are therefore some subset of the possible assignments to the variables $\mathcal{X}$. In order to define a Markov chain, we need to define a process that transitions from one state to the other, converging to a stationary distribution $\pi(\xi)$ which is the desired posterior distribution $P(\xi \mid \boldsymbol{e})$.

In the case of graphical models, our state space has a factorized structure — each state is an assignment to several variables. When defining a transition model over this state space, we can consider a fully general case, where a transition can go from any state to any state. However, it is often convenient to decompose the transition model, considering transitions that only update a single component of the state vector at a time, i.e., only a value for a single variable. In this case, as in several other settings, we often define a set of transition models $\mathcal{T}_1, \ldots, \mathcal{T}_k$, each with its own dynamics. In certain cases, the different transition models are necessary, because no single transition model on its own suffices to ensure regularity.

In other cases, having multiple transition models simply makes the state space more "connected," and therefore speeds the convergence to a stationary distribution.

There are several ways of combining these multiple transition models into a single chain. One common approach is simply to randomly select between them at each step, using any distribution. Thus, for example, at each step, we might select one of $\mathcal{T}_1, \ldots, \mathcal{T}_k$, each with probability $1/k$. Alternatively, we can simply cycle over the different transition models, taking each one in turn. Clearly, this approach does not define a homogeneous chain, as the transition model used in step i is different from the one used in step $i+1$. However, we can simply view the process as defining a single transition model $\mathcal{T}$ each of whose steps is an aggregate step, consisting of first taking $\mathcal{T}_1$, then $\mathcal{T}_2$, ..., through $\mathcal{T}_k$.

In the case of graphical models, we define $\boldsymbol{X} = \mathcal{X} - \boldsymbol{E} = \{X_1, \ldots, X_k\}$. We define a multiple transition chain, where we have a *local transition model* $\mathcal{T}_i$ for each variable $X_i \in \boldsymbol{X}$. Let $\boldsymbol{U}_i = \mathcal{X} - \{X_i\}$, and let $\boldsymbol{u}_i$ denote an instantiation to $\boldsymbol{U}_i$. The model $\mathcal{T}_i$ takes a state $(\boldsymbol{u}_i, x_i)$ and transitions to a state of the form $(\boldsymbol{u}_i, x_i')$. As we discussed above, we can combine the different local transition models into a single global model in various ways.

2.3.4.3 Gibbs Sampling

Gibbs sampling is one simple yet effective Markov chain for factored state spaces, which is particularly efficient for graphical models. We define the local transition model $\mathcal{T}_i$ as follows. Intuitively, we simply "forget" the value of X_i in the current state, and sample a new value for X_i from its posterior given the rest of the current state. More precisely, let $(\boldsymbol{u}_i, x_i)$ be a state in the chain. We define

$$\mathcal{T}((\boldsymbol{u}_i, x_i) \rightarrow (\boldsymbol{u}_i, x_i')) = P(x_i' \mid \boldsymbol{u}_i). \tag{2.19}$$

Note that the transition probability does not depend on the current value x_i of X_i, but only on the remaining state $\boldsymbol{u}_i$.

The Gibbs chain is defined via a set of local transition models; we use the multistep transition model to combine them. Note that the different local transitions are taken consecutively; i.e., having changed the value for a variable X_1, the value for X_2 is sampled based on the new value. Also note that we are only collecting a single sample for every sequence where each local transition has been taken once.

This chain is guaranteed to be regular whenever the distribution is positive, so that every value of X_i has positive probability given an assignment $\boldsymbol{u}_i$ to the remaining variables. In this case, we can get from any state to any state in at most k local transition steps, where $k = |\mathcal{X} - \boldsymbol{E}|$. Positivity is, however, not necessary; there are many examples of nonpositive distributions where the Gibbs chain is regular. It is also easy to show that the posterior distribution $P(\mathcal{X} \mid \boldsymbol{e})$ is a stationary distribution of this process.

Gibbs sampling is particularly well suited to many graphical models, where we can compute the transition probability $P(X_i \mid \boldsymbol{u}_i)$ very efficiently. In particular, as

we now show, this distribution can be done based only on the Markov blanket of X_i. We show this analysis for a Markov network; the extension to Bayesian networks is straightforward. In general, we can decompose the probability of an instantiation as follows:

$$P(x_1 \mid x_2, \ldots, x_n) = \frac{1}{Z} \prod_j \pi_j[\boldsymbol{C}_j] = \frac{1}{Z} \prod_{j \,:\, X_i \in \boldsymbol{C}_j} \pi_j[\boldsymbol{C}_j] \prod_{j \,:\, X_i \notin \boldsymbol{C}_j} \pi_j[\boldsymbol{C}_j].$$

For shorthand, let $\pi_j[x_i, \boldsymbol{u}]$ denote $\pi_j[x_i, \boldsymbol{u}\langle \boldsymbol{C}_j \rangle]$. We can now compute

$$P(x_i' \mid \boldsymbol{u}_i) = \frac{P(x_i', \boldsymbol{u}_i)}{\sum_{x_i''} P(x_i'', \boldsymbol{u}_i)} = \frac{\prod_{\boldsymbol{C}_j \ni X_i} \pi_j[x_i', \boldsymbol{u}_i]}{\sum_{x_i''} \prod_{\boldsymbol{C}_j \ni X_i} \pi_j[(x_i'', \boldsymbol{u}_i)]}. \tag{2.20}$$

This last expression uses only the clique potentials involving X_i, and depends only on the instantiation in $\boldsymbol{u}_i$ of X_i's Markov blanket. In the case of Bayesian networks, this expression reduces to a formula involving only the CPDs of X_i and its children, and its value, again, depends only on the assignment in $\boldsymbol{u}_i$ to the Markov blanket of X_i. It can thus be computed very efficiently.

We note that the Markov chain defined by a graphical model is not necessarily regular, and might not converge to a unique stationary distribution. It turns out that this type of situation can only arise if the distribution defined by the graphical model is nonpositive, i.e., if the CPDs or clique potentials have entries with the value 0.

Theorem 2.34
Let $\mathcal{H}$ be a Markov network such that all of the clique potentials are strictly positive. Then the Gibbs-sampling Markov chain is regular.

2.3.4.4 Building a Markov Chain

As we discussed, the use of MCMC methods relies on the construction of a Markov chain that has the desired properties: regularity, and the target stationary distribution. Above, we described the Gibbs chain, a simple Markov chain that is guaranteed to have these properties under certain assumptions. However, Gibbs sampling is only applicable in certain circumstances; in particular, we must be able to sample from the distribution $P(X_i \mid \boldsymbol{u}_i)$. Although this sampling step is easy for discrete graphical models, there are other types of models where this step is not practical, and the Gibbs chain is not applicable. Unfortunately, it is beyond the scope of this chapter to discuss the Metropolis-Hastings algorithm, a more general method of constructing a Markov chain that is guaranteed to converge to the desired stationary distribution.

2.3.4.5 Generating Samples

The burn-in time for a large Markov chain is often quite large. Thus, the naive algorithm described above has to execute a large number of sampling steps for

every usable sample. However, a key observation is that, if $\boldsymbol{x}^{(t)}$ is sampled from π, then $\boldsymbol{x}^{(t+1)}$ is also sampled from π. Thus, once we have run the chain long enough that we are sampling from the stationary distribution (or a distribution close to it), we can continue generating samples from the same trajectory, and obtain a large number of samples from the stationary distribution.

More formally, assume that we use $\boldsymbol{x}^{(0)}, \ldots, \boldsymbol{x}^{(T)}$ as our burn-in phase, and then collect M samples $\boldsymbol{x}^{(T+1)}, \ldots, \boldsymbol{x}^{(T+M)}$. Thus, we have collected a data set $\mathcal{D}$ where $\boldsymbol{x}^m = \boldsymbol{x}^{(T+m)}$, for $m = 1, \ldots, M$. Assume, for simplicity, that $\boldsymbol{x}^{(T+1)}$ is sampled from π, and hence so are all of the samples in $\mathcal{D}$. It follows that for any function f: $\sum_{m=1}^{M} f(\boldsymbol{x}^m)$ is an unbiased estimator for $\boldsymbol{E}_{\pi(\boldsymbol{X})}[f(\boldsymbol{X})]$.

The key problem, of course, is that consecutive samples from the same trajectory are correlated. Thus, we cannot expect the same performance as we would from M independent samples from π. In other words, the variance of the estimator is significantly higher than that of an estimator generated by M independent samples from π, as discussed above.

One solution to this problem is not to collect consecutive samples from the chain. Rather, having collected a sample $\boldsymbol{x}^{(T)}$, we let the chain run for a while, and collect a second sample $\boldsymbol{x}^{(T+d)}$ for some appropriate choice of d. For d large enough, $\boldsymbol{x}^{(T)}$ and $\boldsymbol{x}^{(T+d)}$ are only slightly correlated, and we can view them as independent samples from π. However, the time d required for "forgetting" the correlation is clearly related to the mixing time of the chain. Thus, chains that are slow to mix initially also require larger d in order to produce close-to-independent samples. Nevertheless, the samples do come from the correct distribution for any value of d, and hence it is often better to compromise and use a shorter d than it is to use a shorter burn-in time T. This method thus allows us to collect a larger number of usable samples with fewer transitions of the Markov chain.

In fact, we can often make even better use of the samples generated using this single-chain approach. Although the samples between $\boldsymbol{x}^{(T)}$ and $\boldsymbol{x}^{(T+d)}$ are not independent samples, there is no reason to discard them. That is, using all of the samples $\boldsymbol{x}^{(T)}, \boldsymbol{x}^{(T+1)}, \ldots, \boldsymbol{x}^{(T+d)}$ produces a provably better estimator than using just the two samples $\boldsymbol{x}^{(T)}$ and $\boldsymbol{x}^{(T+d)}$: our variance is always no higher if we use all of the samples we generated rather than a subset. Thus, the strategy of picking only a subset of the samples is useful primarily in settings where there is a significant cost associated with using each sample (e.g., the evaluation of f is costly), so that we might want to reduce the overall number of samples used.

2.3.4.6 Discussion

This description of the use of Markov chains is quite abstract: It contains no specification of the number of chains to run, the metrics for evaluating mixing, techniques for determining the delay between samples that would allow them to be considered independent, and more. Unfortunately, at this point, there is little theoretical analysis that can help answer these questions for the chains that are of interest to us. Thus, the application of Markov chains is more of an art

than a science, and often requires significant experimentation and hand-tuning of parameters.

Nevertheless, MCMC methods are, for many probabilistic models, the only technique that can achieve reasonable performance. Specifically, unlike forward sampling methods, it does not degrade when the probability of the evidence is low, or when the posterior is very different from the prior. Furthermore, unlike forward sampling, it applies to undirected models as well as to directed models. As such, it is an important component in the suite of approximate inference techniques.

2.4 Learning

Next, we turn our attention to learning graphical models [4, 6]. There are two variants of the learning task: parameter estimation and structure learning. In the parameter estimation task, we assume that the qualitative dependency structure of the graphical model is known; i.e., in the directed model case, $\mathcal{G}$ is given, and in the undirected case, $\mathcal{H}$ is given. In this case, the learning task is simply to fill in the parameters that define the CPDs of the attributes or the parameters which define the potential functions of the Markov network. In the structure learning task, there is no additional required input (although the user can, if available, provide prior knowledge about the structure, e.g., in the form of constraints). The goal is to extract a Bayesian network or Markov network, structure as well as parameters, from the training data alone. We discuss each of these problems in turn.

2.4.1 Parameter Estimation in Bayesian Networks

We begin with learning the parameters for a Bayesian network where the dependency structure is known. In other words, we are given the structure $\mathcal{G}$ that determines the set of parents for each random variable, and our task is to learn the parameters $\boldsymbol{\theta}_{\mathcal{G}}$ that define the CPDs for this structure. Our learning is based on a particular training set $\mathcal{D} = \{x^1, \ldots, x^m\}$, which, for now, we will assume is *complete* (i.e., each instance is fully observed, there are no missing values). While this task is relatively straightforward, it is of interest in and of itself. In addition, it is a crucial component in the structure learning algorithm described in section 2.4.3.

There are two approaches to parameter estimation: *maximum likelihood estimation (MLE)* and Bayesian approaches. The key ingredient for both is the likelihood function: the probability of the data given the model. This function captures the response of the probability distribution to changes in the choice of parameters. The likelihood of a parameter set is defined to be the probability of the data given the model. For a Bayesian network structure $\mathcal{G}$ the likelihood of a parameter set $\boldsymbol{\theta}_{\mathcal{G}}$ is

$$L(\boldsymbol{\theta}_{\mathcal{G}} : \mathcal{D}) = P(\mathcal{D} \mid \boldsymbol{\theta}_{\mathcal{G}}).$$

2.4.1.1 Maximum Likelihood Parameter Estimation

Given the above, one approach to parameter estimation is *maximum likelihood* parameter estimation. Here, our goal is to find the parameter setting $\boldsymbol{\theta}_{\mathcal{G}}$ that maximizes the likelihood $L(\boldsymbol{\theta}_{\mathcal{G}} : \mathcal{D})$. For Bayesian networks, the likelihood can be decomposed as follows:

$$\begin{aligned} L(\boldsymbol{\theta}_{\mathcal{G}}, \mathcal{D}) &= \prod_{j=1}^{m} P(x^j : \boldsymbol{\theta}_{\mathcal{G}}) \\ &= \prod_{j=1}^{m} \prod_{i=1}^{n} P(x_i^j \mid \mathbf{Pa}_{x_i^j} : \boldsymbol{\theta}_{\mathcal{G}}) \\ &= \prod_{i=1}^{n} \prod_{j=1}^{m} P(x_i^j \mid \mathbf{Pa}_{x_i^j} : \boldsymbol{\theta}_{\mathcal{G}}) \end{aligned}$$

We will use $\boldsymbol{\theta}_{X_i|\mathbf{Pa}_i}$ to denote the subset of parameters that determine $P(X_i \mid \mathbf{Pa}_i)$. In the case where the parameters are disjoint (each CPD is parameterized by a separate set of parameters that do not overlap; this allows us to maximize each parameter set independently. We can write the likelihood as follows:

$$L(\boldsymbol{\theta}_{\mathcal{G}} : \mathcal{D}) = \prod_{i=1}^{n} L_i(\boldsymbol{\theta}_{X_i|\mathbf{Pa}_i} : \mathcal{D}),$$

where the *local likelihood* function for X_i is

$$L_i(\boldsymbol{\theta}_{X_i|\mathbf{Pa}_i} : \mathcal{D}) = \prod_{j=1}^{m} P(x_i^j \mid \mathbf{pa}_i^j : \boldsymbol{\theta}_{X_i|\mathbf{Pa}_i}).$$

The simplest parameterization for the CPDs is as a table. Suppose we have a variable X with parents $\boldsymbol{U}$. If we represent that CPD $P(X \mid \boldsymbol{U})$ as a table, then we will have a parameter $\theta_{x|\boldsymbol{u}}$ for each combination of $x \in Val(X)$ and $\boldsymbol{u} \in Val(\boldsymbol{U})$. In this case, we can write the local likelihood function as follows:

$$\begin{aligned} L_X(\boldsymbol{\theta}_{X|\boldsymbol{U}} : \mathcal{D}) &= \prod_{j=1}^{m} \theta_{x^j|\boldsymbol{u}^j} \\ &= \prod_{\boldsymbol{u} \in Val(\boldsymbol{U})} \left[\prod_{x \in Val(X)} \theta_{x|\boldsymbol{u}}^{N_{\boldsymbol{u},x}} \right], \end{aligned} \tag{2.21}$$

where $N_{\boldsymbol{u},x}$ is the number of times $X = x$ and $\mathbf{Pa}_i = \boldsymbol{u}$ in $\mathcal{D}$. That is, we have grouped together all the occurrences of $\theta_{x|\boldsymbol{u}}$ in the product over all instances.

We need to maximize this term under the constraints that, for each choice of value for the parents $\boldsymbol{U}$, the conditional probability is legal:

$$\sum \theta_{x|\boldsymbol{u}} = 1 \quad \text{for all } \boldsymbol{u}.$$

These constraints imply that the choice of value for $\theta_{x|\boldsymbol{u}}$ can impact the choice of value for $\theta_{x'|\boldsymbol{u}}$. However, the choice of parameters given different values $\boldsymbol{u}$ of $\boldsymbol{U}$ are independent of each other. Thus, we can maximize each of the terms in square brackets in (2.21) independently.

We can thus further decompose the local likelihood function for a tabular CPD into a product of simple likelihood functions. It is easy to see that each of these likelihood functions is a *multinomial* likelihood. The counts in the data for the different outcomes x are simply $\{N_{\boldsymbol{u},x} : x \in Val(X)\}$. We can then immediately use the MLE parameters for a multinomial which are simply

$$\hat{\theta}_{x|\boldsymbol{u}} = \frac{N_{\boldsymbol{u},x}}{N_{\boldsymbol{u}}},$$

where we use the fact that $N_{\boldsymbol{u}} = \sum_x N_{\boldsymbol{u},x}$.

2.4.1.2 Bayesian Parameter Estimation

In many cases, maximum likelihood parameter estimation is not robust, as it overfits the training data. The Bayesian approach uses a prior distribution over the parameters to smooth the irregularities in the training data, and is therefore significantly more robust. As we will see in section 2.4.3, the Bayesian framework also gives us a good metric for evaluating the quality of different candidate structures.

Roughly speaking, the Bayesian approach introduces a prior over the unknown parameters, allowing us to specify a joint distribution over the unknown parameters and the data instances, and performs Bayesian conditioning, using the data as evidence, to compute a posterior distribution over these parameters.

Consider the following simple example: we want to estimate parameters for a simple network with two variables X and Y, where X is the parent of Y. Our training data consists of observations x^j, y^j for $j = 1, \ldots, m$. In addition, assume that our CPDs are represented as multinomials and we have unknown parameter vectors $\boldsymbol{\theta}_X$, $\boldsymbol{\theta}_{Y|x^0}$, and $\boldsymbol{\theta}_{Y|x^1}$.

The dependencies between these variables are described in the network of figure 2.6. This is the *meta-Bayesian network* that describes our learning setup. This Bayesian network structure immediately reveals several points. For example, the instances are independent given the unknown parameters. In addition, a common assumption made is that the individual parameter variables are a priori independent. That is, we believe that knowing the value of one parameter tells us nothing about another. This is called *parameter independence*. The suitability of this assumption depends on the domain, and it should be considered with care.

If we accept parameter independence, we can draw an important conclusion. Complete data d-separates the parameters for different CPDs. Given the data set $\mathcal{D}$, we can determine the posterior over $\boldsymbol{\theta}_X$ independently of the posterior over $\boldsymbol{\theta}_{Y|X}$. Once we solve each problem separately, we can combine the results. This is the analogous result to the likelihood decomposition for MLE estimation of section 2.4.1.1.

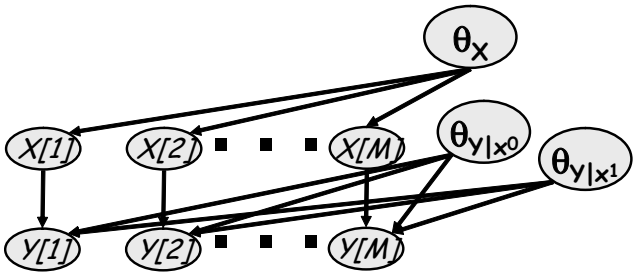


Figure 2.6 The Bayesian network for parameter estimation for a simple two-node Bayesian network.

Consider, for example, the learning setting described in figure 2.6, where we take both X and Y to be binary. We need to represent the posterior $\boldsymbol{\theta}_X$ and $\boldsymbol{\theta}_{Y|X}$ given the data. If we use a Dirichlet prior over $\boldsymbol{\theta}_X$, $\theta_{Y|x^0}$, and $\theta_{Y|x^1}$, then the posterior $P(\boldsymbol{\theta}_X \mid x^1, \ldots, x^M)$ can also be represented as a Dirichlet distribution.

Suppose that $P(\boldsymbol{\theta}_X)$ is a Dirichlet prior with hyperparameters α_{x^0} and α_{x^1}, $P(\boldsymbol{\theta}_{Y|x^0})$ is a Dirichlet prior with hyperparameters $\alpha_{y^0|x^0}$ and $\alpha_{y^1|x^0}$, and $P(\boldsymbol{\theta}_{Y|x^1})$ is a Dirichlet prior with hyperparameters $\alpha_{y^0|x^1}$ and $\alpha_{y^1|x^1}$.

As in decomposition for the likelihood function in section 2.4.1.1, the likelihood terms that involve $\boldsymbol{\theta}_{Y|x^0}$ depend on all the data elements X^j such that $x^j = x^0$ and the terms that involve $\boldsymbol{\theta}_{Y|x^1}$ depend on all the data elements X^j such that $x^j = x^1$ We can decompose the joint distribution over parameters and data as follows:

$$\begin{aligned} P(\boldsymbol{\theta}_{\mathcal{G}}, \mathcal{D}) = P(\boldsymbol{\theta}_X) L_X(\boldsymbol{\theta}_X : \mathcal{D}) \\ P(\boldsymbol{\theta}_{Y|x^1}) \prod_{j: x^j = x^1} P(y^j \mid x^n : \boldsymbol{\theta}_{Y|x^1}) \\ P(\boldsymbol{\theta}_{Y|x^0}) \prod_{j: x^j = x^0} P(y^j \mid x^j : \boldsymbol{\theta}_{Y|x^0}) \end{aligned}$$

Thus, this joint distribution is a product of three separate joint distributions with a Dirichlet prior for some multinomial parameter and data drawn from this multinomial. We can conclude that the posterior for $P(\boldsymbol{\theta}_X \mid \mathcal{D})$ is Dirichlet with hyperparameters $\alpha_{x^0} + N_{x^0}$ and $\alpha_{x^1} + N_{x^1}$; the posterior for $P(\boldsymbol{\theta}_{Y|x^0} \mid \mathcal{D})$ is Dirichlet with hyperparameters $\alpha_{y^0|x^0} + N_{x^0,y^0}$ and $\alpha_{y^1|x^0} + N_{x^0,y^1}$; and the posterior for $P(\boldsymbol{\theta}_{Y|x^1} \mid \mathcal{D})$ is Dirichlet with hyperparameters $\alpha_{y^0|x^1} + N_{x^1,y^0}$ and $\alpha_{y^1|x^1} + N_{x^1,y^1}$.

The same pattern of reasoning we discussed applied to the general case. Let $\mathcal{D}$ be a complete data set for $\mathcal{X}$, and let $\mathcal{G}$ be a network structure over these variables with table CPDs. If the prior $P(\boldsymbol{\theta}_{\mathcal{G}})$ satisfies parameter independence, then

$$P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{D}) = \prod_i \prod_{\mathbf{pa}_i} P(\boldsymbol{\theta}_{X_i|\mathbf{pa}_i} \mid \mathcal{D}).$$

If $P(\boldsymbol{\theta}_{X|\boldsymbol{u}})$ is a Dirichlet prior with hyperparameters $\alpha_{x^1|\boldsymbol{u}}, \ldots, \alpha_{x^K|\boldsymbol{u}}$, then the posterior $P(\boldsymbol{\theta}_{X|\boldsymbol{u}} \mid \mathcal{D})$ is a Dirichlet distribution with hyperparameters $\alpha_{x^1|\boldsymbol{u}} + N_{\boldsymbol{u},x^1}, \ldots, \alpha_{x^K|\boldsymbol{u}} + N_{\boldsymbol{u},x^K}$.

This induces a predictive model in which, for the next instance, we have that

$$P(X_i[m+1] = x_i \mid \boldsymbol{U}[m+1] = \boldsymbol{u}, \mathcal{D}) = \frac{\alpha_{x_i|\boldsymbol{u}} + N_{x_i,\boldsymbol{u}}}{\sum_i \alpha_{x_i|\boldsymbol{u}} + N_{x_i,\boldsymbol{u}}}. \tag{2.22}$$

Putting this all together, we can see that for computing the probability of a new instance, we can use a single network parameterized as usual, via a set of multinomials, but ones computed as in (2.22).

2.4.2 Parameter Estimation in Markov Networks

Unfortunately, for general Markov networks, the likelihood function cannot be decomposed. A notable exception is chordal Markov networks, but we will focus on the general case here. For a network with a set of cliques $\boldsymbol{D}_1, \ldots, \boldsymbol{D}_n$, the likelihood function is given by

$$L(\epsilon, \mathcal{D}) = \prod_{j=1}^{m} \frac{1}{Z} \exp\left[-\sum_{i=1}^{n} \epsilon_i[\boldsymbol{x}_i^j]\right],$$

where $\boldsymbol{x}_i^j$ is the value of the variables $\boldsymbol{D}_i$ in the instance x^j and $Z = \sum_{\boldsymbol{x}} \exp\left[-\sum_{i=1}^{n} \epsilon_i[\boldsymbol{x}_i]\right]$ is the normalization constant. This normalization constant is responsible for coupling the estimation parameters and effectively ruling out a closed-form solution. Luckily, this objective function is concave in ϵ, so we have an unconstrained concave maximization problem, which can be solved by simple gradient ascent or second-order methods.

More concretely, for each $\boldsymbol{u}_i \in \mathit{Val}(\boldsymbol{D}_i)$ we have a parameter $\epsilon_{i,\boldsymbol{u}_i} \in \mathbb{R}$. This is the simplest case of complete parameterization. Often, however, parameters may be tied or clamped to zero. This does not change the fundamental complexity or method of estimation. The derivative of the log-likelihood with respect to $\epsilon_{i,\boldsymbol{u}_i}$ is given by

$$\frac{\partial \log L(\epsilon, \mathcal{D})}{\partial \epsilon_{i,\boldsymbol{u}_i}} = \sum_{j=1}^{m} \left[P(\boldsymbol{u}_i \mid \epsilon) - \boldsymbol{1}\{\boldsymbol{x}_i^j = \boldsymbol{u}_i\}\right] = mP(\boldsymbol{u}_i \mid \epsilon) - N_{\boldsymbol{u}_i}.$$

Note that the gradient is zero when the counts of the data correspond exactly with the expected counts predicted by the model. In practice, a prior on the parameters is used to help avoid overfitting. The standard prior is a diagonal Gaussian, $\epsilon \sim N(0, \sigma^2 I)$, which adds an additional factor of $\frac{\epsilon_{i,\boldsymbol{u}_i}}{\sigma^2}$ to the gradient.

To compute the probability $P(\boldsymbol{u}_i \mid \epsilon)$ needed to evaluate the gradient, we need to perform inference in the Markov network. Unlike in Bayesian networks, where parameters of intractable (large treewidth) graphs can be estimated by simple counting because of local normalization, the undirected case requires inference even during the learning stage. This is one of the prices of the flexibility of global normalization in Markov networks. See further discussion in chapter 4. Because of this added complexity, maximum-likelihood learning of the Markov network

structure is much more expensive and much less investigated; we will focus below on Bayesian networks.

2.4.3 Learning the Bayesian Network Structure

Next we consider the problem of learning the *structure* of a Bayesian network. There are three broad classes of algorithms for BN structure learning:

Constraint-based approaches These approaches view a Bayesian network as a representation of independencies. They try to test for conditional dependence and independence in the data, and then find a network that best explains these dependencies and independencies. Constraint-based methods are quite intuitive; they closely follow the definition of Bayesian network. A potential disadvantage of these methods is they can be sensitive to failures in individual independence tests.

Score-based approaches These approaches view a Bayesian network as specifying a statistical model, and then address learning as a *model selection* problem. These all operate on the same principle: We define a hypothesis space of potential models — the set of possible network structures we are willing to consider — and a scoring function that measures how well the model fits the observed data. Our computational task is then to find the highest-scoring network structure. The space of Bayesian networks is a combinatorial space, consisting of a superexponential number of structures — $2^{O(n^2)}$. Therefore, even with a scoring function, it is not clear how one can find the highest-scoring network. There are very special cases where we can find the optimal network. In general, however, the problem is NP-hard, and we resort to heuristic search techniques. Score-based methods consider the whole structure at once, and are therefore less sensitive to individual failures and are better at making compromises between the extent to which variables are dependent in the data and the "cost" of adding the edge. The disadvantage of the score-based approaches is that they are in general not gauranteed to find the optimal solution.

Bayesian model averaging approaches The third class of approaches do not attempt to learn a single structure. They are based on a Bayesian framework describing a distribution over possible structures and try to average the prediction of all possible structures. Since the number of structures is immense, performing this task seems impossible. For some classes of models this can be done efficiently, and for others we need to resort to approximations.

In this chapter, we focus on the second approach, score-based approaches to structure selection. For details about the other approaches, see [8].

2.4.3.1 Structure Scores

As discussed above, score-based methods approach the problem of structure learning as an optimization problem. We define a score function that can score each candidate structure with respect to the training data, and then search for a high-scoring structure. As can be expected, one of the most important decisions we must make in this framework is the choice of scoring function. In this subsection, we discuss two of the most obvious choices.

The Likelihood Score A natural choice for scoring function is the likelihood function, which we used for parameter estimation. This measures the probability of the data given a model; thus, it seems intuitive to find a model that would make the data as probable as possible.

Assume that we want to maximize the likelihood of the model. Our goal is to find both a graph $\mathcal{G}$ and parameters $\boldsymbol{\theta}_{\mathcal{G}}$ that maximize the likelihood. It is easy to show that to find the maximum-likelihood $(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ pair, we should find the graph structure $\mathcal{G}$ that achieves the highest likelihood when we use the MLE parameters for $\mathcal{G}$. We therefore define

$$\text{score}_L(\mathcal{G} \ : \ \mathcal{D}) = \ell(\langle \mathcal{G}, \hat{\boldsymbol{\theta}}_{\mathcal{G}} \rangle : \mathcal{D}),$$

where $\ell(\langle \mathcal{G}, \hat{\boldsymbol{\theta}}_{\mathcal{G}} \rangle : \mathcal{D})$ is the logarithm of the likelihood function, and $\hat{\boldsymbol{\theta}}_{\mathcal{G}}$ are the maximum-likelihood parameters for $\mathcal{G}$. (It is typically easier to deal with the logarithm of the likelihood.)

The problem with the likelihood score is that it *overfits* the training data. It will learn a model that precisely fits the specifics of the empirical distribution in our training set. This model captures both dependencies that are "true" of the underlying distribution, and dependencies that are artifacts of the specific set of instances that were given as training data. It therefore fails to generalize well to new data cases: these are sampled from the underlying distribution, which is not identical to the empirical distribution in our training set.

However it is reasonable to use the maximum-likelihood score when there are additional mechanisms that disallow overcomplicated structures. For example, learning networks with a fixed indegree. Such a limitation can constrain the tendency to overfit when using the maximum-likelihood score.

Bayesian Score An alternative scoring function is based on Bayesian considerations. Recall that the main principle of the Bayesian approach is that, whenever we have uncertainty over anything, we should place a distribution over it. In this case, we have uncertainty both over structure and over parameters. We therefore define a structure prior $P(\mathcal{G})$ that puts a prior probability on different graph structures, and a parameter prior $P(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G})$ that puts a probability on a different choice of

parameters once the graph is given. By Bayes rule, we have

$$P(\mathcal{G} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathcal{G})P(\mathcal{G})}{P(\mathcal{D})},$$

where, as usual, the denominator is simply a normalizing factor that does not help distinguish between different structures. Then, we define the *Bayesian score* as

$$\text{score}_B(\mathcal{G} \ : \ \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G}), \tag{2.23}$$

The ability to ascribe a prior over structures gives us a way of preferring some structures over others. For example, we can penalize dense structures more than sparse ones. It turns out, however, that this term in the score is almost irrelevant compared to the second term. This first term, $P(\mathcal{D} \mid \mathcal{G})$ takes into consideration our uncertainty over the parameters:

$$P(\mathcal{D} \mid \mathcal{G}) = \int_{\Theta_\mathcal{G}} P(\mathcal{D} \mid \boldsymbol{\theta}_\mathcal{G}, \mathcal{G}) P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G}) d\boldsymbol{\theta}_\mathcal{G}, \tag{2.24}$$

where $P(\mathcal{D} \mid \boldsymbol{\theta}_\mathcal{G}, \mathcal{G})$ is the likelihood of the data given the network $\langle \mathcal{G}, \boldsymbol{\theta}_\mathcal{G} \rangle$ and $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G})$ is our prior distribution over different parameter values for the network $\mathcal{G}$. This term is the *marginal likelihood* of the data given the structure, since we marginalize out the unknown parameters.

Note that the marginal likelihood is different from the maximum-likelihood score. Both terms examine the likelihood of the data given the structure. The maximum-likelihood score returns the maximum of this function. In contrast, the marginal likelihood is the average value of this function, where we average based on the prior measure $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathcal{G})$.

Instantiating this further, if we consider a network with Dirichlet priors, such that $P(\boldsymbol{\theta}_{X_i|\mathbf{pa}_i} \mid \mathcal{G})$ has hyperparameters $\{\alpha^\mathcal{G}_{x_i^j|\boldsymbol{u}_i} : j = 1, \ldots, |X_i|$, then we have that

$$P(\mathcal{D} \mid \mathcal{G}) = \prod_i \prod_{\boldsymbol{u}_i \in Val(\mathbf{Pa}^\mathcal{G}_{X_i})} \frac{\Gamma(\alpha^\mathcal{G}_{X_i|\boldsymbol{u}_i})}{\Gamma(\alpha^\mathcal{G}_{X_i|\boldsymbol{u}_i} + N_{\boldsymbol{u}_i})} \prod_{x_i^j \in Val(X_i)} \left[\frac{\Gamma(\alpha^\mathcal{G}_{x_i^j|\boldsymbol{u}_i} + N_{x_i^j, \boldsymbol{u}_i})}{\Gamma(\alpha^\mathcal{G}_{x_i^j|\boldsymbol{u}_i})} \right],$$

where $\alpha^\mathcal{G}_{X_i|\boldsymbol{u}_i} = \sum_j \alpha^\mathcal{G}_{x_i^j|\boldsymbol{u}_i}$. In practice, we use the logarithm of this formula, which is more manageable to compute numerically.

The Bayesian score is biased toward simpler structures, but as it gets more data, it is willing to recognize that a more complex structure is necessary. In other words, it trades off fit to data with model complexity. To understand behavior, it is useful to consider an approximation to the Bayesian score that better exposes its fundamental properties.

Theorem 2.35
If we use a Dirichlet parameter prior for all parameters in our network, then, as $M \to \infty$, we have that

$$\log P(\mathcal{D} \mid \mathcal{G}) = \ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) - \frac{\log M}{2}\mathrm{Dim}[\mathcal{G}] + O(1),$$

where $\mathrm{Dim}[\mathcal{G}]$ is the number of independent parameters in $\mathcal{G}$.

From this we see that the Bayesian score tends precisely to trade off the likelihood — fit to data — on the one hand, and the model complexity on the other.

This approximation is called the *Bayesian information criterion (BIC) score*:

$$\mathrm{score}_{BIC}(\mathcal{G} \ : \ \mathcal{D}) = \ell(\hat{\boldsymbol{\theta}}_{\mathcal{G}} : \mathcal{D}) - \frac{\log M}{2}\mathrm{Dim}[\mathcal{G}]$$

Our next task is to define the actual priors that are used in the Bayesian score. In the case of the prior of network structures, $P(\mathcal{G})$, note that although this term seems to describe our bias for a certain structure, in fact it plays a relatively minor role. As we can see in theorem 2.35, the logarithm of the marginal likelihood grows linearly with the number of examples, while the prior over structures remains constant. Thus, the structure prior does not play an important role in asymptotic analysis as long as it does not rule out (i.e., assign probability 0) any structure.

In part because of this, it is common to use a uniform prior over structures. Nonetheless, the structure prior can make some difference when we consider small samples. Thus, we might want to encode some of our preferences in this prior. For example, we might penalize edges in the graph, and use a prior

$$P(\mathcal{G}) \propto c^{|\mathcal{G}|},$$

where c is some constant smaller than 1, and $|\mathcal{G}|$ is the number of edges in the graph. In both these choices (the uniform, and the penalty per edge) it suffices to use a value that is proportional to the prior, since the normalizing constant is the same for all choices of $\mathcal{G}$ and hence can be ignored.

It is mathematically convenient to assume that the structure prior satisfies *structure modularity*. This condition requires that the prior $P(\mathcal{G})$ is proportional to a product of terms, where each term relates to one family. Formally,

$$P(\mathcal{G}) \propto \prod_i P(\mathbf{Pa}_{X_i} = \mathbf{Pa}^{\mathcal{G}}_{X_i}),$$

where $P(\mathbf{Pa}_{X_i} = \mathbf{Pa}^{\mathcal{G}}_{X_i})$ denotes the prior probability assigned to choosing the specific set of parents for X_i. Structure priors that satisfy this property do not penalize for global properties of the graph (such as its depth), but only for local properties (such as the number of indegrees).

Next we need to represent our parameter priors. The number of possible structures is superexponential, which makes it difficult to elicit separate parameters for each one.

A simple approach is simply to take some fixed Dirichlet distribution, e.g., $Dirichlet(\alpha, \alpha, \alpha, \ldots, \alpha)$, for every parameter, where α is a predetermined constant. A typical choice is $\alpha = 1$. This prior is often referred to as the *K2 prior*, referring to the name of the system where it was first used.

A more sophisticated approach is called the *BDe prior*. We elicit a prior distribution P' over the entire probability space and an equivalent sample size M' for the set of imaginary samples. We then set the parameters as follows:

$$\alpha_{x_i|\mathbf{pa}_i} = M' \cdot P'(x_i, \mathbf{pa}_i).$$

This choice avoids certain inconsistencies exhibited by the K2 prior. We can represent P' as a Bayesian network, whose structure can represent our prior about the domain structure. Most simply, when we have no prior knowledge, we set P' to be the uniform distribution, i.e., the empty Bayesian network with a uniform marginal distribution for each variable.

The BDe score turns out to satisfy an important property. Two networks are said to be *I-equivalent* if they encode the same set of independence statements. Hence based on observed independencies we cannot distinguish between I-equivalent networks. This suggests that based on observing data cases, we do not expect to distinguish between equivalent networks. The BDe score has the desirable property that I-equivalent networks have the same score, or are *score-equivalent*.

2.4.3.2 Search

We now have a well-defined optimization problem. Our input is

- training set $\mathcal{D}$;
- scoring function (including priors, if needed);
- a set $\mathcal{G}$ of possible network structures (incorporating any prior knowledge).

Our desired output is a network structure (from the set of possible structures) that maximizes the score.

It turns out that, for this discussion, we can ignore the specific choice of score. Our search algorithms will apply unchanged to all three of these scores.

An important property of the scores that affects the efficiency of search is their *decomposability*. A score is decomposable if we can write the score of a network structure $\mathcal{G}$:

$$\text{score}(\mathcal{G} \ : \ \mathcal{D}) = \sum_i \text{FamScore}(X_i \mid \mathbf{Pa}_i^{\mathcal{G}} \ : \ \mathcal{D})$$

All of the scores we have considered are decomposable. Another property that is shared by all these scores is *score equivalence*; if $\mathcal{G}$ is independence-equivalent to $\mathcal{G}'$, then $\text{score}(\mathcal{G} \ : \ \mathcal{D}) = \text{score}(\mathcal{G}' \ : \ \mathcal{D})$.

There are several special cases where structure learning is tractable. We won't go into full details, but two important cases are: (1) learning tree-structured networks and (2) learning networks with known ordering over the variables.

A network is *tree-structured* if each variable has at most one parent. In this case, for decomposable, score-equivalent scores, we can construct an undirected graph, where the weight on an edge $X_i \rightarrow X_j$ is the change in network score if we add X_i as the parent of X_j (note that, because of score-equivalence, this is the same as the change if we add X_j as parent of X_i). We can find a weighted spanning tree of this graph in polynomial time. We can transform the undirected spanning tree into a directed spanning tree by choosing an arbitrary root, and directing edges away from the root.

Another interesting tractable case is the problem of learning a BN structure consistent with some known total order $\prec$ over $\mathcal{X}$ and bounded indegree d. In other words, we restrict attention to structures $\mathcal{G}$ where if $X_i \in \mathbf{Pa}_{X_j}^{\mathcal{G}}$ then $X_i \prec X_j$ and $\left|\mathbf{Pa}_{X_j}^{\mathcal{G}}\right| < d$. For some domains, finding an ordering such as this is relatively straightforward; for example, a temporal flow over the order in which variables take on their values. In this case, for each X_i we can evaluate each possible parent-set of size d from $\{X_1, \ldots, X_{i-1}\}$. This is polynomial in n (but exponential in d).

Unfortunately, the general case, finding an optimally scoring $\mathcal{G}^*$, for bounded degree $d \geq 2$, is $\mathcal{NP}$-hard. Instead of aiming for an algorithm that will always find the highest-scoring network, we resort to heuristic algorithms that attempt to find the best network, but are not guaranteed to do so.

To define the heuristic search algorithm, we must define the search space and search procedure. We can think of a search space as a graph, where each vertex or node is a candidate network structure to be considered, and edges denote possible "moves" that the search procedure can perform. The search procedure defines an algorithm that explores the search space, without necessarily seeing all of it . The simplest search procedure is the greedy one that whenever it is a node chooses to move the neighbor that has the highest score, until it reaches a node that has a better score than all of its neighbors.

To elaborate further, in our case a node in the search space is a complete network structure $\mathcal{G}$ over $\mathcal{X}$. There is a tradeoff in how densely each node is connected with how effective the search will be. If each node has few neighbors, then the search procedure has to consider only few options at each point of the search. Thus, it can afford to evaluate each of these options. However, paths from the initial node to a good one might be long and complex. On the other hand, if each node has many neighbors, there are short paths from each point to another, but we might not be able to pick it, because we don't have time to evaluate all of the options at each step.

A good tradeoff for this problem chooses reasonably few neighbors for each node, but ensures that the "diameter" of the search space remains small. A natural choice of neighbors of a network structure is a set of structures that are identical to it except for small "local" modifications. The most commonly used operators which define the local modifications are

```
Procedure Greedy-Structure-Search (
  G_∅,    // initial network structure
  D    // Fully observed dataset
  score,    // Score
  O,    // A set of search operators
)
1   G_best ← G_∅
2   do
3     G ← G_best
4     Progress ← false
5     for each operator o ∈ O
6       G_o ← o(G)    // Result of applying o on G
7       if G_o is legal structure then
8         if score(G_o : D) > score(G_best : D) then
9           G_best ← G_o
10          Progress ← true
11  while Progress
12
13  return G_best
```

Figure 2.7 Greedy structure search algorithm, with an arbitrary scoring function $\text{score}(\mathcal{G} : \mathcal{D})$.

- add an edge;
- delete an edge;
- reverse an edge.

In other words, if we consider the node $\mathcal{G}$, then the neighboring nodes in the search space are those where we change one edge, either by adding one, deleting one, or reversing the orientation of one. We only consider operations that result in legal networks (i.e., acyclic networks satisfying any constraints such as bounded indegree).

This definition of search space is quite natural and has several desirable properties. First, notice that the diameter of the search space is at most n^2. That is, there is a relatively short path between any two networks we choose. To see this, note that if we consider traversing a path from $\mathcal{G}_1$ to $\mathcal{G}_2$, we can start by deleting all edges in $\mathcal{G}_1$ that do not appear in $\mathcal{G}_2$, and then we can add the edges that are in $\mathcal{G}_2$ and not in $\mathcal{G}_1$. Clearly, the number of steps we take is bounded by the total number of edges we can have, n^2.

Second, recall that the score of a network $\mathcal{G}$ is a sum of local scores. The operations we consider result in changing only one local score term (in the case of addition or deletion of an edge) or two (in the case of edge reversal). Thus, they result in a local change in the score — the "main" mass of the score remains the same. This implies that there is some sense of "continuity" in the score of neighboring nodes.

The search methods most commonly used are *local search procedures*. Such search procedures are characterized by the following design: they keep a "current" candidate node. At each iteration they explore some of the neighboring nodes, and

then decide to make a "step" to one of the neighbors and make it the current candidate. These iterations are repeated until some termination condition. In other words, local search procedures can be thought of as keeping one pointer into the search space and moving it around.

One of the simplest, and often used, search procedures is the *greedy hill-climbing procedure*. The intuition is simple. As the name suggests, at each step we take the step that leads to the largest improvement in the score. The actual details of the procedure are shown in figure 2.7. We pick an initial network structure $\mathcal{G}$ as a starting point; this network can be the empty one, a random choice, the best tree, or a network obtained from some prior knowledge. We compute its score. We then consider all of the neighbors of $\mathcal{G}$ in the space — all of the legal networks obtained by applying a single operator to $\mathcal{G}$ — and compute the score for each of them. We then apply the change that leads to the best improvement in the score. We continue this process until no modification improves the score.

We can improve on the performance of greedy hill-climbing by using more clever search algorithms. Some common extensions are:

- **TABU search:** Keep a list of K most recently visited structures and avoid them, i.e., apply the best move that leads to a structure not on the list. This approach deals with local maxima whose "hill" has fewer than K structures.
- **Random restarts:** Once stuck, apply some fixed number of random edge changes and then restart the greedy search. At the end of the search, select the best structure encountered anywhere on the trajectory. This approach can escape from the basin of one local maximum to another.
- **Simulated annealing:** Evaluate operators in random order. If the randomly selected operator induces an uphill step, move to the resulting structure. (Note: it does not have to be the best of the current neighbors.) If the operator induces a downhill step, apply it with probability inversely proportional to the reduction in score. A *temperature parameter* determines the probability of taking downhill steps. As the search progress, the temperature decreases, and the algorithm becomes less likely to take a downhill step.

2.5 Conclusion

This chapter presented a condensed description of graphical models, including their representation, inference algorithms, and learning algorithms. Many topics have not been covered; we refer the reader to [8] for a more complete description.

References

[1] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125(1-2):3–17, 2001.

[2] W. Buntine. Chain graphs for learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1995.

[3] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York, 1999.

[4] D. Heckerman. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Seattle, WA, 1996.

[5] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, New York, 2001.

[6] M. I. Jordan, editor. *Learning in Graphics Models*. The MIT Press, Cambridge, MA, 1998.

[7] M. I. Jordan. Graphical models. *Statistical Science (Special issue on Bayesian Statistics)*, 19:140–155, 2004.

[8] D. Koller and N. Friedman. BNs and beyond, 2007. To appear.

[9] S. Lauritzen. *Graphical Models*. Oxford University Press, New York, 1996.

[10] S. Lauritzen and N. Wermuth. Graphical models for association between variables, some of which are qualitative and some quantitative. *Annals of Statistics*, 17(1):31–57, 1989.

[11] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, CA, 1988.

[12] K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proceedings of the National Conference on Artificial Intelligence*, 1997.