11 *Inference as Optimization*

11.1 Introduction

In the previous chapters we examined exact inference. We have seen that for many networks we can perform exact inference efficiently. As we have seen, the computational and space complexity of the clique tree is exponential in the tree-width of the network. This means that the exact algorithms we examined become infeasible for networks with a large tree-width. In many real-life applications, we encounter such networks. This motivates examination of approximate inference methods that are applicable to networks where exact inference is intractable.

☞ **In this chapter we consider a class of approximate inference methods, where the approximation arises from constructing an approximation to the target distribution P_Φ. This approximation takes a simpler form that allows for inference.** In general, the simpler approximating form exploits a local factorization structure that is similar in nature to the structure exploited by graphical models.

The specific algorithms we consider differ in many details, and yet they share some common conceptual principles. We now review these principles to provide a common framework for the remaining presentation. In each method, we define a target class $\mathcal{Q}$ of "easy" distributions Q and then search for an instance within that class that is the "best" approximation to P_Φ. Queries can then be answered using inference on Q rather than on P_Φ. All of the methods we describe optimize (roughly) the same target function for measuring the similarity between Q and P_Φ.

constrained optimization

This approach reformulates the inference task as one of optimizing an objective function over the class $\mathcal{Q}$. This problem falls into the category of *constrained optimization*. Such problems can be solved using a variety of different methods. Thus, the formulation of inference from this perspective opens the door to the application of a range of techniques developed in the optimization literature. Currently, the technique most often used in the setting of graphical models is one based on the use of *Lagrange multipliers*, which we review in appendix A.5.3. This method produce a set of equations that characterize the optima of the objective. In our setting, this characterization takes the form of a set of fixed-point equations that define each variable in terms of others. A particularly compelling and elegant result is that

☞ **the fixed-point equations derived from the constrained energy optimization, for any of the methods we describe, can be viewed as passing messages over a graph object.** Indeed, as we will show, even the standard sum-product algorithm for clique trees (algorithm 10.2) can be rederived from this perspective. Moreover, many other message passing algorithms follow from the same derivation.

Methods in this class fall into three main categories. The first category includes methods that

use clique-tree message passing schemes on structures other than trees. This class of methods, which includes the famous *loopy belief propagation* algorithm, can be understood as optimizing approximate versions of the energy functional. The second category includes methods that use message propagation on clique trees with approximate messages. This class of methods, often known as the *expectation propagation* algorithm, maximize the exact energy functional, but with relaxed consistency constraints on the representation Q. Finally, in the third category there are methods that generalize the *mean field* method originating in statistical physics. These methods use the exact energy functional, but they restrict attention to a class $\mathcal{Q}$ consisting of distributions Q that have a particular simple factorization. This factorization is chosen to be simple enough to ensure that we can perform inference with Q.

More broadly, each of these algorithms can be described from two perspectives: as a procedural description of a message passing algorithm, or as an optimization problem consisting of an objective and a constraint space. Historically, the message passing algorithm generally originated first, sometimes long before the optimization interpretation was understood. However, the optimization perspective provides a much deeper understanding of these methods, and it shows that message passing is only one way of performing the optimization; it also helps point the way toward useful generalizations. In the ensuing discussion, we usually begin the presentation of each class of methods by describing a simple variant of the algorithm, providing a concrete manifestation to ground the concepts. We then present the optimization perspective on the algorithm, allowing a deeper understanding of the algorithm. Finally, we discuss generalizations of the simple algorithm, often ones that are derived directly from the optimization perspective.

11.1.1 Exact Inference Revisited ⋆

Before considering approximate inference methods, we start by casting exact inference as an optimization problem. The concepts we introduce here will serve in the discussion of the following approximate inference methods.

Assume we have a factorized distribution of the form

$$P_\Phi(\mathcal{X}) = \frac{1}{Z} \prod_{\phi \in \Phi} \phi(\boldsymbol{U}_\phi), \tag{11.1}$$

where the factors ϕ in Φ comprise the distribution, and the variables $\boldsymbol{U}_\phi = \mathit{Scope}[\phi] \subseteq \mathcal{X}$ are the scope of each factor. For example, the factors might be CPDs in a Bayesian network, generally restricted by an evidence set, or they might be potentials in a Markov network. We are interested in answering queries about the distribution P_Φ. These include queries about marginal probabilities of variables and queries about the partition function Z. As we discussed, if P_Φ is a Bayesian network with instantiated evidence on some variables, then the partition function Z is the probability of the evidence.

Recall that the end product of belief propagation is a calibrated cluster tree. Also recall that a calibrated set of beliefs for the cluster tree represents a distribution. In exact inference we find a set of calibrated beliefs that represent $P_\Phi(\mathcal{X})$. That is, we find beliefs that match the distribution represented by given set of initial potentials. Thus, we can view exact inference as searching over the set of distributions $\mathcal{Q}$ that are representable by the cluster tree to find a distribution Q^* that matches P_Φ.

Intuitively, we can rephrase this question as searching for a calibrated distribution that is as close as possible to P_Φ. There are many possible ways of measuring the distance between two distributions, such as the Euclidean distance (L_2), or the L_1 distance and the related variational distance (see chapter 2). As we will see, our main challenge, however, is our aim to avoid performing inference with the distribution P_Φ; in particular, we cannot effectively compute marginal distributions in P_Φ. Hence, we need methods that allow us to optimize the distance between Q and P_Φ without answering hard queries about P_Φ. 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_Φ without performing reasoning with it.

Recall that the relative entropy between P_1 and P_2 is defined as[1]

$$\boldsymbol{D}(P_1 \| P_2) = \boldsymbol{E}_{P_1}\left[\ln \frac{P_1(\mathcal{X})}{P_2(\mathcal{X})}\right].$$

Also recall that the relative entropy is always nonnegative, 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_Φ that minimizes the relative entropy.

However, as we discussed, the relative entropy is not symmetric — $\boldsymbol{D}(P_1 \| P_2) \neq \boldsymbol{D}(P_2 \| P_1)$. In section 8.5, we discussed the use of relative entropy for projecting a distribution into a restricted class; this projection can aim to minimize either $\boldsymbol{D}(P_\Phi \| Q)$, via the *M-projection*, or $\boldsymbol{D}(Q \| P_\Phi)$, via the *I-projection*. A priori, it might appear that the M-projection is more appropriate, since one of the main information-theoretic justifications for the relative entropy $\boldsymbol{D}(P_\Phi \| Q)$ is the number of bits lost when coding a true message distribution P_Φ using an (approximate) estimate Q. However, as the discussion of section 8.5.2 shows, computing the M-projection $Q = \arg\min_Q \boldsymbol{D}(P_\Phi \| Q)$ — requires that we compute marginals of P_Φ and is therefore equivalent to running inference in P_Φ. Somewhat surprisingly, as we show in the subsequent discussion, this does not apply to I-projection: we can exploit the structure of P_Φ to optimize $\arg\min_Q \boldsymbol{D}(Q \| P_\Phi)$ efficiently, *without* running inference in P_Φ.

M-projection

I-projection

To summarize this discussion, we want to search for a distribution Q that minimizes $\boldsymbol{D}(Q \| P_\Phi)$. To define and analyze this optimization problem formally, we also need to specify the objects we optimize over. Suppose we are given a cluster tree structure $\mathcal{T}$ for P_Φ. That is, $\mathcal{T}$ satisfies the running intersection property and the family preservation property. Moreover, suppose we are given a set of beliefs

$$\boldsymbol{Q} = \{\beta_i : i \in \mathcal{V}_\mathcal{T}\} \cup \{\mu_{i,j} : (i\text{–}j) \in \mathcal{E}_\mathcal{T}, \}$$

where $\boldsymbol{C}_i$ denotes clusters in $\mathcal{T}$, β_i denotes beliefs over $\boldsymbol{C}_i$, and $\mu_{i,j}$ denotes beliefs over $\boldsymbol{S}_{i,j}$ of edges in $\mathcal{T}$.

As in definition 10.6, the set of beliefs in $\mathcal{T}$ defines a distribution Q by the formula

$$Q(\mathcal{X}) = \frac{\prod_{i \in \mathcal{V}_\mathcal{T}} \beta_i}{\prod_{(i\text{–}j) \in \mathcal{E}_\mathcal{T}} \mu_{i,j}}. \tag{11.2}$$

1. Note that, until now, we defined the relative entropy and other information-theoretic terms, such as mutual information, using logarithms to base 2. As will become apparent, in the context of the discussion in this chapter, the natural logarithm (base e) is more suitable. This change is a simple rescaling of the relevant information-theoretic quantities and does not change their basic properties.

calibration

marginal consistency

(See section 10.2.3.) Due to the *calibration* requirement, the set of beliefs $\boldsymbol{Q}$ satisfies the *marginal consistency constraints* if, for each $(i\text{–}j) \in \mathcal{E}_T$, the beliefs on $\boldsymbol{S}_{i,j}$ are the marginal of β_i (and β_j). Recall that theorem 10.4 shows that if $\boldsymbol{Q}$ is a set of calibrated beliefs for $\mathcal{T}$ and Q is the distribution defined by equation (11.2), then

$$\begin{aligned} \beta_i[\boldsymbol{c}_i] &= Q(\boldsymbol{c}_i) \\ \mu_{i,j}[\boldsymbol{s}_{i,j}] &= Q(\boldsymbol{s}_{i,j}). \end{aligned}$$

Thus, the beliefs correspond to marginals of the distribution Q defined by equation (11.2).

Thus, we are now searching over a set of distributions Q that are representable by a set of beliefs $\boldsymbol{Q}$ over the cliques and sepsets in a particular clique tree structure $\mathcal{T}$. Note that when deciding on the representation of $\boldsymbol{Q}$ we are actually making two decisions: We are deciding both on the space of distributions that we are considering (all distributions for which $\mathcal{T}$ is an I-map), and on the representation of these distributions (as a set of calibrated clique beliefs). Both of these decisions are significant components in the specification of our optimization problem.

With these definitions in hand, we can now view exact inference as maximizing $-\boldsymbol{D}(Q\|P_\Phi)$ over the space of calibrated sets $\boldsymbol{Q}$.

CTree-Optimize-KL:

Find $\boldsymbol{Q} = \{\beta_i : i \in \mathcal{V}_T\} \cup \{\mu_{i,j} : (i\text{–}j) \in \mathcal{E}_T\}$
maximizing $-\boldsymbol{D}(Q\|P_\Phi)$
subject to

$$\mu_{i,j}[\boldsymbol{s}_{i,j}] = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{c}_i) \quad \forall (i\text{–}j) \in \mathcal{E}_T, \forall \boldsymbol{s}_{i,j} \in Val(\boldsymbol{S}_{i,j})$$

$$\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) = 1 \quad \forall i \in \mathcal{V}_T.$$

In solving this optimization problem, we conceptually examine different configurations of beliefs that satisfy the marginal consistency constraints, and we select the configuration that maximizes the objective. Such an exhaustive examination, of course, is impossible to perform in practice. However, there are effective solutions to this problem that find the maximum point. We have already seen that, if $\mathcal{T}$ is a proper cluster tree for the set of original potentials Φ, we know that there is a set $\boldsymbol{Q}$ that induces, via equation (11.2), a distribution $Q = P_\Phi$. Because this solution achieves a relative entropy of 0, which is the highest value possible, it is the unique global optimum of this optimization.

Theorem 11.1 *If $\mathcal{T}$ is an I-map of P_Φ, then there is a unique solution to* CTree-Optimize-KL.

This optimum can be found using the exact inference algorithms we developed in chapter 10.

11.1.2 The Energy Functional

The preceding discussion suggests a strategy for constructing approximations of P_Φ. Instead of searching over the space of all calibrated cluster trees, we can search over a space of "simpler" distributions. In this search we will not find a distribution equivalent to P_Φ, yet we might

find one that is reasonably close to P_Φ. Moreover, as part of the design of the target set of distributions, we can ensure that these distributions are ones in which we can perform inference efficiently.

One problem that we will face is that the target of the optimization $\boldsymbol{D}(Q\|P_\Phi)$ is unwieldy for direct optimization. The relative entropy term contains an explicit summation over all possible instantiations of $\mathcal{X}$, an operation that is infeasible in practice. However, since we know the form of $\ln P_\Phi(\xi)$ from equation (11.1), we can exploit its structure to rewrite the relative entropy in a simpler form, as shown in the following theorem.

Theorem 11.2

$\boldsymbol{D}(Q\|P_\Phi) = \ln Z - F[\tilde{P}_\Phi, Q]$

energy functional

where $F[\tilde{P}_\Phi, Q]$ *is the* energy functional

$$F[\tilde{P}_\Phi, Q] = \boldsymbol{E}_Q\left[\ln \tilde{P}(\mathcal{X})\right] + \boldsymbol{H}_Q(\mathcal{X}) = \sum_{\phi\in\Phi} \boldsymbol{E}_Q[\ln \phi] + \boldsymbol{H}_Q(\mathcal{X}). \tag{11.3}$$

PROOF

$$\boldsymbol{D}(Q\|P_\Phi) = \boldsymbol{E}_Q[\ln Q(\mathcal{X})] - \boldsymbol{E}_Q[\ln P_\Phi(\mathcal{X})]. \tag{11.4}$$

Using the product form of P_Φ, we have that

$$\ln P_\Phi(\mathcal{X}) = \sum_{\phi\in\Phi} \ln \phi(\boldsymbol{U}_\phi) - \ln Z.$$

Moreover, recall that $\boldsymbol{H}_Q(\mathcal{X}) = -\boldsymbol{E}_Q[\ln Q(\mathcal{X})]$. Plugging these into equation (11.4), we get

$$\begin{aligned}\boldsymbol{D}(Q\|P_\Phi) &= -\boldsymbol{H}_Q(\mathcal{X}) - \boldsymbol{E}_Q\left[\sum_{\phi\in\Phi} \ln \phi(\boldsymbol{U}_\phi)\right] + \boldsymbol{E}_Q[\ln Z] \\ &= -F[\tilde{P}_\Phi, Q] + \ln Z.\end{aligned}$$

■

Importantly, the term $\ln Z$ does not depend on Q. Hence, minimizing the relative entropy $\boldsymbol{D}(Q\|P_\Phi)$ is equivalent to maximizing the energy functional $F[\tilde{P}_\Phi, Q]$.

This latter term relates to concepts from statistical physics, and it is the negative of what

free energy

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 book, we continue to use the standard terminology of energy functional.

energy term

The energy functional contains two terms. The first, called the *energy term*, involves expectations of the logarithms of factors in Φ. Here, each factor in Φ appears as a separate term. Thus, if the factors that comprise Φ are small, each expectation deals with relatively few variables. The difficulties in dealing with these expectations depends on the properties of the distribution Q. Assuming that inference is "easy" in Q, we should be able to evaluate such expectations

entropy term

relatively easily. The second term, called the *entropy term*, is the entropy of Q. Again, the choice of Q determines whether we can evaluate this term. However, we will see that, for the choices we make, this term will also be tractable.

11.1.3 Optimizing the Energy Functional

In the remainder of this chapter, **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, since $\boldsymbol{D}(Q\|P_\Phi) \geq 0$, we have that

$$\ln Z \geq F[\tilde{P}_\Phi, Q]. \tag{11.5}$$

lower bound

That is, the energy functional is a *lower bound* on 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, $\boldsymbol{D}(Q\|P_\Phi)$ is small), then we can get a good lower-bound approximation to Z. The fact that this approximation is a lower bound will play an important role in later chapters on learning.

variational method

In this chapter, we explore inference methods that can be viewed as strategies for optimizing the energy functional. These kinds of methods are often referred to as *variational methods*. The name refers to a general strategy in which we want to solve a problem by introducing new variational parameters that increase the degrees of freedom over which we optimize. Each choice of these parameters gives an approximate answer. We then attempt to optimize the variational parameters to get the best approximation. In our case, the task is to answer queries about P_Φ, and the variational parameters describe the distribution Q. In the methods we consider, we vary these parameters to try to find a good approximation to the target query.

11.2 Exact Inference as Optimization

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

As we have already seen, the optimization problem CTree-Optimize-KL has a unique solution. We start by reformulating the optimization problem in terms of the energy functional. As we have seen, maximizing the energy functional is equivalent to minimizing the relative entropy between Q and P_Φ.

Once we restrict attention to calibrated cluster trees, we can further simplify the objective function. More precisely, 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 beliefs in $\boldsymbol{Q}$. This form reveals the structure in the distribution, and it is therefore a much better starting point for further analysis. As we will see, this form is also the basis for our approximations in subsequent sections.

Definition 11.1

factored energy functional

Given a cluster tree $\mathcal{T}$ with a set of beliefs $\boldsymbol{Q}$ and an assignment α that maps factors in P_Φ to clusters in $\mathcal{T}$, we define the factored energy functional*:*

$$\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] = \sum_{i \in \mathcal{V}_\mathcal{T}} \boldsymbol{E}_{\boldsymbol{C}_i \sim \beta_i}[\ln \psi_i] + \sum_{i \in \mathcal{V}_\mathcal{T}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(i-j) \in \mathcal{E}_\mathcal{T}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}), \tag{11.6}$$

where ψ_i is the initial potential assigned to $\boldsymbol{C}_i$:

$$\psi_i = \prod_{\phi, \alpha(\phi)=i} \phi,$$

■

and $\boldsymbol{E}_{\boldsymbol{C}_i \sim \beta_i}[\cdot]$ denotes expectation on the value $\boldsymbol{C}_i$ given the beliefs β_i

Before we prove that the energy functional is equivalent to its factored variant, let us first study its components. The first term is a sum of terms of the form $\boldsymbol{E}_{\boldsymbol{C}_i \sim \beta_i}[\ln \psi_i]$. Recall that ψ_i 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 beliefs β_i are a distribution over $Val(\boldsymbol{C}_i)$. We can therefore compute the expectation $\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) \ln \psi_i$. The last two terms are entropies of the beliefs associated with the clusters and sepsets in the tree. The important benefit of this reformulation is that all the terms are *local*, in the sense that they refer to a specific belief factor. As we will see, this will make our tasks much simpler.

Proposition 11.1 *If $\boldsymbol{Q}$ is a set of calibrated beliefs for $\mathcal{T}$, and Q is defined by equation (11.2), then*

$$\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] = F[\tilde{P}_\Phi, Q].$$

PROOF Note that $\ln \psi_i = \sum_{\phi, \alpha(\phi)=i} \ln \phi$. Moreover, since $\beta_i(\boldsymbol{c}_i) = Q(\boldsymbol{c}_i)$, we conclude that

$$\sum_i \boldsymbol{E}_{\boldsymbol{C}_i \sim \beta_i}[\ln \psi_i] = \sum_\phi \boldsymbol{E}_{\boldsymbol{C}_i \sim Q}[\ln \phi].$$

It remains to show that

$$\boldsymbol{H}_Q(\mathcal{X}) = \sum_{i \in \mathcal{V}_\mathcal{T}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(i-j) \in \mathcal{E}_\mathcal{T}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}).$$

This equality follows directly from equation (11.2) and theorem 10.4. ■

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 beliefs. Marginal consistency is a constraint on the beliefs that requires neighboring beliefs to agree on the marginal distribution on their joint subset. It is equivalent to requiring that the beliefs be calibrated. Thus, we pose the following constrained optimization procedure:

CTree-Optimize:

Find $\boldsymbol{Q} = \{\beta_i : i \in \mathcal{V}_\mathcal{T}\} \cup \{\mu_{i,j} : (i-j) \in \mathcal{E}_\mathcal{T}\}$
maximizing $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$
subject to

$$\mu_{i,j}[\boldsymbol{s}_{i,j}] = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{c}_i) \qquad \forall (i-j) \in \mathcal{E}_\mathcal{T}, \forall \boldsymbol{s}_{i,j} \in Val(\boldsymbol{S}_{i,j}) \tag{11.7}$$

$$\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) = 1 \qquad \forall i \in \mathcal{V}_\mathcal{T} \tag{11.8}$$

$$\beta_i(\boldsymbol{c}_i) \geq 0 \qquad \forall i \in \mathcal{V}_\mathcal{T}, \boldsymbol{c}_i \in Val(\boldsymbol{C}_i). \tag{11.9}$$

The constraints equation (11.7), equation (11.8), and equation (11.9) ensure that the beliefs in $\boldsymbol{Q}$ are calibrated and represent legal distributions (exercise 11.2).

11.2.1 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 *fixed-point equations*. As we show, these equations turn out to be the update equations in the sum-product belief-propagation procedure (CTree-SP-calibrate in algorithm 10.2). Thus, if we turn these equations into an iterative algorithm, as we will describe, we obtain precisely the belief propagation algorithm in clique trees. We note that for this derivation and other similar ones later in the chapter, we restrict attention to models where all of the potentials are strictly positive (contain no zero entries). Although the results generally hold also for the case of deterministic potentials (zero entries), the proofs are considerably more complex and are outside the scope of this book.

Recall that a stationary point of a function is either a local maximum, a local minimum, or a saddle point. In the optimization problem CTree-Optimize, there is a single global maximum (see theorem 11.1). Although we do not show it here, one can show that it is also the only stationary point (see exercise 11.3), and thus once we find a stationary point, we know that we have found the maximum.

Lagrange multipliers

We want to *characterize* this stationary point by a set of equations that must hold when the choice of beliefs in $\boldsymbol{Q}$ is at the stationary point. Recall that our aim is to maximize the function $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$ under the consistency constraints. The method of *Lagrange multipliers*, reviewed in appendix A.5.3, provides us with tools for dealing with constrained optimization. Because the characterization of the stationary point is of central importance to later developments, we examine how to construct such a characterization using the method of Lagrange multipliers.

When using the method of Lagrange multipliers, we start by defining a Lagrangian with a Lagrange multiplier for each of the constraints on the function we want to optimize. In our case, we have the constraints in equation (11.7) and equation (11.8). We note that, in principle, we also need to introduce a Lagrange multiplier for the inequality constraint that ensures that all beliefs are nonnegative. However, as we will see, the assumption that factors are strictly positive implies that the beliefs we construct in the solution to the optimization problem will be nonnegative, and thus we do not need to enforce these constraints actively. We therefore obtain the following Lagrangian:

$$\begin{aligned}\mathcal{J} &= \tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] \\ &\quad - \sum_{i \in \mathcal{V}_T} \lambda_i \left(\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) - 1 \right) \\ &\quad - \sum_i \sum_{j \in \mathrm{Nb}_i} \sum_{\boldsymbol{s}_{i,j}} \lambda_{j \to i}[\boldsymbol{s}_{i,j}] \left(\sum_{\boldsymbol{c}_i \sim \boldsymbol{s}_{i,j}} \beta_i(\boldsymbol{c}_i) - \mu_{i,j}[\boldsymbol{s}_{i,j}] \right),\end{aligned}$$

where Nb_i is the neighbors of $\boldsymbol{C}_i$ in the clique tree. We introduce Lagrange multipliers λ_i for each beliefs factor β_i to ensure that it sums to 1. We also introduce, for each pair of neighboring cliques i and j and assignment to their sepset $\boldsymbol{s}_{i,j}$, a Lagrange multiplier $\lambda_{j \to i}[\boldsymbol{s}_{i,j}]$ to ensure

that the marginal distribution of $\boldsymbol{s}_{i,j}$ in β_j is consistent with its value in the sepset beliefs $\mu_{i,j}$. (Note that we also introduce another Lagrange multiplier for the direction $i \rightarrow j$.)

Remember that $\mathcal{J}$ is a function of the clique beliefs $\{\beta_i\}$, the sepset beliefs $\{\mu_{i,j}\}$, and the Lagrange multipliers. To find the maximum of the Lagrangian, we take its partial derivatives with respect to $\beta_i(\boldsymbol{c}_i)$, $\mu_{i,j}[\boldsymbol{s}_{i,j}]$, and the Lagrange multipliers. These last derivatives reconstruct the original constraints. The first two types of derivatives require some work. Differentiating the Lagrangian (see exercise 11.1), we get that

$$\begin{aligned}
\frac{\partial}{\partial \beta_i(\boldsymbol{c}_i)} \mathcal{J} &= \ln \psi_i[\boldsymbol{c}_i] - \ln \beta_i(\boldsymbol{c}_i) - 1 - \lambda_i - \sum_{j \in \mathrm{Nb}_i} \lambda_{j \rightarrow i}[\boldsymbol{s}_{i,j}] \\
\frac{\partial}{\partial \mu_{i,j}[\boldsymbol{s}_{i,j}]} \mathcal{J} &= \ln \mu_{i,j}[\boldsymbol{s}_{i,j}] + 1 + \lambda_{i \rightarrow j}[\boldsymbol{s}_{i,j}] + \lambda_{j \rightarrow i}[\boldsymbol{s}_{i,j}].
\end{aligned}$$

At the stationary point, these derivatives are zero. Equating each derivative to 0, rearranging terms, and exponentiating, we get

$$\begin{aligned}
\beta_i(\boldsymbol{c}_i) &= \exp\{-1 - \lambda_i\} \psi_i[\boldsymbol{c}_i] \prod_{j \in \mathrm{Nb}_i} \exp\{-\lambda_{j \rightarrow i}[\boldsymbol{s}_{i,j}]\} \\
\mu_{i,j}[\boldsymbol{s}_{i,j}] &= \exp\{-1\} \exp\{-\lambda_{i \rightarrow j}[\boldsymbol{s}_{i,j}]\} \exp\{-\lambda_{j \rightarrow i}[\boldsymbol{s}_{i,j}]\}.
\end{aligned}$$

These equations describe beliefs as functions of terms of the form $\exp\{-\lambda_{i \rightarrow j}[\boldsymbol{s}_{i,j}]\}$. In fact, $\mu_{i,j}$ is a product of two such terms (and a constant). This suggests that these terms play the role of a message $\delta_{i \rightarrow j}$. To make this more explicit, we define

$$\delta_{i \rightarrow j}[\boldsymbol{s}_{i,j}] = \exp\left\{-\lambda_{i \rightarrow j}[\boldsymbol{s}_{i,j}] - \frac{1}{2}\right\}.$$

(We add the term $-\frac{1}{2}$ to deal with the additional $\exp\{-1\}$ term, but since this is a multiplicative constant, it is not that crucial.) We can now rewrite the resulting system of equations as

$$\begin{aligned}
\beta_i(\boldsymbol{c}_i) &= \exp\left\{-\lambda_i - 1 + \frac{1}{2}|\mathrm{Nb}_i|\right\} \psi_i(\boldsymbol{c}_i) \prod_{j \in \mathrm{Nb}_i} \delta_{j \rightarrow i}[\boldsymbol{s}_{i,j}] \\
\mu_{i,j}[\boldsymbol{s}_{i,j}] &= \delta_{i \rightarrow j}[\boldsymbol{s}_{i,j}] \delta_{j \rightarrow i}[\boldsymbol{s}_{i,j}].
\end{aligned}$$

Combining these equations with equation (11.7), we now rewrite the message $\delta_{i \rightarrow j}$ as a function of other messages:

$$\begin{aligned}
\delta_{i \rightarrow j}[\boldsymbol{s}_{i,j}] &= \frac{\mu_{i,j}[\boldsymbol{s}_{i,j}]}{\delta_{j \rightarrow i}[\boldsymbol{s}_{i,j}]} \\
&= \frac{\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{C}_i, \boldsymbol{s}_{i,j})}{\delta_{j \rightarrow i}[\boldsymbol{s}_{i,j}]} \\
&= \exp\left\{-\lambda_i - 1 + \frac{1}{2}|\mathrm{Nb}_i|\right\} \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i(\boldsymbol{c}_i) \prod_{k \in \mathrm{Nb}_i - \{j\}} \delta_{k \rightarrow i}[\boldsymbol{s}_{i,k}].
\end{aligned}$$

Note that the term $\exp\left\{-\lambda_i - 1 + \frac{1}{2}|\mathrm{Nb}_i|\right\}$ is a constant (since it does not depend on $\boldsymbol{c}_i$), and when we combine these equations with equation (11.8), we can solve for λ_i to ensure that

this constant normalizes the clique beliefs β_i. We note that if the original factors define a distribution that sums to 1, then the solution for λ_i that satisfies equation (11.8) will be one where $\lambda_i = \frac{1}{2}|\text{Nb}_i - 1|$, that is, the normalizing constant is 1.

This derivation proves the following result.

Theorem 11.3

A set of beliefs $\boldsymbol{Q}$ is a stationary point of CTree-Optimize *if and only if there exists a set of factors $\{\delta_{i\to j}[\boldsymbol{S}_{i,j}] : (i\text{–}j) \in \mathcal{E}_{\mathcal{T}}\}$ such that*

$$\delta_{i\to j} \propto \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \left(\prod_{k \in \text{Nb}_i - \{j\}} \delta_{k\to i} \right) \tag{11.10}$$

and moreover, we have that

$$\begin{aligned} \beta_i &\propto \psi_i \left(\prod_{j \in \text{Nb}_i} \delta_{j\to i} \right) \\ \mu_{i,j} &= \delta_{j\to i} \cdot \delta_{i\to j}. \end{aligned}$$

fixed-point equations

This theorem characterizes 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, equation (11.10) defines each message in terms of *other* messages, allowing an easy iterative approach to solving the fixed point equations. These same themes appear in all the approaches we will discuss later in this chapter.

11.2.2 Inference as Optimization

The fixed-point characterization of theorem 11.3 focuses on the relationships that hold at the maximum point (or points). However, they also hint at a way of achieving these relationships. Intuitively, a change in $\boldsymbol{Q}$ that reduces the differences between the left-hand and right-hand side of these equations will get us closer to a maximum point. The most direct way of reducing such discrepancies is to apply 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.

More precisely, we initialize all of the $\delta_{i\to j}$'s to 1 and then iteratively apply equation (11.10), 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 a clique tree), we can guarantee that this process converges to a solution satisfying all of the equations in equation (11.10); the other equations are now easy to satisfy.

Each assignment step defined by a fixed-point equation corresponds to a message passing step, where an outgoing message $\delta_{i\to j}$ is defined in terms of incoming messages $\delta_{k\to i}$. The fact that the process requires multiple assignments to converge corresponds to the fact that inference requires multiple message passing steps. In this specific example, a particular order of applying the fixed-point equation reconstructs the sum-product message passing algorithm in cluster trees shown in algorithm 10.2. As we will see, however, when we consider other variants of the optimization problem, the associated fixed-point equations result in new algorithms.

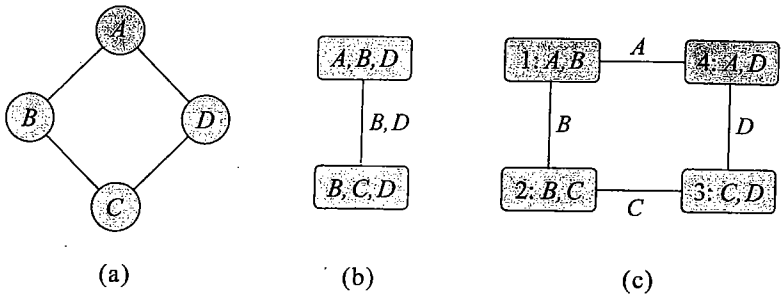


Figure 11.1 An example of a cluster graph. (a) A simple network. (b) A clique tree for the network in (a). (c) A cluster graph for the same network.

11.3 Propagation-Based Approximation

In this section, we consider approximation methods that use exactly the same message propagation as in exact inference. However, these propagation schemes use a general-purpose cluster graph, as in definition 10.1, rather than a clique tree. Since the constraints defining a clique tree were crucial in ensuring exact inference, the message-propagation schemes that use cluster graphs will generally not provide the correct answers.

We begin by defining the general message passing algorithm in a cluster graph. We then show that it can be derived, using the same process as in the previous section, from a set of fixed-point equations induced by the stationary points of an approximate energy functional.

11.3.1 A Simple Example

Consider the simple Markov network of figure 11.1a. Recall that, to perform exact inference within this network, we must first reduce it to a tree, such as the tree of figure 11.1b. Inference in this simple tree involves passing messages over the sepset, which consists of the variables $\{B, D\}$.

Now suppose that, instead, we perform inference as follows. We set up four clusters, which correspond to the four initial potentials: $C_1 = \{A, B\}$, $C_2 = \{B, C\}$, $C_3 = \{C, D\}$, $C_4 = \{A, D\}$. We connect these clusters to each other as shown in the *cluster graph* of figure 11.1c. Note that this cluster graph contains loops (undirected cycles), and is therefore not a tree; such graphs are often called *loopy*. Nevertheless, we can apply the belief-update propagation algorithm CTree-BU-calibrate (algorithm 10.3). Although in our discussion of that algorithm we assumed that the input is a tree, there is nothing in the algorithm itself that relies on that fact. In each step of the algorithm we propagate a message between neighboring clusters. Thus, it is perfectly applicable to a general cluster graph that may not necessarily be a tree.

The clusters in this cluster graph are smaller than those in the clique tree of figure 11.1b; therefore, the message passing steps are less expensive. But what is the result of this procedure? Suppose we propagate messages in the following order $\mu_{1,2}$, $\mu_{2,3}$, $\mu_{3,4}$, and then $\mu_{4,1}$. In the first message, the $\{A, B\}$ cluster passes information to the $\{B, C\}$ cluster through a marginal distribution on B. This information is then propagated to next cluster, and so on. However, in the final message $\mu_{4,1}$, this information reaches the original cluster, but this time as observation

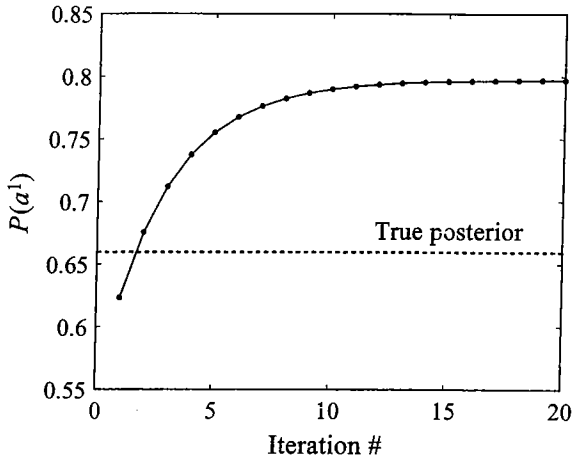


Figure 11.2 An example run of loopy belief propagation in the simple network of figure 11.1a. In this run, all potentials prefer consensus assignments over nonconsensus ones. In each iteration, we perform message passing for all the edges in the cluster graph of figure 11.1b.

about the values of A. As an example, suppose all clusters favor consensus joint assignments; that is, $\beta_1(a^0, b^0)$ and $\beta_1(a^1, b^1)$ are much larger than $\beta_1(a^1, b^0)$ and $\beta_1(a^0, b^1)$, and similarly for the other beliefs. Thus, if the message $\mu_{1,2}$ strengthens the belief that $B = b^1$, then the message $\mu_{2,3}$ will increase the belief in $C = c^1$ and so on. Once we get around the loop, the message $\mu_{4,1}$ will strengthen the support in $A = a^1$. This message will be incorporated into the cluster as though it were independent evidence that did not depend on the initial propagation. Now, if we continue to apply the same sequence of propagations again, we will keep increasing the beliefs in the assignment of $A = a^1$. This behavior is illustrated in figure 11.2. As we can see, in later iterations the procedure overestimates the marginal probability of A. However, the effect of the "feedback" decays until the iterations converge.

This simple experiment already suggests several important issues we need to consider:

convergence

- In the case of cluster trees, we described a sequence of message propagations that calibrate the tree in two passes. Once the tree is calibrated, additional message propagations do not change any of the beliefs. Thus, we can say that the propagation process has *converged*. When we consider our example, it seems clear that **the process may not converge in two passes, since information from one pass will circulate and affect the next round. Indeed, it is far from clear that the propagation of beliefs necessarily converges at all.**

- In the case of cluster trees, we saw that, in a calibrated tree, each cluster of beliefs is the joint marginal of the cluster variables. As our example suggests, for cluster graph propagation, the beliefs on A are not necessarily the marginal probability in P_Φ. Thus, the question is the relationship between the calibrated cluster graph and the actual probability distribution.

Before we address these questions, we present the algorithm in more general terms.

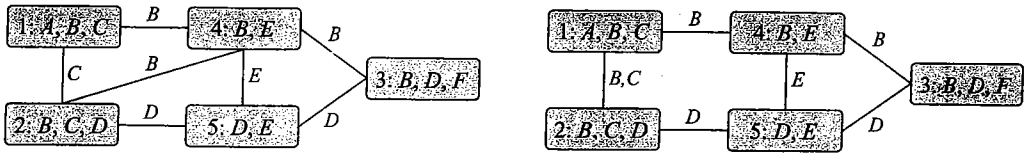


Figure 11.3 **Two examples of generalized cluster graph for an MRF** with potentials over $\{A, B, C\}$, $\{B, C, D\}$, $\{B, D, F\}$, $\{B, D\}$ and $\{D, E\}$.

Box 11.A — Case Study: Turbocodes and loopy belief propagation. *The idea of propagating messages in loopy graphs was first proposed in the early days of the field, in parallel with the introduction of the first exact inference algorithms. As we discussed in box 9.B, one of the first inference algorithms was Pearl's message passing for singly connected Bayesian networks (polytrees). In his 1988 book, Pearl says:*

> *When loops are present, the network is no longer singly connected and local propagation schemes will invariably run into trouble ... If we ignore the existence of loops and permit the nodes to continue communicating with each other as if the network were singly connected, messages may circulate indefinitely around the loops and the process may not converge to a stable equilibrium ... Such oscillations do not normally occur in probabilistic networks ... which tend to bring all messages to some stable equilibrium as time goes on. However, this asymptotic equilibrium is not coherent, in the sense that it does not represent the posterior probabilities of all nodes of the networks.*

loopy belief propagation

As a consequence of these problems, the idea of loopy belief propagation *was largely abandoned for many years.*

Surprisingly, the revival of loopy belief propagation is due to a seemingly unrelated advance in coding theory. The area of coding addresses the problem of sending messages over a noisy channel, and recovering it from the garbled result. Formally, the coding task can be defined as follows. We wish to send a k-bit message $u_1, \ldots, u_k$. We code *the message using a number of bits $x_1, \ldots, x_n$, which are then sent over the noisy channel, resulting in a set of (possibly corrupted) outputs $y_1, \ldots, y_n$, which can be either discrete or continuous. Different channels introduce noise in different ways: In a simple Gaussian noise model, each bit sent is corrupted independently by the addition of some Gaussian noise; another simple model flips each bit independently with some probability; more complex channel models, where noise is added in a correlated way to consecutive bits, are also used. The* message decoding *task is to recover an estimate $\hat{u}_1, \ldots, \hat{u}_k$ from $y_1, \ldots, y_n$. The* bit error rate *is the probability that a bit is ultimately decoded incorrectly. This error rate depends on the code and decoding algorithm used and on the amount of noise in the channel. The* rate *of a code is k/n — the ratio between the number of bits in the message and the number of bits used to transmit it.*

message decoding

For example, a very simple repetition code takes each bit and transmits it three times, then decodes the bit by majority voting on the three (noisy) copies received. If the channel corrupts each bit with probability p, the bit error rate of this algorithm is $p^3 + 3p^2$, which, for reasonable values

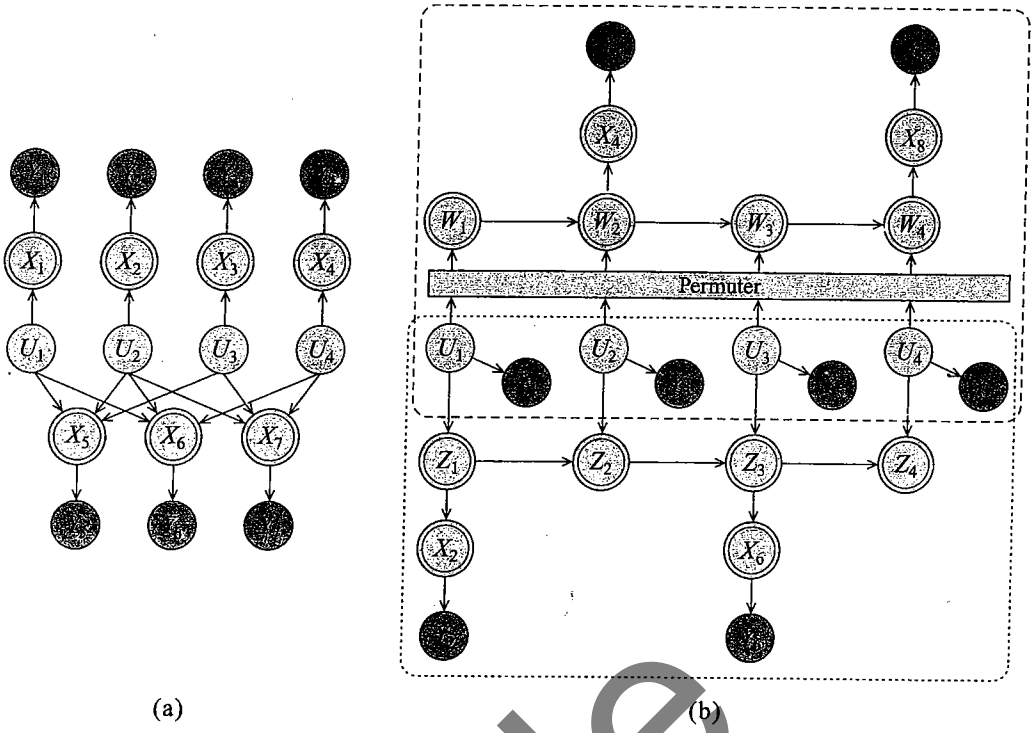


Figure 11.A.1 — Two examples of codes (a) A $k = 4, n = 7$ parity check code, where every four message bits are sent along with three bits that encode parity checks. (b) A $k = 4, n = 8$ turbocode. Here, the $\boldsymbol{X}^a$ bits X_1, X_3, X_5, X_7 are simply the original bits U_1, U_2, U_3, U_4 and are omitted for clarity of the diagram; the $\boldsymbol{X}^b$ bits use a *shift register* — a state bit that changes with each bit of the message, where the ith state bit depends on the $(i-1)$st state bit and on the ith message bit. The code uses two shift registers, one applied to the original message bits and one to a set of permuted message bits (using some predetermined permutations). The sent bits contain both the original message bits and some number of the state bits.

of p, is much lower than p. The rate of this code is $1/3$*, because for every message bit, three bits are transmitted. In general, we can get better bit error rates by increasing the redundancy of the code, so we want to compare the bit error rate of different codes that have the same rate. Repetition codes are some of the least efficient codes designed. Figure 11.A.1a shows a simple rate 4/7 parity check code, where every four message bits are sent along with three bits that encode parity checks (exclusive ORs) of different subsets of the four bits.*

In 1948, Claude Shannon provided a theoretical analysis of the coding problem (Shannon 1948). For a given rate, Shannon provided an upper bound on the maximum noise level that can be tolerated while still achieving a certain bit error rate, no matter which code is used. Shannon also showed that there exist channel codes that achieve this limit, but his proof was nonconstructive —

he did not present practical encoders and decoders that achieve this limit.

Since Shannon's landmark result, multiple codes were suggested. However, despite a gradual improvement in the quality of the code (bit-error rate for a given noise level), none of the codes even came close to the Shannon limit. The big breakthrough came in the early 1990s, when Berrou et al. (1993) came up with a new scheme that they called a turbocode, *which, empirically, came much closer to achieving the Shannon limit than any other code proposed up to that point. However, their decoding algorithm had no theoretical justification, and, while it seemed to work well in real examples, could be made to diverge or converge to the wrong answer. The second big breakthrough was the subsequent realization that turbocodes were simply performing belief propagation on a Bayesian network representing the probability model for the code and the channel noise.*

turbocode

To understand this, we first observe that message decoding can easily be reformulated as a probabilistic inference task: We have a prior over the message bits $\mathbf{U} = \langle U_1, \ldots, U_k \rangle$, a (usually deterministic) function that defines how a message is converted into a sequence of transmitted bits $X_1, \ldots, X_n$, and another (stochastic) model that defines how the channel randomly corrupts the X_i's to produce Y_i's. The decoding task can then be viewed as finding the most likely joint assignment to $\mathbf{U}$ given the observed message bits $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, or (alternatively) as finding the posterior $P(U_i \mid \mathbf{y})$ for each bit U_i. The first task is a MAP inference task, and the second task one of computing posterior probabilities. Unfortunately, the probability distribution is of high dimension, and the network structure of the associated graphical model is quite densely connected and with many loops.

The turbocode approach, as first proposed, comprised both a particular coding scheme, and the use of a message passing algorithm to decode it. The coding scheme transmits two sets of bits: one set comprises the original message bits $\mathbf{X}^a = \langle X_1^a, \ldots, X_k^a \rangle = \mathbf{u}$, and the second some set $\mathbf{X}^b = \langle X_1^b, \ldots, X_k^b \rangle$ of transformed bits (like the parity check bits, but more complicated). The received bits then can also be partitioned into the noisy $\mathbf{y}^a, \mathbf{y}^b$. Importantly, the code is designed so that the message can be decoded (albeit with errors) using either $\mathbf{y}^a$ or $\mathbf{y}^b$. The turbocoding algorithm then works as follows: It uses the model of $\mathbf{X}^a$ (trivial in this case) and of the channel noise to compute a posterior probability over $\mathbf{U}$ given $\mathbf{y}^a$. It then uses that posterior $\pi_a(U_1), \ldots, \pi_a(U_k)$ as a prior over $\mathbf{U}$ and computes a new posterior over $\mathbf{U}$, using the model for $\mathbf{X}^b$ and the channel, and $\mathbf{y}^b$ as the evidence, to compute a new posterior $\pi_b(U_1), \ldots, \pi_b(U_k)$. The "new information," which is $\pi_b(U_i)/\pi_a(U_i)$, is then transmitted back to the first decoder, and the process repeats until a stopping criterion is reached. In effect, the turbocoding idea was to use two weak coding schemes, but to "turbocharge" them using a feedback loop. Each decoder is used to decode one subset of received bits, generating a more informed distribution over the message bits to be subsequently updated by the other. The specific method proposed used particular coding scheme for the $\mathbf{X}^b$ bits, illustrated in figure 11.A.1b.

This process looked a lot like black magic, and in the beginning, many people did not even believe that the algorithm worked. However, when the empirical success of these properties was demonstrated conclusively, an attempt was made to understand its theoretical properties. McEliece et al. (1998) subsequently showed that the specific message passing procedure proposed by Berrou et al. is precisely an application of belief propagation (with a particular message passing schedule) to the Bayesian network representing the turbocode (as in figure 11.A.1b).

☞ **This revelation had a tremendous impact on both the coding theory community and the graphical models community. For the former, loopy belief propagation provides a general-purpose algorithm for decoding a large family of codes. By separating the**

algorithmic question of decoding from the question of the code design, it allowed the development of many new coding schemes with improved properties. These codes have come much, much closer to the Shannon limit than any previous codes, and they have revolutionized both the theory and the practice of coding. For the graphical models community, it was the astounding success of loopy belief propagation for this application that led to the resurgence of interest in these approaches, and subsequently to much of the work described in this chapter.

11.3.2 Cluster-Graph Belief Propagation

cluster graph

The basis for our message passing algorithm is the *cluster graph* of definition 10.1, first defined in section 10.1.1. In that section, we required that cluster graphs be trees and that they respect the running intersection property. Those requirements led us to the definition of a clique tree. Here, we remove the first of these two assumptions, allowing inference to be performed on a loopy cluster graph. However, we still wish to require a variant of the running intersection property that is generalized to this case: for any two clusters containing X, there is precisely one path between them over which information about X can be propagated.

Definition 11.2 running intersection property

We say that $\mathcal{U}$ satisfies 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 there is a single path between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$ for which $X \in \boldsymbol{S}_e$ for all edges e in the path.* ■

This generalized running intersection property implies that all edges associated with X form a tree that spans all the clusters that contain X. Thus, intuitively, there is only a single path by which information *that is directly about* X can flow in the graph. Both parts of this assumption are significant. The fact that some path must exist forces information about X to flow between all clusters that contain it, so that, in a calibrated cluster graph, all clusters must agree about the marginal distribution of X. The fact that there is at most one path prevents information about X from cycling endlessly in a loop, making our beliefs more extreme due to "cyclic arguments."

Importantly, however, since the graph is not necessarily a tree, the same pair of clusters might also be connected by other paths. For example, in the cluster graph of figure 11.3a, we see that the edges labeled with B form a subtree that spans all the clusters that contain B. However, there are loops in the graph. For example, there are two paths from $\boldsymbol{C}_3 = \{B, D, F\}$ to $\boldsymbol{C}_2 = \{B, C, D\}$. The first, through $\boldsymbol{C}_4$, propagates information about B, and the second, through $\boldsymbol{C}_5$, propagates information about D. Thus, we can still get circular reasoning, albeit less directly than we would in a graph that did not satisfy the running intersection property; we return to this point in section 11.3.8. Note that while in the case of trees the definition of running intersection implied that $\boldsymbol{S}_{i,j} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$, in a graph this equality is no longer enforced by the running intersection property. For example, cliques $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ in figure 11.3a have B in common, but $\boldsymbol{S}_{1,2} = \{C\}$.

In clique trees, inference is performed by calibrating beliefs. In a cluster graph, we can also associate cluster $\boldsymbol{C}_i$ with *beliefs* β_i. We now say that a cluster graph is *calibrated* if for each

beliefs

calibrated cluster graph

edge $(i–j)$, connecting the clusters $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, we have that

$$\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i = \sum_{\boldsymbol{C}_j - \boldsymbol{S}_{i,j}} \beta_j;$$

that is, the two clusters agree on the marginal of variables in $\boldsymbol{S}_{i,j}$. Note that this definition is weaker than cluster tree calibration, since the clusters do not necessarily agree on the joint marginal of all the variables they have in common, but only on those variables in the sepset. However, if a calibrated cluster graph satisfies the running intersection property, then the marginal of a variable X is identical in all the clusters that contain it.

Algorithm 11.1 Calibration using sum-product belief propagation in a cluster graph

```
    Procedure CGraph-SP-Calibrate (
      Φ,    // Set of factors
      U     // Generalized cluster graph Φ
    )
1     Initialize-CGraph
2     while graph is not calibrated
3       Select (i–j) ∈ E_U
4       δ_{i→j}(S_{i,j}) ← SP-Message(i, j)
5     for each clique i
6       β_i ← ψ_i · ∏_{k∈Nb_i} δ_{k→i}
7     return {β_i}

    Procedure Initialize-CGraph (
      U
    )
1     for each cluster C_i
2       β_i ← ∏_{φ : α(φ)=i} φ
3     for each edge (i–j) ∈ E_U
4       δ_{i→j} ← 1
5       δ_{j→i} ← 1
6

    Procedure SP-Message (
      i,    // sending clique
      j     // receiving clique
    )
1     ψ(C_i) ← ψ_i · ∏_{k∈(Nb_i−{j})} δ_{k→i}
2     τ(S_{i,j}) ← ∑_{C_i−S_{i,j}} ψ(C_i)
3     return τ(S_{i,j})
```

How do we calibrate a cluster graph? Because calibration is a local property that relates adjoining clusters, we want to try to ensure that each cluster is sharing information with its

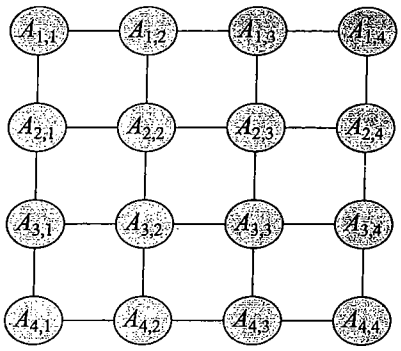


Figure 11.4 An example of a 4×4 two-dimensional grid network

neighbors. From the perspective of a single cluster C_i, there is not much difference between a cluster graph and a cluster tree. The cluster is related to each neighboring cluster through an edge that conveys information on variables in the sepset. Thus, we can transmit information by simply having one cluster pass a message to the other.

In our preceding example, we applied this modification to the belief-update message propagation algorithm. Initially, this transformation appears inapplicable to the sum-product calibration of algorithm 10.2, since the sum-product algorithm sends a message only when the sending clique is ready to transmit, that is, when all other incoming messages have been received. In the loopy cluster graph, initially, there is no cluster that has received any incoming messages. Thus, no cluster is ready to transmit, and the algorithm is deadlocked. However, in section 10.3, we showed that the two algorithms are actually equivalent; that is, any sequence of sum-production propagation steps can be emulated by the same sequence of belief-update propagation steps and leads to the same beliefs. In this transformation, we have that $\mu_{i,j} = \delta_{i \to j} \delta_{j \to i}$. Thus, we can construct a "deadlock-free" variant of the sum-product message passing algorithm simply by initializing all messages $\delta_{i \to j} = \mathbf{1}$. This initialization of the sum-product algorithm is equivalent to the standard initialization of the belief update algorithm, in which $\mu_{i,j} = \mathbf{1}$. Importantly, in this variant of the sum-product algorithm, each cluster begins with all of the incoming messages initialized, and therefore it can send any of the outgoing messages at any time, without waiting for any other cluster.

Algorithm 11.1 shows the sum-product message passing algorithm for cluster graphs; other than the fact that the algorithm is applied to graphs rather than trees, the algorithm is identical to CTree-SP-Calibrate. In much the same manner, we can adapt CTree-BU-Calibrate to define a procedure CGraph-BU-Calibrate that operates over cluster graphs using belief-update message passing steps. Both of these algorithms are instances of a general class of algorithms called *cluster-graph belief propagation*, which passes messages over cluster graphs.

cluster-graph belief propagation

Before we continue, we note that cluster-graph belief propagation can be significantly cheaper than performing exact inference. A canonical example of a class of networks that is compactly representable yet hard for inference is the class of grid-structured Markov networks (such as the ones used in image analysis; see box 4.B). In these networks, each variable $A_{i,j}$ corresponds to a point on a two-dimensional grid. Each edge in this network corresponds to a potential

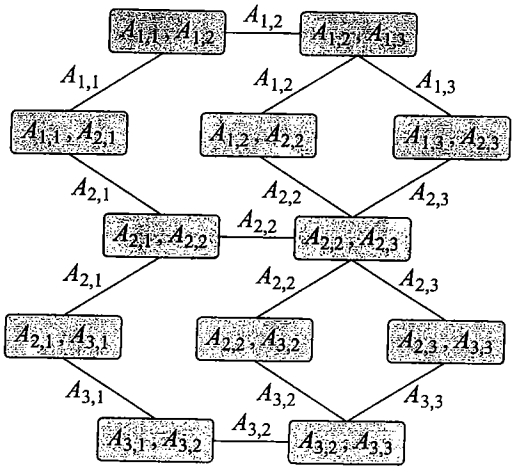


Figure 11.5 An example of generalized cluster graph for a 3×3 grid network

between adjacent points on the grid, with $A_{i,j}$ connected to the four nodes $A_{i-1,j}$, $A_{i+1,j}$, $A_{i,j-1}$, $A_{i,j+1}$ (except for nodes $A_{i,j}$ on the boundary of the grid); see figure 11.4. Such a network has only pairwise potentials, and hence it is very compactly represented. Yet, exact inference requires separating sets, which are as large as cutsets in the grid. Hence, in an $n \times n$ grid, exact computation is exponential in n.

However, we can easily create a generalized cluster graph for grid networks that directly corresponds to the factors in the network. In this cluster graph, each cluster represents beliefs over two neighboring grid variables, and each cluster has a small number of adjoining edges that connect it to other clusters that share one of the two variables. See figure 11.5 for an example for a small 3×3 grid. (Note that there are several ways of constructing such a cluster graph; this figure represents one reasonable choice.) A round of propagations in the generalized cluster graph is linear in the size of the grid (quadratic in n).

11.3.3 Properties of Cluster-Graph Belief Propagation

What can we say about the properties and guarantees provided by cluster-graph belief propagation? We now consider some of the ramifications of the "mechanical" operation of message passing in the graph. Later, when we discuss cluster-graph belief propagation as an optimization procedure, we will revisit this question from a different perspective.

11.3.3.1 Reparameterization

reparameterization

cluster graph invariant

Recall that in section 10.2.3 we showed that belief propagation maintains an invariant property. This allowed us to show that the convergence point represents a *reparameterization* of the original distribution. We can directly extend this property to cluster graphs, resulting in a *cluster graph invariant.*

Theorem 11.4 *Let $\mathcal{U}$ be a generalized cluster graph over a set of factors Φ. Consider the set of beliefs $\{\beta_i\}$ and sepsets $\{\mu_{i,j}\}$ at any iteration of* CGraph-BU-Calibrate*; then*

$$\tilde{P}_\Phi(\mathcal{X}) = \frac{\prod_{i \in \mathcal{V}_\mathcal{U}} \beta_i[\boldsymbol{C}_i]}{\prod_{(i-j) \in \mathcal{E}_\mathcal{U}} \mu_{i,j}[\boldsymbol{S}_{i,j}]}.$$

where $\tilde{P}_\Phi(\mathcal{X}) = \prod_{\phi \in \Phi} \phi$ is the unnormalized distribution defined by Φ.

PROOF Recall that $\beta_i = \psi_i \prod_{j \in \mathrm{Nb}_i} \delta_{j \to i}$ and that $\mu_{i,j} = \delta_{j \to i} \delta_{i \to j}$. We now have

$$\begin{aligned}
\frac{\prod_{i \in \mathcal{V}_\mathcal{U}} \beta_i[\boldsymbol{C}_i]}{\prod_{(i-j) \in \mathcal{E}_\mathcal{U}} \mu_{i,j}[\boldsymbol{S}_{i,j}]} &= \frac{\prod_{i \in \mathcal{V}_\mathcal{U}} \psi_i[\boldsymbol{C}_i] \prod_{j \in \mathrm{Nb}_i} \delta_{j \to i}[\boldsymbol{S}_{i,j}]}{\prod_{(i-j) \in \mathcal{E}_\mathcal{U}} \delta_{j \to i}[\boldsymbol{S}_{i,j}] \delta_{i \to j}[\boldsymbol{S}_{i,j}]} \\
&= \prod_{i \in \mathcal{V}_\mathcal{U}} \psi_i[\boldsymbol{C}_i] \\
&= \prod_{\phi \in \Phi} \phi(\boldsymbol{U}_\phi) = \tilde{P}_\Phi(\mathcal{X}).
\end{aligned}$$

Note that the second step is based on the fact that each message $\delta_{i \to j}$ appears exactly once in the numerator and the denominator and thus can be canceled. ■

☞ **This property shows that cluster-graph belief propagation preserves all of the information about the original distribution. In particular, it does not "dilute" the original factors by performing propagation along loops. Hence, we can view the process as trying to represent the original factors anew in a more useful form.**

11.3.3.2 Tree Consistency

Recall that theorem 10.4 implies that, in a calibrated cluster tree, the belief over a cluster is the marginal of the distribution. Thus, in a calibrated cluster tree, we can "read off" the marginals of P_Φ locally from clusters that contain them. More precisely, by normalizing the beliefs factor β_i (so that it sums to 1), we get the marginal distribution over $\boldsymbol{C}_i$. An obvious question is whether a corresponding property holds for cluster-graph belief propagation. Suppose we manage to calibrate a generalized cluster graph and normalize the resulting beliefs; do we have an interpretation for the beliefs in each cluster?

As we saw in our simple example (figure 11.2), the beliefs we compute by BU-message are not necessarily marginals of P_Φ, but rather an approximation. Can we say anything about the quality of this approximation? To characterize the beliefs we get at the end of the process, we can use the cluster tree invariant property applied to subtrees of a cluster graph.

Consider a subtree $\mathcal{T}$ of $\mathcal{U}$; that is, a subset of clusters and edges that together form a tree that satisfies the running intersection property. For example, consider the cluster graph of figure 11.1c. If we remove one of the clusters and its incident edges, we are left with a proper cluster tree. Note that the running intersection property is not necessarily as easy to achieve in general, since removing some edges from the cluster graph may result in a graph that violates the running intersection property relative to a variable, necessitating the removal of additional edges, and so on.

Once we select a tree $\mathcal{T}$, we can think of it as defining a distribution

$$P_{\mathcal{T}}(\mathcal{X}) = \frac{\prod_{i \in \mathcal{V}_{\mathcal{T}}} \beta_i(\boldsymbol{C}_i)}{\prod_{(i-j) \in \mathcal{E}_{\mathcal{T}}} \mu_{i,j}[\boldsymbol{S}_{i,j}]}.$$

If the cluster graph is calibrated, then by definition so is $\mathcal{T}$. And so, because $\mathcal{T}$ is a tree that satisfies the running intersection property, we can apply theorem 10.4, and we conclude that

$$\beta_i(\boldsymbol{C}_i) = P_{\mathcal{T}}(\boldsymbol{C}_i). \tag{11.11}$$

tree consistency

That is, the beliefs over $\boldsymbol{C}_i$ in the tree are the marginal of $P_{\mathcal{T}}$, a property called *tree consistency*.

As a concrete example, consider the cluster graph of figure 11.1c. Removing the cluster $\boldsymbol{C}_4 = \{A, D\}$, we are left with a proper cluster tree $\mathcal{T}$. The preceding argument implies that once we have calibrated the cluster graph, we have $\beta_1(A, B) = P_{\mathcal{T}}(A, B)$. This result suggests that $\beta_1(A, B) \neq P_\Phi(A, B)$; to show this formally, contrast equation (11.11) with theorem 11.4. We see that the tree distribution involves some of the terms that define the joint distribution. Thus, we can conclude that

$$P_{\mathcal{T}}(A, B, C, D) = P_\Phi(A, B, C, D)\frac{\mu_{3,4}[D]\mu_{1,4}[A]}{\beta_4(A, D)}.$$

We see that unless $\beta_4(A, D) = \mu_{3,4}[D]\mu_{1,4}[A]$, $P_{\mathcal{T}}$ will be different from P_Φ. This conclusion suggests that, in this example, the beliefs $\beta_1(A, B)$ in the calibrated cluster graph are not the marginal $P_\Phi(A, B)$.

Clearly, we can apply the same type of reasoning using other subtrees of $\mathcal{U}$. And so we reach the surprising conclusion that equation (11.11) must hold with respect to every cluster tree embedded in $\mathcal{U}$. In our example, we can see that by removing a single cluster, we can construct three different trees that contain $\boldsymbol{C}_1$. The same beliefs $\beta_1(A, B)$ are the marginal of the three distributions defined by each of these trees. While these three distributions agree on the joint marginal of A and B, they can differ on the joint marginal distributions of other pairs of variables.

cluster graph residual

Moreover, these subtrees allow us to get insight about the quality of the marginal distributions we read from the calibrated cluster graph. Consider our example again: we can use the *residual* term $\frac{\mu_{3,4}[D]\mu_{1,4}[A]}{\beta_4(A,D)}$ to analyze the error in the marginal distribution. In this simple example, this analysis is fairly straightforward (see exercise 11.4).

In other cases, the analysis can be more complex. For example, suppose we want to find a subtree in the cluster graph for a grid (e.g., figure 11.5). To construct a tree, we must remove a nontrivial number of clusters. More precisely, because each cluster corresponds to an edge in the grid, a cluster tree corresponds to a subtree of the grid. For an $n \times n$ grid, such a tree will have at most $n^2 - 1$ edges of the $2n(n-1)$ edges in the grid. Thus, each cluster tree contains about half of the clusters in the original cluster graph. In such a situation the residual term is more complex, and we cannot necessarily evaluate it.

11.3.4 Analyzing Convergence ⋆

A key question regarding the belief propagation algorithm is whether and when it converges. Indeed, there are many networks for which belief propagation does not converge; see box 11.C.

Although we cannot hope for convergence in all cases, it is important to understand when this algorithm does converge. We know that if the cluster graph is a tree then the algorithm will converge. Can we find other classes of cluster graphs for which we can prove convergence?

synchronous BP

One method of analyzing convergence is based on the following important perspective on belief propagation. This analysis is easier to perform on a variant of BP called *synchronous BP* that performs all of the message updates simultaneously. Consider the update step that takes all of the messages $\boldsymbol{\delta}^t$ at a particular iteration t and produces a new set of messages $\boldsymbol{\delta}^{t+1}$ for the next step. Letting Δ be the space of all possible messages in the cluster graph, we can view the belief-propagation update operator as a function $G_{BP} : \Delta \mapsto \Delta$. Consider the standard sum-product message update:

$$\delta'_{i \to j} \propto \sum_{C_i - S_{i,j}} \psi_i \cdot \prod_{k \in (\mathrm{Nb}_i - \{j\})} \delta_{k \to i},$$

BP operator

where we normalize each message to sum to 1; this renormalization step is essential to avoid a degenerate convergence to the $\mathbf{0}$ message. We can now define the *BP operator* as the function that simultaneously takes one set of messages and computes a new one:

$$G_{BP}(\{\delta_{i \to j}\}) = \{\delta'_{i \to j}\}.$$

The question of convergence of the algorithm now reduces to one of asking whether repeated applications of the operator G_{BP} are guaranteed to converge.

One interesting, albeit strong, condition that guarantees convergence is the *contraction property*:

Definition 11.3 contraction

For a number $\alpha \in [0, 1)$, an operator G over a metric space $(\Delta, \boldsymbol{D}(;))$ is an α-contraction *relative to the distance function $\boldsymbol{D}(;)$ if, for any $\boldsymbol{\delta}, \boldsymbol{\delta}' \in \Delta$, we have that:*

$$\boldsymbol{D}(G(\boldsymbol{\delta}); G(\boldsymbol{\delta}')) \leq \alpha \boldsymbol{D}(\boldsymbol{\delta}; \boldsymbol{\delta}'). \tag{11.12}$$

■

In other words, an operator is a contraction if its application to two points in the space is guaranteed to decrease the distance between them by at least some constant factor $\alpha < 1$.

A basic result in analysis shows that, under fairly weak conditions, if an operator G is a contraction, we have that repeated applications of G are guaranteed to converge to a unique fixed point:

Proposition 11.2 fixed-point

Let G be an α-contraction of a complete metric space $(\Delta, \boldsymbol{D}(;))$. Then there is a unique fixed-point *$\boldsymbol{\delta}^*$ for which $G(\boldsymbol{\delta}^*) = \boldsymbol{\delta}^*$. Moreover, for any $\boldsymbol{\delta}$, we have that*

$$\lim_{n \to \infty} G^n(\boldsymbol{\delta}) = \boldsymbol{\delta}^*.$$

The proof is left as an exercise (exercise 11.5).

Indeed, the *contraction rate* α can be used to provide bounds on the rate of convergence of the algorithm to its unique fixed point: To reach a point that is guaranteed to be within ϵ of $\boldsymbol{\delta}^*$, it suffices to apply G the following number of times:

$$\log_\alpha \frac{\epsilon}{\mathrm{diameter}(\Delta)},$$

where $\text{diameter}(\Delta) = \max_{\boldsymbol{\delta},\boldsymbol{\delta}' \in \Delta} \boldsymbol{D}(\boldsymbol{\delta}; \boldsymbol{\delta}')$.

Applying this analysis to the operator G induced by the belief-propagation message update is far from trivial. This operator is complex and nonlinear, because it involves both multiplying messages and a renormalization step. A review of these analyses is outside the scope of this book. At a high level, these results show that if the factors in the network are fairly "smooth," one can guarantee that the synchronous BP operator is a contraction and hence converges to a unique fixed point. We describe one of the simplest of these results, in order to give a flavor for this type of analysis.

This analysis applies to synchronous loopy belief propagation over a pairwise Markov network with two-valued random variables $X_i \in \{-1, +1\}$. Specifically, we assume that the network model is parameterized as follows:

$$P(x_1, \ldots, x_n) = \frac{1}{Z} \exp\left(\sum_{(i,j)} \epsilon_{i,j}(x_i, x_j) + \sum_i \epsilon_i(x_i),\right),$$

where we assume for simplicity of notation that $\epsilon_{i,j} = 0$ when X_i and X_j are not neighbors in the network.

hyperbolic tangent

We begin by introducing some notation. The *hyperbolic tangent* function is defined as:

$$\tanh(w) = \frac{e^w - e^{-w}}{e^w + e^{-w}} = \frac{e^{2w} - 1}{e^{2w} + 1}.$$

The hyperbolic tangent has a shape very similar to the sigmoid function of figure 5.11a. The following condition can be shown to suffice for G_{BP} to be a contraction, and hence for the convergence of belief propagation to a unique fixed point:

$$\max_i \max_{j \in \text{Nb}_i} \sum_{k \in \text{Nb}_i - \{j\}} \tanh|\epsilon_{k,i}| < 1. \tag{11.13}$$

Intuitively, this expression measures the total extent to which i's neighbors other than j can influence the message from i to j. The larger the magnitude of the parameters in the network, the larger this sum.

The analysis of the more general case is significantly more complex but shares the same intuitions. At a very high level, if we can place strong bounds on the skew of the parameters in a factor:

$$\max_{\boldsymbol{x},\boldsymbol{x}'} \phi(\boldsymbol{x})/\phi(\boldsymbol{x}'),$$

we can guarantee convergence of belief propagation. Intuitively, the lower the skew of the factors in our network, the more each message update "smoothes out" differences between entries in the messages, and therefore also makes different messages more similar to each other.

While the conditions that underlie these theorems are usually too stringent to hold in practice, this analysis does provide useful insight. First, it suggests that **networks with potentials that are closer to deterministic are more likely to have problems with convergence, an observation that certainly holds in practice.** Second, although global contraction throughout the space is a very strong assumption, a contraction property in a region of the space may be plausible, guaranteeing convergence of the algorithm if it winds up (or is initialized) in this region. These results and their ramifications are only now being explored.

11.3.5 Constructing Cluster Graphs

So far, we have taken the cluster graph to be given. However, the choice of cluster graph is generally far from obvious, and it can make a significant difference to the algorithm. Recall that, even in exact inference, more than one clique tree can be used to perform inference for a given distribution. However, while these different trees can vary in their computational cost, they all give rise to the same answers. **In the case of cluster graph approximations, different graphs can lead to very different answers. Thus, when selecting a cluster graph, we have to consider trade-offs between cost and accuracy, since cluster graphs that allow fast propagation might result in a poor approximation.**

It is important to keep in mind that the structure of the cluster graph determines the propagation steps the algorithm can perform, and thus dictate what type of information is passed during the propagations. These choices directly influence the quality of the results.

Example 11.1

Consider, for example, the cluster graphs $\mathcal{U}_1$ and $\mathcal{U}_2$ of figure 11.3a and figure 11.3b. Both are fairly similar, yet in $\mathcal{U}_2$ the edge between C_1 and C_2 involves the marginal distribution over B and C. On the other hand, in $\mathcal{U}_1$, we propagate the marginal only over C. Intuitively, we expect inference in $\mathcal{U}_2$ to better capture the dependencies between B and C. For example, assume that the potential of C_1 introduces strong correlations between B and C (say $B = C$). In $\mathcal{U}_2$, this correlation is conveyed to C_2 directly. In $\mathcal{U}_1$, the marginal on C is conveyed on the edge $(1,2)$, while the marginal on B is conveyed through C_4. In this case, the strong dependency between the two variables is lost. In particular, if the marginal on C is diffuse (close to uniform), then the message C_1 sends to C_4 will also have a uniform distribution on B, and from C_2's perspective the messages on B and C will appear as two independent variables. ■

On the other hand, if we introduce many messages between clusters or increase the scope of these messages, we run the risk of constructing a tree that violates the running intersection property. And so, we have to worry about methods that ensure that the resulting structure is a proper cluster graph. We now consider several approaches for constructing cluster graphs.

11.3.5.1 Pairwise Markov Networks

pairwise Markov networks

We start with 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. Many problems are naturally formulated as pairwise Markov networks, including the grid networks we discussed earlier and Boltzmann distributions (see box 4.C). Indeed, if we are willing to transform our variables, any distribution can be reformulated as a pairwise Markov network (see exercise 11.10).

One straightforward transformation of such a network into a cluster graph is as follows: 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 between the cluster $C_{(i,j)}$ that corresponds to the edge X_i—X_j and the clusters C_i and C_j that correspond to the univariate factors over X_i and X_j. Figure 11.6 illustrates this construction in the case of a 3 by 3 grid network.

Because there is a direct correspondence between the clusters in the cluster graphs and vari-

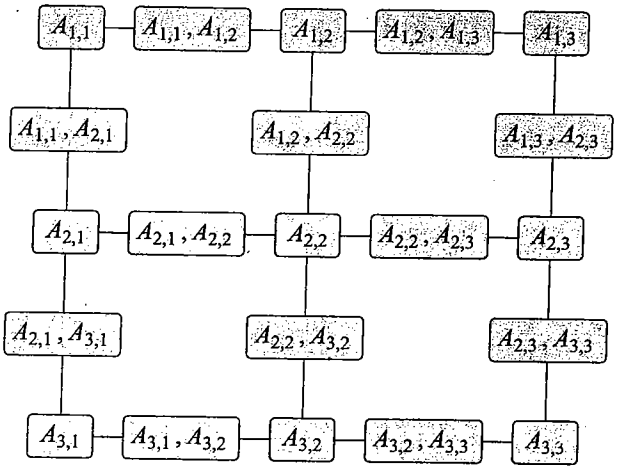


Figure 11.6 A generalized cluster graph for the 3×3 grid when viewed as pairwise MRF

ables or edges in the original Markov network, it is often convenient to think of the propagation steps as operations on the original network. Moreover, since 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. (See exercise 11.9.) Indeed, early versions of cluster-graph belief propagation were stated in these terms. This algorithm is known as *loopy belief propagation*, since it uses propagation steps used by algorithms for Markov trees, except that it was applied to networks with loops.

loopy belief propagation

11.3.5.2 Bethe Cluster Graph

A natural question is how we can extend this idea 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.

Bethe cluster graph

One simple construction, called the *Bethe cluster graph*, uses a bipartite graph. The first layer consists of "large" clusters, with one cluster for each factor ϕ in Φ, 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 11.7a.

We can easily verify that this cluster graph is a proper one. First, by construction, it satisfies the family preservation property. Second, the edges that mention a variable X form a star-shaped subgraph with edges from the univariate cluster for X to all the large clusters that contain X. It is also easy to check that, if we apply this procedure to a pairwise Markov network, it results in "natural" cluster graph for the pairwise network that we discussed. The construction of this cluster graph is simple and can easily be automated.

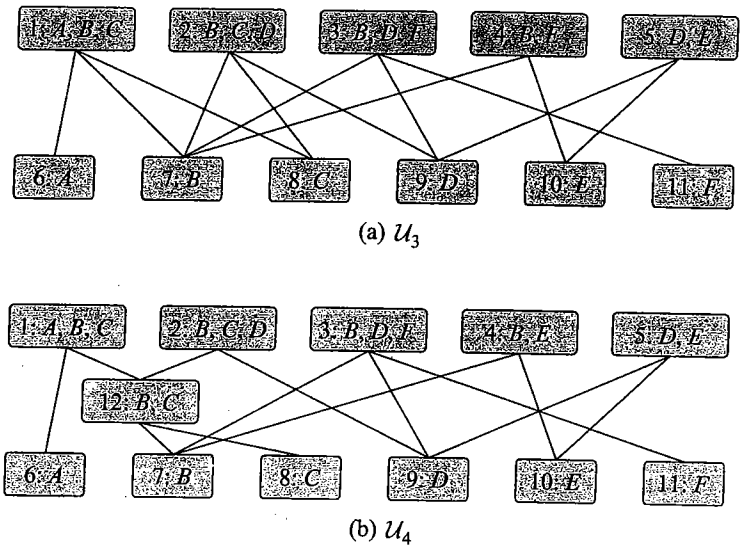


Figure 11.7 Examples of generalized cluster graphs for network with potentials over $\{A, B, C\}$, $\{B, C, D\}$, $\{B, D, F\}$, $\{B, E\}$ and $\{D, E\}$. For visual clarity, sepsets have been omitted — the sepset between any pair of clusters is the intersection of their scopes. (a) Bethe factorization. (b) Capturing interactions between $\{A, B, C\}$ and $\{B, C, D\}$.

11.3.5.3 Beyond Marginal Probabilities

The main limitation of using the Bethe cluster graph is that information between different clusters in the top level is passed through univariate marginal distributions. Thus, interactions between variables are lost during propagations. Consider the example of figure 11.7a. Suppose that $\boldsymbol{C}_1$ creates a strong dependency between B and C. These two variables are shared with $\boldsymbol{C}_2$. However, the messages between two clusters are mediated through the univariate factors. And thus, interactions introduced by one cluster are not directly propagated to the other.

One possible solution is to merge some of the large clusters. For example, if we want to capture the interactions between $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ in figure 11.7a, we can replace both of them by a cluster with the score A, B, C, D. This new cluster will allow us to capture the interactions between the factors involved in these two clusters. This modification, however, comes at a price, since the cost of manipulating a cluster grows exponentially with this scope. Moreover, this approach seems excessive in this case, since we can summarize these interactions simply using a distribution over B and C. This intuition suggests the construction of figure 11.7b. Note that this cluster graph is equivalent to figure 11.3b; see exercise 11.6. An alternative approach tries to "compensate" somehow for the violation of the running intersection property using a more complex message passing algorithm; see section 11.3.7.3.

Can we generalize this construction? A reasonable goal might be to capture all pairwise interactions. We can try to use a construction similar to the Bethe approximation, but introducing an intermediate level that includes pairwise clusters. In the same manner as we introduced $\boldsymbol{C}_{12}$ in figure 11.7b, we can introduce other pairs that are shared by more than two

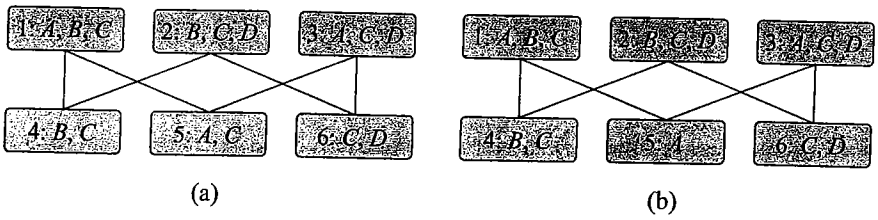


Figure 11.8 Examples of generalized cluster graph for network with potentials over $\{A, B, C\}$, $\{B, C, D\}$, and $\{A, C, D\}$. For visual clarity, sepsets have been omitted — the sepset between any pair of clusters is the intersection of their scopes. (a) A Bethe-like factorization with pairwise marginals that leads to an illegal cluster graph. (b) One possible way to make this graph legal.

clusters. As a concrete example, consider the factors $\boldsymbol{C}_1 = \{A, B, C\}$, $\boldsymbol{C}_2 = \{B, C, D\}$, and $\boldsymbol{C}_3 = \{A, C, D\}$. The relevant pairwise factors that capture interactions among these clusters are $\{B, C\} = \boldsymbol{C}_1 \cap \boldsymbol{C}_2$, $\{C, D\} = \boldsymbol{C}_2 \cap \boldsymbol{C}_3$, and $\{A, C\} = \boldsymbol{C}_1 \cap \boldsymbol{C}_3$. The resulting cluster graph appears in figure 11.8a. Unfortunately, a quick check shows that this cluster graph does *not* satisfy the running intersection property — all the edges in this graph are labeled by C, and together they form a loop. As a result, information concerning C can propagate indefinitely around the loop, "overcounting" the effect of C in the result.

How do we avoid this problem? In this specific example, we can consider a weaker approximation by removing C from one of the intersection sets. For example, if we remove C from $\boldsymbol{C}_5$, we get the cluster graph of figure 11.8b. This cluster graph satisfies the running intersection property.

belief propagation nonconvergence

Box 11.B — Skill: Making loopy belief propagation work in practice. *One of the main problems with loopy belief propagation is* nonconvergence. *This problem is particularly serious when we build systems that use inference as a subroutine within other tasks, for example, as the inner loop of a learning algorithm (see, for example, section 20.5.1). Several approaches have been used for addressing this nonconvergence issue. Some are fairly simple heuristics. Others are more sophisticated, and typically are based on the characterization of cluster-graph belief propagation as optimizing the approximate free-energy functional.*

A first observation is that, often, nonconvergence is a local problem. In many practical cases, most of the beliefs in the network do converge, and only a small portion of the network remains problematic. In such cases, it is often quite reasonable simply to stop the algorithm at some point (for example, when some predetermined amount of time has elapsed) and use the beliefs at that point, or a running average of the beliefs over some time window. This heuristic is particularly reasonable when we are not interested in individual beliefs, but rather in some aggregate over the entire network, for example, in a learning setting.

A second observation is that nonconvergence is often due to oscillations in the beliefs (see section 11.3.1). This observation suggests that we dampen the oscillations by reducing the difference between two subsequent updates. Consider the belief-propagation update rule in SP-Message(i, j)*:*

$$\delta_{i \to j} \leftarrow \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \prod_{k \neq j} \delta_{k \to i}.$$

damping

We can replace this line by a damped *version that averages the update $\delta_{i \to j}$ with the previous message between the two cliques:*

$$\delta_{i \to j} \leftarrow \lambda \left(\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \prod_{k \neq j} \delta_{k \to i} \right) + (1 - \lambda) \delta_{i \to j}^{\text{old}}, \tag{11.14}$$

where λ is the damping weight and $\delta_{i \to j}^{\text{old}}$ is the previous value of the message. When $\lambda = 1$, this update is equivalent to standard belief propagation. For $0 < \lambda < 1$, the update is partial and although it shifts β_j toward agreement with β_i, it leaves some momentum for the old value of the belief, a dampening effect that in turn reduces the fluctuations in the beliefs. It turns out that this damped update rule is "equivalent" to the original update rule, in that a set of beliefs is a convergence point of the damped update if and only if it is a convergence point of standard updates (see exercise 11.13). Moreover, one can show that, if run from a point close enough to a stable convergence points *of the algorithm, with a sufficiently small λ, this damped update rule is guaranteed to converge. Of course, this guarantee is not very useful in practice, but there are indeed many cases where the damped update rule is convergent, whereas the original update rule oscillates indefinitely.*

stable convergence points

message scheduling

A broader-spectrum heuristic, which plays an important role not only in ensuring convergence but also in speeding it up considerably, is intelligent message scheduling. *It is tempting to implement BP message passing as a synchronous algorithm, where all messages are updated at once. It turns out that, in most cases, this schedule is far from optimal, both in terms of reaching convergence, and in the number of messages required for convergence. The latter problem is easy to understand: In a cluster graph with m edges, and diameter d, synchronous message passing requires $m(d-1)$ messages to pass information from one side of the graph to the other. By contrast, asynchronous message passing, appropriately scheduled, can pass information between two clusters at opposite ends of the graph using $d-1$ messages. Moreover, the fact that, in synchronous message passing, each cluster uses messages from its neighbors that are based on their previous beliefs appears to increase the chances of oscillatory behavior and nonconvergence in general.*

asynchronous BP

tree reparameterization

In practice, an *asynchronous* message passing schedule works significantly better than the synchronous approach. Moreover, even greater improvements can be obtained by scheduling messages in a guided way. *One approach, called* tree reparameterization (TRP), *selects a set of trees, each of which spans a large number of the clusters, and whose union covers all of the edges in the network. The TRP algorithm then iteratively selects a tree and does an upward-downward calibration of the tree, keeping all other messages fixed. Of course, calibrating this tree has the effect of "uncalibrating" other trees, and so this process repeats. This approach has the advantage of passing information more globally within the graph. It therefore converges more often, and more quickly, than other asynchronous schedules, particularly if the trees are selected using a careful design that accounts for the properties of the problem.*

An even more flexible approach attempts to detect dynamically in which parts of the network messages would be most useful. Specifically, as we observed, often some parts of the network converge fairly quickly, whereas others require more messages. We can schedule messages in a way that accounts for their potential usefulness; for example, we can pass a message between clusters where the beliefs disagree most strongly on the sepset. This approach, called residual belief propagation *is convenient, since it is fully general and does not require a deep understanding of the properties of the network. It also works well across a range of different real-world networks.*

residual belief propagation

An alternative general-purpose approach to avoiding nonconvergence is to directly optimize the energy functional. Here, several methods have been proposed. The simplest is to use standard optimization methods such as gradient ascent to optimize $\tilde{F}[\tilde{P}_\Phi, Q]$ (see appendix A.5.2 and exercise 11.12). Other methods are more specialized to the form of the energy functional, and they often turn out to be more efficient (see Historical Notes). Although these methods do improve convergence, they are somewhat complex to implement, and have not (at this time) been used extensively in practice.

It turns out that many of the parameter settings encountered during a learning algorithm are problematic, and cause cluster-graph belief propagation to diverge. Intuitively, in many real-world problems, "appropriate" parameters encode strong constraints that tend to drive the algorithm toward well-behaved regions of the space. However, the parameters encountered during an iterative learning procedure have no such properties, and often allow the algorithm to end up in difficult regions. One approach is to train some parameters of the model separately, using a simpler network. We then use these parameters as our starting point in the general learning procedure. The use of "reasonable" parameters in the model can stabilize BP, allowing it to converge within the context of the general learning algorithm.

local maxima

A final problem with cluster-graph belief propagation is the fact that the energy functional objective is multimodal, and so there are many local maxima *to which a cluster-graph belief propagation algorithm might converge (if it converges). One can, of course, apply any of the standard approaches for addressing optimization of multimodal functions, such as initializing the algorithm heuristically, or using multiple restarts with different initializations. In the setting of BP, initialization must be done with care, so as not to lose the connection to the correct underlying distribution P_Φ, as reflected by the invariant of theorem 11.4. In sum-product belief propagation, we can simply initialize the messages to something other than 1. In belief update propagation, care must be taken to initialize messages and beliefs in a coordinate way, to preserve P_Φ.*

Box 11.C — Case Study: BP in practice. *To convey the behavior of belief propagation in practice, we demonstrate its performance on an 11×11 (121 binary variables) Ising grid (see box 4.C). The potentials of the network were randomly sampled as follows: Each univariate potential was sampled uniformly in the $[0, 1]$; for each pair of variables X_i, Z_j, $w_{i,j}$ is sampled uniformly in the range $[-C, C]$ (recall that in an Ising model, we define the negative log potential $\epsilon_{i,j}(x_i, x_j) = -w_{i,j}x_ix_j$). This sampling process creates an energy function where some potentials are attractive ($w_{i,j} > 0$) and some are repulsive ($w_{i,j} < 0$), resulting in a nontrivial inference problem. The magnitude of C (11 in this example) controls the magnitude of the energy forces and higher values correspond, on average, to more challenging inference problems.*

Figure 11.C.1 illustrates the convergence behavior on this problem. (a) shows the percentage of messages converged as a function of time for three variants of the belief propagation algorithm: synchronous BP with smoothing (dashed line), where only a small fraction of the messages ever converge; asynchronous BP with smoothing that converges (solid line); asynchronous BP with no smoothing (dash-dot line) that does not fully converge. The benefit of using asynchronous propagation over synchronous updating is obvious. At first, it appears is if smoothing messages is not beneficial. This is because some percentage of messages can converge quickly when updates are not

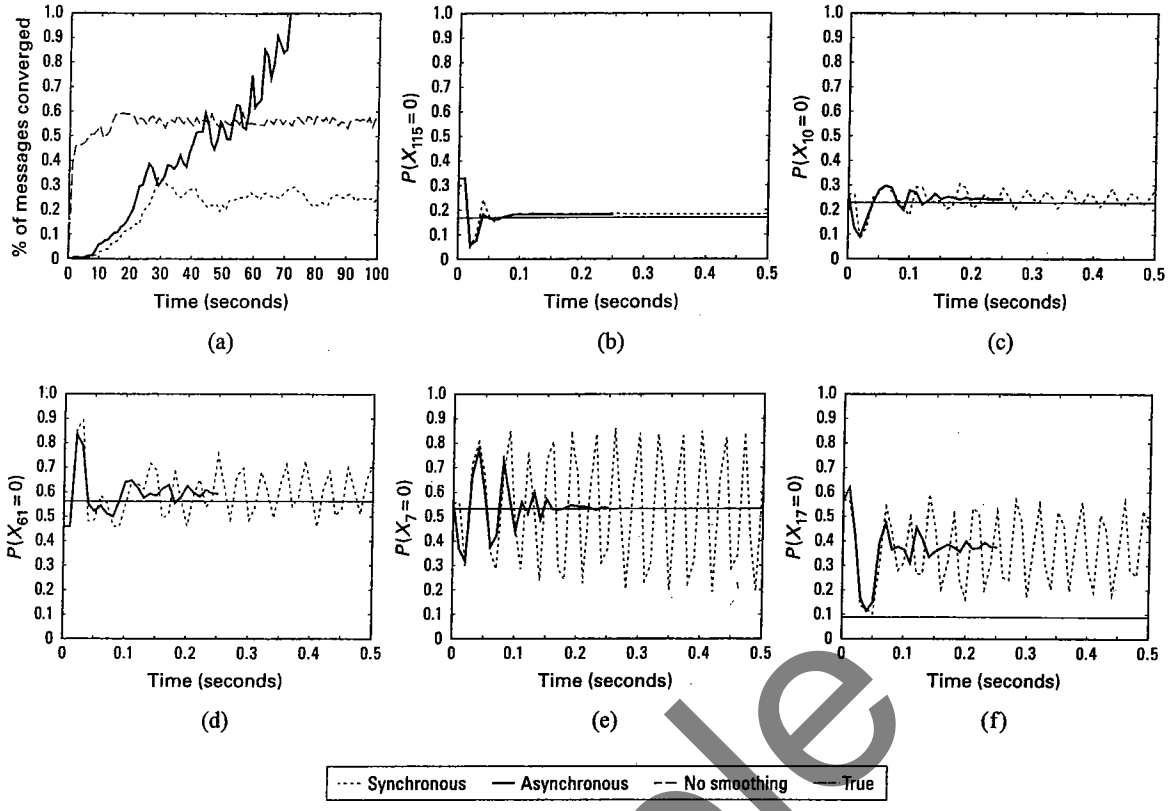


Figure 11.C.1 — Example of behavior of BP in practice on an 11 × 11 Ising grid. (a) Percentage of messages converged as a function of time for three different BP variants. (b) A marginal where both variants converge rapidly. (c–e) Marginals where the synchronous BP marginals oscillate around the asynchronous BP marginals. (f) A marginal where both variants are inaccurate.

slowed down by smoothing. However, the overall benefit of smoothing is evident, and without it the algorithm never converges.

The remaining panels illustrate the progression of the marginal beliefs over the course of the algorithm. (b) shows a marginal where both the synchronous and asynchronous updates converge quite rapidly and are close to the true marginal (thin solid black). Such behavior is atypical, and it comprises only around 10 percent of the marginals in this example. In the vast majority of the cases (almost 80 percent in this example), the synchronous beliefs oscillate around the asynchronous ones ((c)–(e)). In many cases, such as the ones shown in (e), the entropy of the synchronous beliefs is quite significant. For about 10 percent of the marginals (for example (f)), both the asynchronous and synchronous marginals are inaccurate. In these cases, using more informed message schedules can significantly improve the algorithms performance.

These qualitative differences between the BP variants are quite consistent across many random

and real-life models. Typically, the more complex the inference problem, the larger the gaps in performance. For very complex real-life networks involving tens of thousands of variables and multiple cycles, even asynchronous BP is not very useful and more elaborate propagation methods or convergent alternatives must *be adopted.*

11.3.6 Variational Analysis

So far, our discussion of cluster-graph belief propagation has been procedural, 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 cluster-graph belief propagation can be justified using the energy functional formulation of section 11.1. Specifically, the messages passed by cluster-graph belief propagation can be derived from fixed-point equations for the stationary points of an approximate version of the energy functional of equation (11.3). As we will see, this formulation provides significant insight into the generalized belief propagation algorithm. It allows us to understand better the convergence properties of cluster-graph belief propagation and to characterize its convergence points. It also suggests generalizations of the algorithm that have better convergence properties, or that optimize a better approximation to the energy functional.

Our construction will be similar to the one in section 11.2 for exact inference. However, there are important differences that underlie the fact that this algorithm is only an approximate inference algorithm.

factored energy functional

First, the exact energy functional $F[\tilde{P}_\Phi, Q]$ has terms involving the entropy of an entire joint distribution; thus, it cannot be tractably optimized. However, the *factored energy functional* $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$ is defined in terms of entropies of clusters and sepsets, each of which can be computed efficiently based purely on local information at the clusters. Importantly, however, unlike for clique trees, $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$ is no longer simply a reformulation of the energy functional, but rather an approximation of it.

However, even the factored energy functional cannot be optimized over the space of all marginals $\boldsymbol{Q}$ that correspond to some actual distribution P_Φ. More precisely, consider some cluster graph $\mathcal{U}$; for a distribution P, we define $\boldsymbol{Q}_P = \{P(\boldsymbol{C}_i)\}_{i \in \mathcal{V}_\mathcal{U}} \cup \{P(\boldsymbol{S}_{i,j})\}_{(i-j) \in \mathcal{E}_\mathcal{U}}$. We now define the *marginal polytope* of $\mathcal{U}$ to be

marginal polytope

$$Marg[\mathcal{U}] = \{\boldsymbol{Q}_P : P \text{ is a distribution over } \mathcal{X}\} \tag{11.15}$$

That is, the marginal polytope is the set of all cluster (and sepset) beliefs that can be obtained from marginalizing an actual distribution P. It is called the marginal polytope because it is the set of marginals obtained from the polytope of all probability distributions over $\mathcal{X}$. Unfortunately, not every set of beliefs that correspond to clusters in $\mathcal{U}$ is in the marginal polytope; that is, there are calibrated cluster graph beliefs that do not represent the marginals of any single coherent joint distribution over $\mathcal{X}$ (see exercise 11.2). However, the marginal polytope is a complex object with exponentially many facets. (In fact, the problem of determining whether a set of beliefs is in the marginal polytope can be shown to be $\mathcal{NP}$-hard.) Thus, optimizing a function over the

marginal polytope is a computationally difficult task that is generally as hard as exact inference over the cluster graph. To circumvent these problems, we instead we perform our optimization over the *local consistency polytope*:

local consistency polytope

$$Local[\mathcal{U}] = \left\{ \begin{array}{l|lll} \{\beta_i : i \in \mathcal{V}_\mathcal{U}\} \cup & \mu_{i,j}[\boldsymbol{s}_{i,j}] & = & \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i(\boldsymbol{c}_i) \quad \forall (i\text{-}j) \in \mathcal{E}_\mathcal{U}, \forall \boldsymbol{s}_{i,j} \in Val(\boldsymbol{S}_{i,j}) \\ \{\mu_{i,j} : (i\text{-}j) \in \mathcal{E}_\mathcal{U}\} & 1 & = & \sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) \quad \forall i \in \mathcal{V}_\mathcal{U} \\ & \beta_i(\boldsymbol{c}_i) & \geq & 0 \quad \forall i \in \mathcal{V}_\mathcal{U}, \boldsymbol{c}_i \in Val(\boldsymbol{C}_i). \end{array} \right\} \tag{11.16}$$

pseudo-marginals

We can think of the local consistency polytope as defining a set of *pseudo-marginal distributions*, each one over the variables in one cluster. The constraints imply that these pseudo-marginals must be calibrated and therefore locally consistent with each other. However, they are not necessarily marginals of a single underlying joint distribution.

Overall, we can write down an optimization problem as follows:

CGraph-Optimize:

Find $\boldsymbol{Q}$
maximizing $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$
subject to

$$\boldsymbol{Q} \in Local[\mathcal{U}] \tag{11.17}$$

☞ **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.**

In section 11.1, we noted that the energy functional is a lower bound on the log-partition function; thus, by maximizing it, we get better approximations of P_Φ. 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, since it may lead to a good approximation of the log-partition function.

This maximization problem directly generalizes CTree-Optimize to the case of cluster graphs. Not surprisingly, we can derive a similar analogue to theorem 11.3, where we characterize the stationary points of this optimization problem as solutions to a set of *fixed-point equations*.

fixed-point equations

Theorem 11.5

A set of beliefs $\boldsymbol{Q}$ is a stationary point of CGraph-Optimize *if and only if for every edge $(i\text{-}j) \in \mathcal{E}_\mathcal{U}$ there are auxiliary factors $\delta_{i \to j}(\boldsymbol{S}_{i,j})$ and $\delta_{j \to i}(\boldsymbol{S}_{j,i})$ so that*

$$\delta_{i \to j} \propto \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \cdot \prod_{k \in \mathrm{Nb}_i - \{j\}} \delta_{k \to i}. \tag{11.18}$$

and moreover, we have that

$$\begin{aligned} \beta_i &\propto \psi_i \cdot \prod_{j \in \mathrm{Nb}_i} \delta_{j \to i} \\ \mu_{i,j} &= \delta_{j \to i} \cdot \delta_{i \to j}. \end{aligned}$$

The proof is identical to the proof of theorem 11.3.

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 equation (11.18) to redefine each $\delta_{i \to j}$ in terms of the current values of other $\delta_{k \to i}$. This process is identical (up to a renormalization step) to the update formula we use in CTree-SP-calibrate (algorithm 10.2). Indeed, if we define CGraph-SP-Calibrate, a cluster graph version of CTree-SP-Calibrate, the message passing steps would be simply executing this iterative process using the fixed-point equation. Theorem 11.5 shows that convergence points of this procedure are related to stationary points of $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$.

Corollary 11.1

$\boldsymbol{Q}$ is the convergence point of applying CGraph-SP-Calibrate$(\Phi, \mathcal{U})$ *if and only if $\boldsymbol{Q}$ is a stationary point of $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$.*

Due to the equivalence between sum-product and belief update messages, it follows that convergence points of CGraph-BU-Calibrate are also convergence points of CGraph-SP-Calibrate.

Corollary 11.2

At convergence of CGraph-BU-Calibrate, *the set of beliefs is a stationary point of $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$.*

It is tempting to interpret this result as stating that the convergence points of belief propagation are maxima of the factored energy functional. However, there are several gaps between the theorem and this idealized interpretation, which it is important to understand. First, we note that maxima of a function are not necessarily fixed points. In this case, we can verify that $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$ is bounded from above, and thus must have a maximum. However, if the maximum is a boundary point (where some of the probabilities in $\boldsymbol{Q}$ are 0), it may not be a fixed point. Fortunately, this situation is rare in practice, and it can be guaranteed not to arise under fairly benign assumptions.

Second, we note that maxima are not the only fixed points of the belief propagation algorithm; minima and saddle points are also fixed points. Intuitively, however, such solutions are not likely to be stable, in the sense that slight perturbations to the messages will drive the process away from them. Indeed, it is possible to show (although this result is outside the scope of this book) that *stable convergence points* of belief propagation are always local maxima of the function.

stable convergence points

The most important limitation of this result, however, is that it does not show that we can reach these maxima by applying belief propagation steps. There is no guarantee that the message passing steps of cluster-graph belief propagation necessarily improve the energy functional: a message passing step may increase or decrease the energy functional. Indeed, as we showed, there are examples where the belief propagation procedure oscillates indefinitely and fails to converge. Even more surprisingly, this problem is not simply a matter of the algorithm being unable to "find" the maximum. One can show examples where the global maximum is not a *stable* convergence point of belief propagation. That is, while it is, in principle, a fixed point of the algorithm, it will never be reached in practice, since even a slight perturbation will give rise to oscillatory behavior.

Nevertheless, this result is of significant importance in several ways. First, it provides us with a declarative semantics for cluster-graph belief propagation in terms of optimization of a target functional. The success of the belief propagation algorithm, when it converges, leads us to hope that the development of new, possibly more convergent, methods to solve the optimization

problem may give rise to good solutions. Second, the declarative view defines the problem in terms of an objective — the factored energy functional — and a set of constraints — the set of locally consistent pseudo-marginals. Both of these are approximations to the ones used in the optimization problem for exact inference. When we view the task from this perspective, some potential directions for improvements become obvious: We can perhaps achieve a better approximation by making our objective a better approximation to the true energy functional, or by tightening our constraints so as to make the constraint space closer to the exact marginal polytope. We will describe some of the extensions based on these ideas; others are mentioned in section 11.7.

11.3.7 Other Entropy Approximations ⋆

The variational analysis of the previous section provides us with a framework for understanding the properties of this type of approximation, and for providing significant generalizations.

11.3.7.1 Motivation

To understand this general framework, consider first the form of the factored energy functional when our cluster graph $\mathcal{U}$ 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 Φ, and the other consisting of univariate clusters. When the cluster graph is calibrated, these univariate clusters have the same distribution as the sepsets between them and the factors in the first layer. As such, we can combine together the entropy terms for all the sepsets labeled by X and the associated univariate cluster and rewrite the energy functional, as follows:

Proposition 11.3 *If $\boldsymbol{Q} = \{\beta_\phi : \phi \in \Phi\} \cup \{\beta_i(X_i)\}$ is a calibrated set of beliefs for a Bethe cluster graph $\mathcal{U}$ with clusters $\{\boldsymbol{C}_\phi : \phi \in \Phi\} \cup \{X_i : X_i \in \mathcal{X}\}$, then*

$$\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] = \sum_{\phi \in \Phi} \boldsymbol{E}_{Scope[\phi] \sim \beta_\phi}[\ln \phi] + \sum_{\phi \in \Phi} \boldsymbol{H}_{\beta_\phi}(\boldsymbol{C}_\phi) - \sum_i (d_i - 1)\boldsymbol{H}_{\beta_i}(X_i), \tag{11.19}$$

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

Note that equation (11.19) is equivalent to the factored energy functional only when $\boldsymbol{Q}$ is calibrated. However, because 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 energy functional. Equation (11.19) is the negation of a term known as the *Bethe free energy* in statistical mechanics. The Bethe cluster graph we discussed earlier is a construction that is designed to match the Bethe free energy functional.

Bethe free energy

Why is this reformulation useful? Recall that, in our discussion of generalized cluster graphs, we required the running intersection property. This property has two important implications. First is 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 is that there is no cycle of clusters and sepsets all of which contain X. We motivated this assumption by noting that it prevents us from allowing information about X to cycle endlessly through a loop. This new formulation provides a more formal justification. As we can see, 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. Owing to the running intersection property, the number of sepsets 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, the entropy of X_i appears with positive sign d_i times, and with negative sign $d_i - 1$ times, so overall it is counted exactly once.

counting number

This reformulation suggests a much more general class of entropy approximations. We can construct define a set of *regions* $\mathbf{R}$, each with its own scope $\boldsymbol{C}_r$ and its own *counting number* κ_r. We can now define the following *weighted approximate entropy*:

weighted approximate entropy

$$\tilde{\boldsymbol{H}}_{\boldsymbol{Q}}^{\kappa}(\mathcal{X}) = \sum_r \kappa_r \boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r). \tag{11.20}$$

Example 11.2

Bethe cluster graph

The simple Bethe cluster graph *of section 11.3.5.2 fits easily into this new framework. The construction has two levels of regions: a set of "large" $\mathbf{R}^+$, where each $r \in \mathbf{R}^+$ contains multiple variables, and singleton regions containing the individual variables $X_i \in \mathcal{X}$. Both types of regions have counting numbers: κ_r for $r \in \mathbf{R}^+$ and κ_i for $X_i \in \mathcal{X}$. All factors in Φ are assigned only to large regions, so that $\psi_i = 1$ for all i. We use Nb_r to denote the set $\{X_i \in \boldsymbol{C}_r\}$, and Nb_i to denote the set $\{r : X_i \in \boldsymbol{C}_r\}$.*

To capture exactly the Bethe free energy, we set each large region to have a counting number of 1, and each singleton region corresponding to X_i to have a counting number of $1 - d_i$ where d_i is the number of large regions that contain X_i in their scope. We see that in this construction the region graph energy functional is identical to the Bethe free energy of equation (11.19). ■

However, this framework also allows us to capture much richer constructions.

Example 11.3

Consider again the example of figure 11.8a. As we discussed in section 11.3.5.3, this cluster graph has the benefit of maintaining the pairwise correlations between all pairs of variables when passing messages between clusters. Unfortunately, it is not a legal cluster graph, since it does not satisfy the running intersection property. We can obtain another perspective on the problem with this cluster graph by examining the energy functional associated with it:

$$\begin{aligned}\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] &= \boldsymbol{E}_{\beta_1}[\ln \phi_1(A,B,C)] + \boldsymbol{E}_{\beta_2}[\ln \phi_2(B,C,D)] + \boldsymbol{E}_{\beta_3}[\ln \phi_3(A,C,D)] \\ &\quad + \boldsymbol{H}_{\beta_1}(A,B,C) + \boldsymbol{H}_{\beta_2}(B,C,D) + \boldsymbol{H}_{\beta_3}(A,C,D) \\ &\quad - \boldsymbol{H}_{\beta_4}(B,C) - \boldsymbol{H}_{\beta_5}(A,C) - \boldsymbol{H}_{\beta_6}(C,D).\end{aligned}$$

As we can see, the variable C appears in three clusters and three sepsets. As a consequence, the counting number of C in the energy functional is 0. This means that we are undercounting the entropy of C in the approximation. Indeed, as we discussed, this cluster graph does not satisfy the running intersection property. Thus, we considered modifying the graph by removing C from one of the sepsets. However, if we consider this problem from the perspective of the energy functional, we can deal with the problem by adding another factor β_7 that has C as its scope. If we add $\boldsymbol{H}_{\beta_7}(C)$ to the energy functional we solve the undercounting problem. This results in a modified energy functional

$$\begin{aligned}\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] &= \boldsymbol{E}_{\beta_1}[\ln \phi_1(A,B,C)] + \boldsymbol{E}_{\beta_2}[\ln \phi_2(B,C,D)] + \boldsymbol{E}_{\beta_3}[\ln \phi_3(A,C,D)] \\ &\quad + \boldsymbol{H}_{\beta_1}(A,B,C) + \boldsymbol{H}_{\beta_2}(B,C,D) + \boldsymbol{H}_{\beta_3}(A,C,D) + \boldsymbol{H}_{\beta_7}(C) \\ &\quad - \boldsymbol{H}_{\beta_4}(B,C) - \boldsymbol{H}_{\beta_5}(A,C) - \boldsymbol{H}_{\beta_6}(C,D).\end{aligned}$$

This is simply an instance of our weighted entropy approximation, with seven regions: the three triplets, the three pairs, and the singleton C. ■

This perspective provides a clean and simple framework for proposing generalizations to the class of approximations defined by the cluster graph framework. Of course, to formulate our optimization problem fully, we need to define the constraints and construct algorithms that solve the resulting optimization problems. We now address these issues in the context of two different classes of weighted entropy approximations.

11.3.7.2 Convex Approximations

One of the biggest problems with the objective used in standard loopy BP is that it gives rise to a nonconvex optimization problem. In fact, the objective often has multiple local optima. These properties make the optimization hard and the answers nonrobust. However, a different choice of counting numbers can lead to a concave optimization objective, and hence to a convex optimization problem. Such problems are much easier to solve using a range of algorithms, and the solutions offer a satisfying guarantee of optimality. We first define the class of convex BP objectives and then describe one solution algorithm.

We focus our discussion on energy functionals whose structure uses the two-layer Bethe cluster graph structure of example 11.2, but where the counting numbers are different. To preserve the desired semantics of the counting numbers, we require:

$$\kappa_i = 1 - \sum_{r \in \mathrm{Nb}_i} \kappa_r, \tag{11.21}$$

ensuring that the total counting number of terms involving the entropy of X_i is precisely 1. When we define $\kappa_r = 1$ for all $r \in \mathbf{R}$, this constraint implies the counting numbers in the Bethe free energy.

We now introduce the following condition on the counting numbers:

Definition 11.4 convex counting numbers

We say that a vector of counting numbers κ_r is convex *if there exist* nonnegative *numbers ν_r, ν_i, and $\nu_{r,i}$ such that:*

$$\begin{array}{rcll} \kappa_r & = & \nu_r + \sum_{i \,:\, X_i \in \boldsymbol{C}_r} \nu_{r,i} & \text{for all } r \\ \kappa_i & = & \nu_i - \sum_{r \,:\, X_i \in \boldsymbol{C}_r} \nu_{r,i} & \text{for all } i \end{array} \tag{11.22}$$

■

Assuming that we have a set of convex counting numbers, we can rewrite the weighted approximate entropy of equation (11.20) as:

$$\begin{aligned} &\sum_r \kappa_r \boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r) + \sum_i \kappa_i \boldsymbol{H}_{\beta_i}(X_i) = \\ &\quad \sum_r \nu_r \boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r) + \sum_{r, X_i \in \boldsymbol{C}_r} \nu_{r,i} (\boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r) - \boldsymbol{H}_{\beta_i}(X_i)) + \sum_i \nu_i \boldsymbol{H}_{\beta_i}(X_i). \end{aligned} \tag{11.23}$$

Importantly, when the beliefs satisfy the marginal-consistency constraints, the terms in the second summation can be rewritten as conditional entropies:

$$\boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r) - \boldsymbol{H}_{\beta_i}(X_i) = \boldsymbol{H}_{\beta_r}(\boldsymbol{C}_r \mid X_i).$$

Plugging this result back into equation (11.23), we obtain an objective that is a summation of terms each of which is either an entropy or a conditional entropy, all with positive coefficients. Because both entropies and conditional entropies are convex, we obtain the following result:

Proposition 11.4 *The function in equation (11.23) is a concave function for any set of beliefs Q that satisfies the marginal consistency constraints.*

concave over constraints

convex entropy

This type of objective function is called *concave over the constraints*, since it is not generally concave, but it is concave over the subspace that satisfies the constraints of our optimization problem. An entropy as in equation (11.23) that uses convex counting numbers is called a *convex entropy*.

Assuming that the potentials are all strictly positive, we can now conclude that the optimization problem RegionGraph-Optimize with convex counting numbers is a convex optimization problem that has a unique global optimum.

Convex optimization problems can, in principle, be solved by a range of different algorithms, all of which guarantee convergence to the unique global optimum. However, the basic optimization problem can easily get intractably large. Recall that to formulate our optimization space, we need to introduce an optimization variable for each assignment of values to each cluster in our cluster graph, and a constraint for each assignment of values to each sepset in the graph.

Example 11.4 *Consider a grid-structured network corresponding to a modestly sized 500×500 image, where each pixel can take 100 values. The structure of the graph is a pairwise network, with approximately $2 \times 250,000$ clusters (pairwise edges), each of which can take $100 \times 100 = 10,000$ values. The total number of variables is therefore $500,000 \times 10,000 = 5 \times 10^9$, an unmanageable number for most optimizers.* ■

Fortunately, due to the convexity of this problem, we have that strong duality holds (see appendix A.5.4), and therefore we can find a solution to this problem by solving its dual. The message passing algorithms that we derive from the Lagrange multipliers are one method that we can use for solving the dual. (For example, exercise 11.17 provides one message passing algorithm for a Bethe cluster graph with general counting numbers.) However, the message passing algorithms are not directly optimizing the objective. Rather, they characterize the optimum using a set of fixed-point equations, and attempt to converge to the optimum by iterating through these equations. This process is generally not guaranteed to achieve the optimum, even when the problem is convex. Again, we can consider using other optimization algorithms over the dual problem. However, a message passing approach has some important advantages, such as modularity and efficiency.

Fortunately, a careful reformulation of the message passing scheme can be shown to guarantee convergence to the global optimum. This reformulation is different for synchronous and asynchronous message passing. We present the asynchronous version, which is simpler and also likely to be more efficient in practice.

The algorithm, shown in algorithm 11.2, uses the following quantities in its computations:

$$\hat{\nu}_i = \nu_i + \sum_{r \in \mathrm{Nb}_i} \nu_r; \qquad \hat{\nu}_{i,r} = \nu_r + \nu_{i,r}. \tag{11.24}$$

In each message passing iteration, it traverses the variables X_i in a round-robin fashion; for each X_i, it computes two sets of messages: incoming messages $\delta_{r \to i}(X_i)$ from regions to variables,

Algorithm 11.2 Convergent message passing for Bethe cluster graph with convex counting numbers

Procedure Convex-BP-Msg (
  $\psi_r(\boldsymbol{C}_r)$ // set of initial potentials
  $\sigma_{i \to r}(\boldsymbol{C}_r)$ // Current node-to-region messages
)
1 **for** $i = 1, \ldots, n$
2 // Compute incoming messages from neighboring regions to X_i
3 **for** $r \in \mathrm{Nb}_i$
4 $\delta_{r \to i}(X_i) \leftarrow \sum_{\boldsymbol{C}_r - X_i} \left(\psi_r(\boldsymbol{C}_r) \prod_{j \in \mathrm{Nb}_r - \{i\}} \sigma_{j \to r}(\boldsymbol{C}_r) \right)^{\frac{1}{\nu_{i,r}}}$
5 // Compute beliefs for X_i, renormalizing to avoid numerical underflows
6 $\beta_i(X_i) \leftarrow \frac{1}{Z_{X_i}} \prod_{r \in \mathrm{Nb}_i} (\delta_{r \to i}(X_i))^{\nu_{i,r}/\nu_i}$
7 // Compute outgoing messages from X_i to neighboring regions
8 **for** $r \in \mathrm{Nb}_i$
9 $\sigma_{i \to r}(\boldsymbol{C}_r) \leftarrow \left(\psi_r(\boldsymbol{C}_r) \prod_{j \in \mathrm{Nb}_r - \{i\}} \sigma_{j \to r}(\boldsymbol{C}_r) \right)^{-\frac{\nu_{i,r}}{\nu_{i,r}}} \left(\frac{\beta_i(X_i)}{\delta_{r \to i}(X_i)} \right)^{\nu_r}$
10 **return** $\{\sigma_{i \to r}(\boldsymbol{C}_r)\}_{i, r \in \mathrm{Nb}_i}$

factor graph

and outgoing messages $\sigma_{i \to r}(\boldsymbol{C}_r)$ from variables to regions (essentially passing messages over the *factor graph*). The overall process is initialized (in the first message passing iteration) by setting $\sigma_{i \to r} = 1$. This algorithm is guaranteed to converge to the global maximum of our convex energy functional.

This derivation applies to any set of convex counting numbers, leaving open the question of which counting numbers are likely to be give the best approximation. Although there is currently no theoretical analysis answering this question, intuitively, we might argue that we want the counting numbers for different regions to be as close as possible to uniform. This intuition is also supported by the fact that the Bethe approximation, which sets all $\kappa_r = 1$, obtains very high-quality approximations when it converges. Thus, we can try to select nonnegative coefficients ν_i, ν_r, and $\nu_{i,r}$ for which κ_r and κ_i, defined via equation (11.22), satisfy equation (11.21) and minimize

$$\sum_{r \in \mathbf{R}^+} (\kappa_r - 1)^2. \tag{11.25}$$

TRW

Other choices are also possible. For example, the *tree-reweighted belief propagation (TRW)* algorithm computes convex counting numbers for a pairwise Markov network using the following process: We first define a probability distribution ρ over trees $\mathcal{T}$ in the network, such that each edge in the pairwise network is present in at least one tree. This distribution defines a set of weights:

$$\begin{aligned} \kappa_i &= -\sum_{\mathcal{T} \ni X_i} \rho(\mathcal{T}) \\ \kappa_{i,j} &= \sum_{\mathcal{T} \ni (X_i, X_j)} \rho(\mathcal{T}) \end{aligned} \tag{11.26}$$

convex counting numbers

This computation results in a set of *convex counting numbers* (see exercise 11.18). Preliminary results suggest that the TRW counting numbers and the ones derived from optimizing equation (11.25) appear to achieve similar performance in practice.

However, the comparison to standard (Bethe-approximation) BP is less clear. **When standard BP converges, it generally tends to produce better results than the convex counterparts, and almost universally it converges much faster. Conversely, when standard BP does not converge, the convex algorithms have an advantage;** but, as we discuss in box 11.B, there are many tricks we can use to improve the convergence of BP, so it is not clear how often nonconvergence is a problem. One setting where a convergent algorithm can have important benefits is in those settings (chapter 19 and chapter 20) where we generally learn the model using iterative, hill-climbing methods that use inference in the inner loop for tasks such as gradient computations. There, the use of a nonconvergent algorithm for computing the gradient can severely destabilize the learning algorithm. In other settings, however, the decision of whether to use standard or convex BP is one of approximate optimization of a pretty good (although still approximate) objective, versus exact optimization of an objective that is generally not as good. The right decision in this trade-off is not clear, and needs to be made specifically for the target application.

11.3.7.3 Region Graph Approximations

As illustrated in example 11.3, a very different motivation for using an objective based on different counting numbers is to improve the quality of the approximation by better capturing interactions between variables. As we showed in this example, we can use the notion of a weighted entropy approximation to define a (hopefully) better approximation to the entropy. Of course, to specify the optimization problem fully, we also need to specify the constraints. In this example, it is fairly straightforward to do so: we want $\beta_7(C)$ to be consistent with the marginal probability of C in one of the other beliefs that mention C. Now, we have an optimization problem that seems to solve the problem we set out to solve: It can compute beliefs on each of the original factors while maintaining consistency at the level of each pairwise marginal shared among these factors.

However, the new optimization problem we defined is not one that corresponds to a cluster graph. To see this, notice that β_7 appears in the role of a cluster. But, if it is a cluster, it would have to be connected to one of the other factors by a sepset with scope C, which would require an additional term in the energy functional associated with this cluster graph. Thus, it is not immediately clear how we would go about optimizing the new modified functional.

We now discuss a general framework that defines the form of the optimization objective and the constraints for constructions that capture higher-level interactions between the variables. We also describe a message passing algorithm that can be used to find fixed points of this optimization problem.

Region Graphs The basic structure we consider is similar to a cluster graph, but unlike cluster graphs we no longer distinguish two types of vertices (clusters and sepsets). Rather, we can have a more deeply nested hierarchy of regions, related by containment.

Definition 11.5 region graph

A region graph $\mathcal{R}$ *is a directed graph over a set of vertices* $\mathbf{R}$. *Each vertex* r *is called a* region *and*

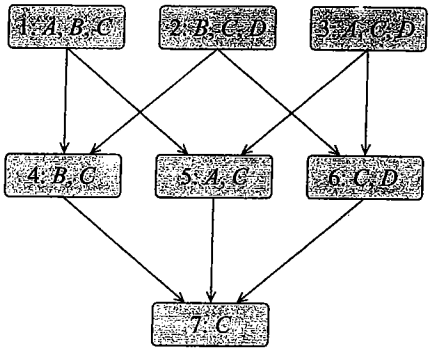


Figure 11.9 An example of simple region graph

is associated with a distinct set of random variables $\boldsymbol{C}_r$. *Whenever there is an arc from a region* r_i *to a region* r_j *we require that* $Scope[r_i] \supset Scope[r_j]$. *Regions that have no incoming edges are called* top regions.

counting numbers

Each region is associated with a counting number $\kappa_r \in \mathbb{R}$. *Note that the counting number may be negative, positive, or zero.* ■

Because containment is transitive, we have that if there is a directed path from r to r', then $Scope[r'] \subset Scope[r]$. Thus, a region graph is acyclic.

To define the energy term in the free energy, we must assign our original factors in Φ to regions in the region graph. Here, because different regions are counted to different extents, it is useful to allow a factor to be assigned to more than one region. Thus, for each $\phi \in \Phi$ we have a set of regions $\alpha(\phi) \subset \mathbf{R}$ such that $Scope[\phi] \subseteq \boldsymbol{C}_r$. This property is analogous to the *family-preservation property* for cluster graphs. Throughout this book, we assume without loss of generality that any $r \in \alpha(\phi)$ is a top region.

family preservation

We are now ready to define the energy functional associated with a region graph:

$$\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}] = \sum_r \kappa_r \boldsymbol{E}_{\boldsymbol{C}_r \sim \beta_r}[\ln \psi_r] + \tilde{\boldsymbol{H}}_{\boldsymbol{Q}}^{\boldsymbol{\kappa}}(\mathcal{X}), \tag{11.27}$$

where ψ_r is defined as:

$$\psi_r = \prod_{\phi \in \Phi \,:\, r \in \alpha(\phi)} \phi.$$

As with cluster graphs, a region graph defines a set of beliefs, one per region. We use the notation $\beta_r(\boldsymbol{C}_r)$ to denote the belief associated with the region r over the set of variables $\boldsymbol{C}_r = Scope[r]$.

Figure 11.9 demonstrates the region graph construction for the approximation of example 11.3. This example contains three regions that correspond to the initial factors in the distribution we want to approximate. The lower set of regions are the pairwise intersections between the three factors. The lowest region is associated with the variable C.

Whereas the counting numbers specify the energy functional, the graph structure specifies the constraints over the beliefs that we wish to associate with the regions. In particular, we

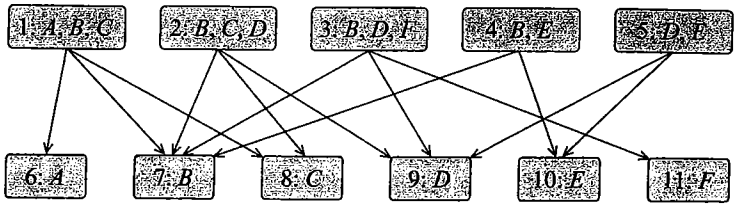


Figure 11.10 The region graph corresponding to the Bethe cluster graph of figure 11.7a

want the beliefs to satisfy the calibration constraints that are implied by the edges in the region graph structure.

Definition 11.6 region graph calibration

Given a region graph, a set of region beliefs is calibrated *if whenever $r \rightarrow r'$ appears in the region graph then*

$$\sum_{\boldsymbol{C}_r - \boldsymbol{C}_{r'}} \beta_r(\boldsymbol{C}_r) = \beta_{r'}(\boldsymbol{C}'_r) \tag{11.28}$$

■

The region graph structure provides a very general framework for specifying energy-functional optimization problems. The set of regions and their counting numbers tell us which components appear in the energy functional, and with what weight. The arcs in the region graph tell us which consistency constraints we wish to enforce. We can choose which consistency constraints we want to enforce by adding or removing regions or edges between them; we note that this can be done without affecting the energy functional by simply giving the regions introduced a counting number of 0. The more edges we have, the more constraints we require regarding the calibration of different beliefs.

The region graph framework is very general, and it can encode a broad spectrum of optimization objectives and constraints. However, not all such formulations are equally reasonable as approximations to our true objective, which is the exact energy functional of equation (11.3). The following requirement captures some of the essential properties that make a region graph construction suitable for this purpose.

Definition 11.7

For each variable X_i, let $\mathbf{R_i} = \{\mathbf{r} : \mathbf{X_i} \in Scope[\mathbf{r}]\}$; for each factor $\phi \in \Phi$, let $\mathbf{R}_\phi = \{\mathbf{r} : Scope[\phi] \subseteq \boldsymbol{C}_\mathbf{r}\}$. We now define the following conditions for a region graph:

- variable connectedness*: for every variable X_i, the set $\mathbf{R_i}$ forms a single connected component;*
- factor connectedness*: for every factor ϕ, the set $\mathbf{R}_\phi$ forms a single connected component;*
- factor preservation*:*

$$\sum_{r \in \alpha(\phi)} \kappa_r = 1.$$

- running intersection*: for every variable X_i,*

$$\sum_{r \in \mathbf{R_i}} \kappa_r = 1.$$

■

The connectedness requirement for variables ensures that the beliefs about any individual variable X_i in any calibrated region graph will be consistent for all beliefs that contain X_i. The counting number condition for variables ensures that we are not not overcounting or undercounting the entropy of X_i. Together, these conditions extend the running intersection property, ensuring that the subgraph containing X_i is connected and that X_i is "counted" once in total. We note that, like the running intersection property, this requirement does not address the extent to which we count the contribution associated with the interactions between pairs or larger subsets of variables.

The factor-preservation condition for factors ensures that, when we sum up the energy terms of the different regions, each factor is counted exactly once in total. As we will see, this ensures that a calibrated region graph will still encode our original distribution P_Φ. Finally, the connectedness condition for factors ensures that we cannot double-count contributions of our initial factors: that is, a factor cannot "flow" around a loop.

An examination confirms that all parts of the region graph condition hold both for the Bethe region graph and for the region graph of figure 11.9.

How do we construct a valid region graph? One approach is based on the following simple recursive construction for the counting numbers. We first define, for a region r,

$$\mathbf{Up}(r) = \{r' : (r' \to r) \in \mathcal{E}_\mathcal{R}\}$$

to be the set of regions that are directly upwards of r; similarly, we define

$$\mathbf{Down}(r) = \{r' : (r \to r') \in \mathcal{E}_\mathcal{R}\}.$$

We also define the *upward closure* of r to be the set $\mathbf{Up}^*(r)$ of all the regions from which there is a directed path to r, and the *downward closure* $\mathbf{Down}^*(r)$ to be all the regions that can be reached by a directed path from r; finally, we define $\mathbf{Down}^+(r) = \{r\} \cup \mathbf{Down}^*(r)$.[2]

We can now define the counting numbers recursively, using the following formula:

$$\kappa_r = 1 - \sum_{r' \in \mathbf{Up}^*(r)} \kappa_{r'}. \tag{11.29}$$

This condition ensures that the sum of the counting numbers of r and all of the regions above it will be 1. Intuitively, we can think of the counting number of the region r as correcting for overcounting or undercounting of the weight of the scope of r by regions above it. Now, assume that our region graph is structured so that, for each variable X_i, there is a unique region r_i such that every other region whose scope contains X_i is an ancestor of r_i. Then, we are guaranteed that both the connectedness and the counting number condition for X_i hold. We can similarly require that for any factor ϕ, there is a unique region r_ϕ such that any other region whose scope contains $Scope[\phi]$ is an ancestor of r_ϕ. This construction guarantees the requirements for factor connectedness and counting numbers.

It is easy to see that the Bethe region graph of example 11.2 satisfies both of these conditions. Moreover, this process of guaranteeing that a unique minimal region exists for each X_i is essentially what we did in example 11.3 to produce a valid region graph.

saturated region graph

These conditions provide us with a simple strategy for constructing a *saturated region graph.*

2. Some of the literature on region graphs use the terminology of parents and children of regions. To avoid the confusion with similar terminology in Bayesian networks, we use the terms *up* and *down* here.

Algorithm 11.3 Algorithm to construct a saturated region graph

```
Procedure Build-Saturated-Region-Graph (
  R    // a set of initial regions
)
1   repeat
2     For any r_1, r_2 ∈ R
3       Z ← Scope[r_1] ∩ Scope[r_2]
4       if Z ≠ ∅
5         and R does not contain a region with scope Z then
6           create region r with Scope[r] = Z
7           R ← R ∪ {r}
8     Until convergence
9     Initialize R as an empty graph with R as vertices
10    for each r_1 ≠ r_2 ∈ R
11      if Scope[r_2] ⊂ Scope[r_1] and
12      not exist r_3 ∈ R such that Scope[r_2] ⊂ Scope[r_3] ⊂ Scope[r_1] then
13          add an arc r_1 → r_2 to the region graph R
14    return R
```

We start with initial set of regions. Often, these regions will be the initial factors in P_Φ, although we can decide to work with bigger regions that capture some more global interactions. We then extend this set of regions into a valid region graph, where our goal is to represent appropriately any subset of variables that is shared by some of the regions. We therefore expand the set of regions to be closed under intersections. We connect these regions so that the upward closure of each region contains all of its supersets. The full procedure is shown in algorithm 11.3. Unlike the Bethe approximation, this region graph maintains the consistency of higher-order marginals. The example of figure 11.9 is an example of running this procedure on the original set of regions $\{A, B, C\}$, $\{B, C, D\}$, and $\{A, C, D\}$. As our previous discussion suggests, this procedure guarantees a region graph that satisfies the region graph condition.

Belief Propagation in Region Graphs Given a region graph, we are faced with the task of optimizing the free energy associated with its structure:

RegionGraph-Optimize:

Find $\boldsymbol{Q} = \{\beta_r : i \in \mathcal{V}_{\mathcal{R}}\}$
maximizing $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$
subject to

$$\sum_{\boldsymbol{C}_{r'} - \boldsymbol{C}_r} \beta_{r'}(\boldsymbol{c}_{r'}) = \beta_r(\boldsymbol{c}_r) \quad \forall i \in \mathcal{V}_{\mathcal{R}}, \forall r' \in \mathbf{Up}(r), \forall \boldsymbol{c}_r \in Val(\boldsymbol{C}_r) \tag{11.30}$$

$$\sum_{\boldsymbol{C}_r} \beta_r(\boldsymbol{c}_r) = 1 \quad \forall i \in \mathcal{V}_{\mathcal{R}} \tag{11.31}$$

$$\beta_r(\boldsymbol{c}_r) \geq 0 \quad \forall i \in \mathcal{V}_{\mathcal{R}}, \boldsymbol{c}_r \in Val(\boldsymbol{C}_r). \tag{11.32}$$

Our strategy for devising algorithms for solving this optimization problem is similar to the approach we took in section 11.3.6. Using the method of Lagrange multipliers, we characterize the stationary points of the target function (given the constraints) as a set of fixed-point equations. We then find an iterative algorithm that attempts to reach such a stationary point.

We first characterize the fixed point via the Lagrange multipliers. As before, we form a Lagrangian by introducing terms for each of the constraints: from equation (11.30), we obtain a Lagrange multiplier $\lambda_{r,r'}(\boldsymbol{c}_r)$ for every pair $r' \in \mathbf{Up}(r)$ and every $\boldsymbol{c}_r \in Val(\boldsymbol{C}_r)$; from equation (11.31), we obtain a Lagrange multiplier λ_r for every r and every $\boldsymbol{c}_r \in Val(\boldsymbol{C}_r)$; as before, we assume that we are dealing with interior fixed points only, and so do not have to worry about the inequality constraint. We differentiate the Lagrangian relative to each of the region beliefs $\beta_r(\boldsymbol{c}_r)$, and obtain the following set of *fixed-point equations*:

fixed-point equations

$$\kappa_r \ln \beta_r(\boldsymbol{c}_r) = \lambda_r + \kappa_r \psi_r(\boldsymbol{c}_r) - \sum_{r' \in \mathbf{Up}(r)} \lambda_{r,r'}(\boldsymbol{c}_r) + \sum_{r' \in \mathbf{Down}(r)} \lambda_{r',r}(\boldsymbol{c}_{r'}). \tag{11.33}$$

For regions for which $\kappa_r \neq 0$, we can rewrite this equation to conclude that:

$$\beta_r(\boldsymbol{c}_r) = \frac{1}{Z_r} \psi_r(\boldsymbol{c}_r) \left(\frac{\prod_{r' \in \mathbf{Down}(r)} \delta_{r \to r'}(\boldsymbol{c}_{r'})}{\prod_{r' \in \mathbf{Up}(r)} \delta_{r' \to r}(\boldsymbol{c}_r)} \right)^{1/\kappa_r}. \tag{11.34}$$

From this equality, one can conclude the following result:

Theorem 11.6

Assume that our region graph satisfies the family preservation property. Then, at fixed points of the RegionGraph-Optimize *optimization problem, we have that:*

$$P_\Phi(\mathcal{X}) \propto \prod_r (\beta_r)^{\kappa_r}. \tag{11.35}$$

The proof is derived from a simple cancellation of messages in the different terms (see exercise 11.16).

This result tells us that we can reparameterize the initial distribution P_Φ in terms of the final beliefs obtained as fixed points of the region graph optimization problem. It tells us that we can represent the distribution in terms of a calibrated set of beliefs for the individual regions. This

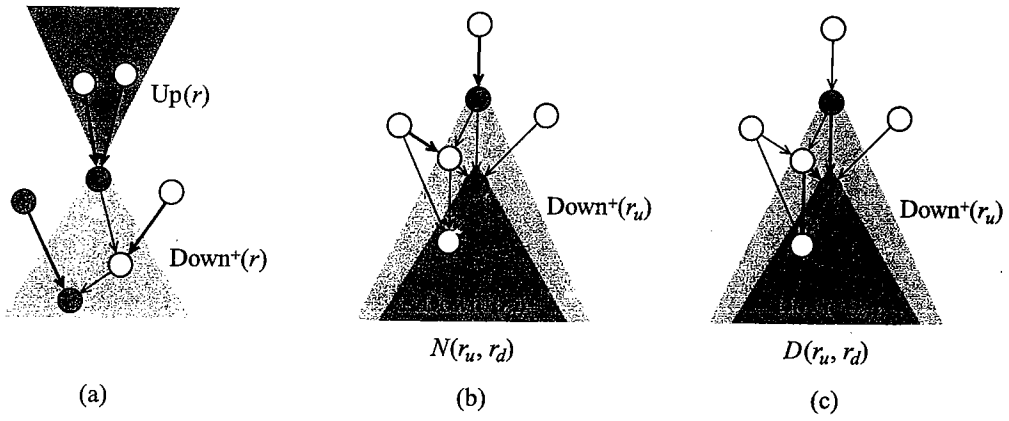


Figure 11.11 The messages participating in different region graph computations. Participating messages are marked with thicker arrows. (a) The computation of the beliefs $\beta_r(C_r)$; (b) The set $N(r_u, r_d)$ participating in the computation of the message $\delta_{r_u \to r_d}$; (c) The set $D(r_u, r_d)$ participating in the computation of the message $\delta_{r_u \to r_d}$.

result is a very powerful one, because it holds for any set of counting numbers that satisfies the family preservation property — a very large class. Of course, this result only shows that any fixed point is a reparameterization of the distribution, but not that such a reparameterization exists. However, under our assumption that all of the initial factors in Φ are strictly positive, one can show that such a fixed point, and hence a corresponding reparameterization, necessarily exists.

message passing algorithm

As we can see, unlike the case of cluster graphs, the fixed-point equations for region graphs are more involved, and do not lead directly to an elegant *message passing algorithm.* Indeed, the derivation of the update rules from the Lagrange multipliers often involves multiple steps of algebraic manipulation. (Although such derivations are possible in restricted cases; see exercise 11.17 and exercise 11.19.) In the remainder of this section, we present, without derivation, one set of update rules that can be derived from equation (11.34), in the specific case where the counting numbers are as presented in equation (11.29).

The basic idea is similar to the message update used in cluster graphs. There, each sepset carried messages sent by one of the clusters through the sepset to the neighboring cluster. We can think of that as a message from the cluster to the sepset. The analog of this in region graph is a message from a region to a region below it. Thus, for each pair $r_1 \to r_2$ in the region graph we will have a message $\delta_{r_1 \to r_2}$ whose scope is $Scope[r_2]$. All messages are associated with "downward" edges of the form $r_1 \to r_2$, but they are used to define the beliefs and messages of regions that contain them.

The definitions of the messages and the beliefs are somewhat involved. We begin by defining

the beliefs of a region as a function of these messages, which is somewhat simpler:

$$\beta_r(\boldsymbol{C}_r) = \psi_r(\boldsymbol{C}_r) \left(\prod_{r_u \in \mathbf{Up}(r)} \delta_{r_u \to r}(\boldsymbol{C}_r) \right) \left(\prod_{r_d \in \mathbf{Down}^*(r)} \prod_{r_u \in \mathbf{Up}(r_d) - \mathbf{Down}^+(r)} \delta_{r_u \to r_d}(\boldsymbol{C}_{r_d}) \right). \tag{11.36}$$

In other words, the belief of a region is the product of three groups of terms. The first two are very natural: the initial beliefs ψ_r (1 for all regions except the top ones) and the messages from its upward regions. The last group contain all messages sent to regions below the region r from regions other than the region r itself and regions below it. In other words, these are messages from "external" sources to regions below the region r; see figure 11.11a. Thus, the beliefs of the region are not influenced by messages it sends down, but only by messages sent to it or its subsets from other regions.

Again, it is instructive to compare this definition to our definition of beliefs in cluster graphs. In cluster graphs, the belief over a sepset is the product of messages from the neighboring clusters. These messages correspond in our case to messages from upward regions. The belief over a cluster $\boldsymbol{C}$ is the product of its initial potential and messages sent from neighboring clusters to the sepsets adjacent to $\boldsymbol{C}$. The sepsets correspond to the regions in $\mathbf{Down}^+(\boldsymbol{C})$; the message sent by another clique $\boldsymbol{C}'$ to this sepset corresponds to messages sent by an "external" source to a region in $\mathbf{Down}^+(\boldsymbol{C})$.

We now move to defining the computation of a message from r_u to r_d, also illustrated in figure 11.11b,c:

$$\delta_{r_u \to r_d}(\boldsymbol{C}_{r_d}) = \frac{\sum_{\boldsymbol{C}_{r_u} - \boldsymbol{C}_{r_d}} \psi_{r_u}(\boldsymbol{C}_{r_u}) \prod_{r_1 \to r_2 \in N(r_u, r_d)} \delta_{r_1 \to r_2}(\boldsymbol{C}_{r_2})}{\prod_{r_1 \to r_2 \in D(r_u, r_d)} \delta_{r_1 \to r_2}(\boldsymbol{C}_{r_2})}. \tag{11.37}$$

The numerator involves the initial factor assigned to the region, and a product of messages associated with the set of edges

$$N(r_u, r_d) = \{(r_1 \to r_2) \in \mathcal{E}_{\mathcal{R}} : r_1 \notin \mathbf{Down}^+(r_u), r_2 \in \mathbf{Down}^+(r_u) - \mathbf{Down}^+(r_d)\}.$$

This set contains edges from sources "external" to r_u that are outside the scope of influence of r_d; that is, they either enter r_u directly, or enter regions below r_u that are not below r_d. The denominator involves a product of the messages in the set:

$$D(r_u, r_d) = \{(r_1 \to r_2) \in \mathcal{E}_{\mathcal{R}} : r_1 \in \mathbf{Down}^+(r_u) - \mathbf{Down}^+(r_d), r_2 \in \mathbf{Down}^+(r_d)\}.$$

This set counts information that would be passed from r_u to regions below r_d *indirectly* — not through r_d. We want to divide by these messages, since otherwise the same information would be incorporated multiple times into the beliefs and the messages.

Example 11.5 *We now return to the region graph of figure 11.9 that corresponds to the potential function we discussed in example 11.3.*

Applying equation (11.36) to this example, we get a set of equation representing the potentials as function of the initial factors and the messages:

$$\begin{aligned}
\beta_1 &= \psi_1 \delta_{2\to 4} \delta_{3\to 5} \delta_{6\to 7} \\
\beta_2 &= \psi_2 \delta_{1\to 4} \delta_{3\to 6} \delta_{5\to 7} \\
\beta_3 &= \psi_3 \delta_{1\to 5} \delta_{2\to 6} \delta_{4\to 7} \\
\beta_4 &= \delta_{1\to 4} \delta_{2\to 4} \delta_{5\to 7} \delta_{6\to 7} \\
\beta_5 &= \delta_{1\to 5} \delta_{3\to 5} \delta_{4\to 7} \delta_{6\to 7} \\
\beta_6 &= \delta_{2\to 6} \delta_{3\to 6} \delta_{4\to 7} \delta_{5\to 7} \\
\beta_7 &= \delta_{4\to 7} \delta_{5\to 7} \delta_{6\to 7}.
\end{aligned}$$

Applying equation (11.37), we can construct the messages. For example,

$$\delta_{4\to 7} = \sum_B \delta_{1\to 4} \delta_{2\to 4}.$$

One easy way to derive this message directly is to use the marginal consistency constraint:

$$\beta_7 = \sum_B \beta_4.$$

Plugging in the expanded form of the two beliefs, we get

$$\delta_{4\to 7} \delta_{5\to 7} \delta_{6\to 7} = \sum_B \delta_{1\to 4} \delta_{2\to 4} \delta_{5\to 7} \delta_{6\to 7}.$$

If we now isolate $\delta_{4\to 7}$ we get

$$\delta_{4\to 7} = \frac{\sum_b \delta_{1\to 4} \delta_{2\to 4} \delta_{5\to 7} \delta_{6\to 7}}{\delta_{5\to 7} \delta_{6\to 7}}.$$

After we cancel out the common terms $\delta_{5\to 7}$ and $\delta_{6\to 7}$, we get the desired form.

This message is essentially identical to the message in a cluster graph where we marginalize the other incoming messages in the cluster to send a message to a particular sepset. Here region 4 behaves as a cluster and region 7 as a sepset. The other messages incoming to region 7 have a similar form.

Messages into the middle layer regions have more complex form. For example

$$\delta_{1\to 4} = \frac{\sum_A \psi_1 \delta_{3\to 5}}{\delta_{2\to 4} \delta_{5\to 7}}.$$

Again, we can use the marginal consistency constraint

$$\beta_4 = \sum_A \beta_1$$

to reconstruct the message. Plugging in the expanded form of the two beliefs, and isolating $\delta_{1\to 4}$ we get

$$\delta_{1\to 4} = \frac{\sum_A \psi_1 \delta_{2\to 4} \delta_{3\to 5} \delta_{6\to 7}}{\delta_{2\to 4} \delta_{5\to 7} \delta_{6\to 7}}.$$

After we cancel out $\delta_{6\to 7}$, we get the desired form. ■

These definitions set up a message passing algorithm similar to CGraph-SP-Calibrate, except that we use the messages as formulated in equation (11.37). As with belief propagation on cluster graphs, we can prove that convergence points of such propagations are stationary points of the RegionGraph-Optimize optimization problem.

Theorem 11.7 *A set of beliefs Q is a stationary point of* RegionGraph-Optimize *for region graph $\mathcal{R}$ if and only if for every edge $(i-j) \in \mathcal{E}_{\mathcal{R}}$ there are auxiliary factors $\delta_{u \to d}(C_d)$ that satisfy equation (11.36) and equation (11.37).*

This result is a direct generalization of theorem 11.5, and is proved in a similar way. We leave the detail as an exercise (see exercise 11.14). Much of the discussion following theorem 11.5 applies here. In particular, we do not have guarantees that iterations of message passing will converge. However, if they do, we have reached a stationary point of the energy functional. In practice, the experience is that when we consider moving from the Bethe approximation to "richer" region graphs that contain intermediate regions with larger subsets, problems of nonconverging runs are less common. For example, a region graph construction for grids is much more convergent than the corresponding cluster graph (see exercise 11.15). However, except for special cases (for example, region graphs that correspond to cluster trees), we do not know how to characterize region graphs where belief propagation converges.

11.3.8 Discussion

Cluster-graph belief propagation methods such as the ones we have described in this chapter provide a general-purpose mechanism for approximate inference in graphical models. In principle, they apply to any network, including networks with high tree-width, for which exact inference is intractable. They have been applied successfully to a large number of dramatically different applications, including (among many others) message decoding in communication over a noisy channel (see box 11.A), predicting protein structure (see box 20.B), and image segmentation (see box 4.B).

However, **it is important to keep in mind that cluster-graph belief propagation is not a global panacea to the problem of inference in graphical models. The algorithm may not converge, and when it does converge, there may be multiple different convergence points.** Although there are currently no conditions characterizing precisely when cluster-graph belief propagation converges, several factors seem to play a role.

The first is the topology of the network: A network containing a large number of short loops is more likely to be nonconvergent. Although this notion has been elusive to characterize in practice, it has been shown that cluster-graph belief propagation is guaranteed to converge on networks with a single loop.

An even more significant factor is the extent to which the factors parameterizing the network are skewed, or close to deterministic. Intuitively, deterministic factors can cause difficulties in several ways. First, they often induce strong correlations between variables, which cluster-graph belief propagation (depending on the approximation chosen) can lose. This error can have an effect not only for the correlated variables, but also for marginals of variables that interact with both. Second, close-to-deterministic factors allow information to be propagated reliably through long paths in the network. Recall that part of our motivation for the running intersection property was to prevent information about some variable to be propagated infinitely through a

loop. While the running intersection property prevents such loops from occurring structurally, deterministic potentials allow us to recreate them using an appropriate choice of parameters. For example, if A is deterministically equal to B, then we can have a cycle of clusters where A appears in some of the clusters and B in others. Although this cluster graph may satisfy the running intersection property relative to A, effectively there is a cycle in which the same variable appears in all clusters. Finally, as we discussed in section 11.3.4, factors that are less skewed provide smoothing of the messages, reducing oscillations; indeed, one can even prove that, if the skew of the factors in the network is sufficiently bounded, it can give rise to a contraction property that guarantees convergence.

☞ **In summary, the key factor relating to convergence of belief propagation appears to be the extent to which the network contains strong influences that "pull" a variable in different directions. Owing to its local nature, the algorithm is incapable of reconciling these different constraints, and it can therefore oscillate as different messages arrive that pull it in one direction or another.**

A second problem relates to the quality of the results obtained. Despite the appeal and importance of the energy-based analysis, it does not (except in a few rare cases — see section 11.7) provide any guarantees on the accuracy of the marginals obtained by cluster-graph belief propagation. This is in contrast to the sampling-based methods of chapter 12, where we are at least assured that, if we run the algorithm for long enough, we will obtain accurate estimates of the posteriors. (Of course, the key question of "how long is long enough" does not usually have an answer, so it is not clear how important this distinction is in practice.) Empirical results show that, in the settings where cluster-graph belief propagation convergence is more likely (not too many tight loops, no highly skewed factors), one also often obtains reasonable answers.

Importantly, these answers are often good but overconfident: The value $x \in Val(X)$ to which cluster-graph belief propagation gives the highest probability is often the value for which $P_\Phi(X = x)$ is indeed the highest, but the probability assigned to x by the approximation is often too high. This phenomenon arises (partly) from the fact the cluster-graph belief propagation ignores correlations between messages and can therefore count the same piece of evidence multiple times as it arrives along different paths, leading to overly strong conclusions. In other cases, however, the answers obtained by cluster-graph belief propagation are simply wrong (see section 11.3.1); unfortunately, there is currently no way of determining when the answers returned by a run of cluster-graph belief propagation are reasonable approximations to the true marginals.

The intuitions described previously do help us, however, to design approximations that are more likely to produce good answers. In general, we cannot construct a cluster graph that preserves all of the higher-order interactions among the factors. Hence, we need to decide which factors to include in the cluster graph and how to relate them. As the preceding discussion suggests, we do better if we construct approximations that incorporate tight loops and maintain the strongest factors within clusters as much as possible. While these intuitions provide reasonable rules of thumb on how to construct approximations, it is not obvious how to capture them within a general-purpose automated cluster-graph construction procedure.

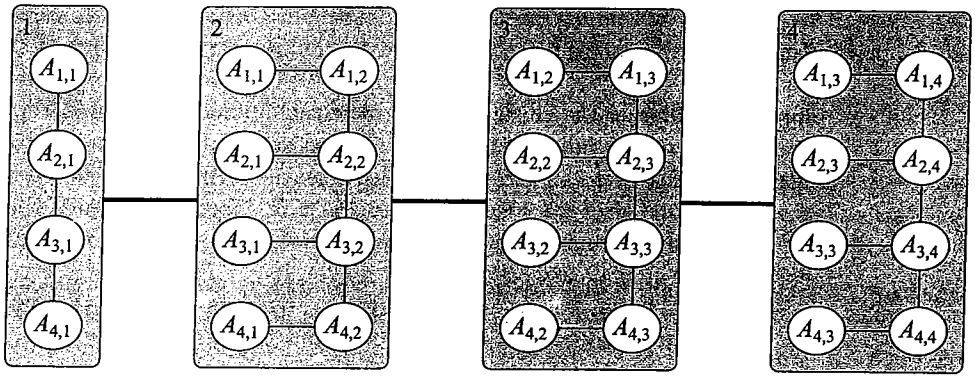


Figure 11.12 A cluster for a 4×4 grid network. The structure within each cluster represents the arcs whose factors are assigned to that cluster.

11.4 Propagation with Approximate Messages ⋆

The cluster-graph belief propagation methods achieved approximation by "relaxing" the requirement of having a cluster tree. Instead, we used a cluster graph and constructed an approximation by a set of pseudo-marginals. This approximation avoided the need to construct large clusters, which incur an exponential cost in terms of memory and running time. **In this section, we consider an approach in which the simplification occurs within a given cluster structure; rather than simplify this structure, we perform approximate propagation steps within it. This allows us to keep the correct clusters and gain efficiency by using more compact representations and operations on these clusters (at the cost of introducing approximations).** Importantly, this approach is orthogonal to the methods we described in the previous section, in that approximate message passing can occur both within a clique tree or a cluster graph approximation. For ease of presentation, we focus on the case of clique trees in this section, but the ideas easily carry through to the more general setting.

The basic concept behind the methods described in this section is the use of approximate messages in clique tree (or cluster graph) propagation. Instead of representing messages in the clique trees as factors, we use more compact representations of approximate factors. There are many different schemes that we can use for approximating messages. To ground the discussion, we begin in section 11.4.1 by describing one important instantiation of this general framework, which is very natural in our setting — message approximation using a factored form (for example, as a product of marginals). In section 11.4.2 and section 11.4.3 we then discuss algorithms that perform message passing using approximate messages. In section 11.4.4 we describe a general algorithm, called *expectation propagation*, that applies to any approximation in the exponential family. Finally, in section 11.4.5, we show that the expectation propagation algorithm for the exponential family can be derived from a variational analysis, similar to the ones we discussed in the previous section.

expectation propagation

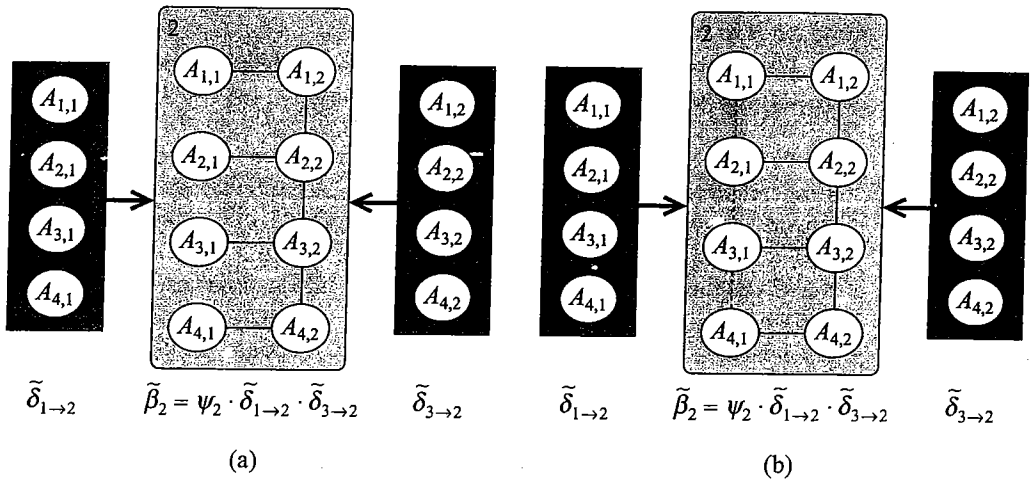


Figure 11.13 Effect of different message factorizations on the structure of the belief factor $\tilde{\beta}_2$ in the example of figure 11.12.

11.4.1 Factorized Messages

We begin by considering a concrete example where message approximation may be useful. Consider again the 4×4 grid network of figure 11.4. We can construct a cluster tree for this network as shown in figure 11.12. (Note that this cluster tree is not the optimal one for this network.) In our discussion of inference until now, we ignored the inner structure of clusters and treated the cluster as being associated with a single factor. Now, however, we keep track of the structure of the original factors associated with the cluster. We will see shortly why keeping this structure will help us. In figure 11.12 this structure is portrayed by "subnetworks" within each cluster.

Calibration in this cluster tree involves sending messages that are joint measures over four variables. An intuitive idea is to simplify the messages by using a factored representation. For example, consider the message $\delta_{1\to 2}$, and suppose that we approximate it by a factored form

$$\tilde{\delta}_{1\to 2}[A_{1,1}, A_{2,1}, A_{3,1}, A_{4,1}] = \tilde{\delta}_{1\to 2}[A_{1,1}]\tilde{\delta}_{1\to 2}[A_{2,1}]\tilde{\delta}_{1\to 2}[A_{3,1}]\tilde{\delta}_{1\to 2}[A_{4,1}].$$

What can we gain from such an approximation? Clearly, this form provides a more concise representation of the message. Instead of representing the message with values for the sixteen possible joint assignments, we send a message with two values for each variables (leading to a total of eight "parameters"). If we consider bigger grids, then the saving will be more substantial.

Does this saving gain us efficiency? Naively, the answer is no. Recall that the main computational cost of exact inference is generating the message in a cluster. This requires multiplying incoming messages with the original factor of the cluster and marginalization. In our example, C_2 involves eight variables, and so this operation should involve operations over the $2^8 = 256$ joint values of these variables. Thus, one might argue that the saving of eight parameters in representing the message does not deal with the core computational problem we have at hand

and leads to negligible improvement.

It turns out, however, that if we consider the internal structure of the cluster we can exploit the factored form of the message. Consider computing the beliefs over C_2 given messages from C_1 and C_3. In exact computation, we multiply the potential ψ_2 with $\delta_{1\to 2}$ and $\delta_{3\to 2}$ and normalize to get the beliefs about C_2. However, if we approximate both messages by a product of univariate terms, we notice that the product of the messages with the factors in C_2 forms a network structure that we can exploit. In our example, this network is a tree-structured network shown in figure 11.13a. We can easily calibrate this network and answer queries about the beliefs over C_2 without enumerating all joint assignments to variables in this cluster.

We can get similar savings even if we use a richer approximation that can better capture dependencies among variables in the message. Suppose we approximate $\delta_{1\to 2}$ by a factored form that corresponds to the chain structure $A_{1,1}$—$A_{2,1}$—$A_{3,1}$—$A_{4,1}$. This approximation makes conditional independence assumptions about the variables, but it captures some of the dependencies among them. In this example, this representation actually captures the message $\delta_{1\to 2}$. However, we can check that $\delta_{3\to 2}$ does not satisfy the conditional independence of $A_{1,2}$ and $A_{3,2}$ given $A_{2,2}$. Thus, in this case, a chain representation is an approximation. If we multiply these two approximations with the cluster C_2, we get a set of factors that has the structure shown in figure 11.13b. Although not a tree, this graph has a tree-width of 2, regardless of the grid size. Thus, once again, we can use exact inference methods on the resulting product of factors.

factor set

We can exploit this intuition by maintaining both the initial potentials and the messages as *factor sets*. For the initial potential, these factors are the parameterization of the original network; for messages, these factors are introduced by the approximation. A factor set $\vec{\phi} = \{\phi_1, \ldots, \phi_k\}$ provides a compact representation for the higher-dimensional factor $\phi_1 \cdot \phi_2 \cdot \cdots \cdot \phi_k$.

factor set product

Recall that belief propagation involves two main operations: product and marginalization. The *product* of factor sets is easy. Suppose we have the factor sets $\vec{\phi}_1$ and $\vec{\phi}_2$. The factor set $\vec{\phi}_1 \cdot \vec{\phi}_2$ is simply the factor set that contains the union of the two factor sets (we assume that the components of the factor sets are distinct).

factor set marginalization

The difficult operation is *marginalization*. Suppose we have a factor set $\vec{\phi} = \{\phi_1, \ldots, \phi_k\}$, and we consider the marginalization $\sum_X \vec{\phi}$. This operation couples all the factors that contain X. In general, for a well-constructed clique tree, the message that results from marginalizing a clique will not satisfy any conditional independence statements and therefore cannot be factorized (see exercise 11.21).

Returning to our example of figure 11.12, one of our inference steps is to compute:

$$\delta_{2\to 3} = \sum_{A_{1,1}, A_{2,1}, A_{3,1}, A_{4,1}} \psi_2 \cdot \delta_{1\to 2}.$$

To achieve the efficient inference steps we discussed earlier, we want to approximate $\delta_{2\to 3}$ by a product of simpler terms. This is an instance of a problem we encountered in section 8.5 — approximating a given distribution by another distribution from a given family of distributions. In our case, the approximating family is the set of distributions that take a particular product form. As we discussed in section 8.5, there are several ways of projecting a distribution P into some constrained class of distributions $\mathcal{Q}$. Indeed, throughout our discussion in this chapter so far, we have been searching for a distribution Q which is the I-projection of P_Φ — the one

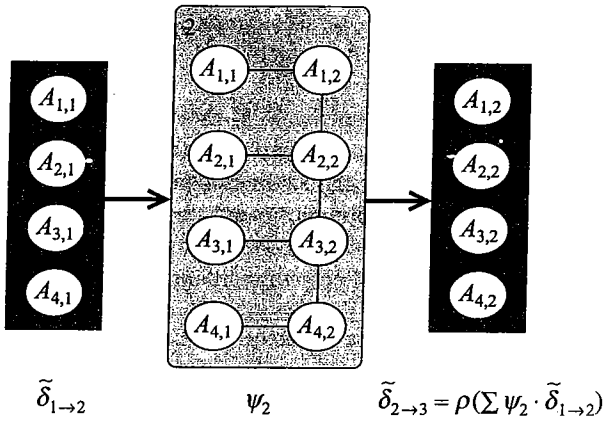


Figure 11.14 Example of propagation in cluster tree with factorized messages

M-projection

that minimizes $\boldsymbol{D}(Q\|P_\Phi)$. However, as we discussed in section 8.5, we cannot compute the I-projection of a distribution in closed form. Conversely, the *M-projection* for many classes $\mathcal{Q}$ of factored distributions has an easy closed form expression. For example, the M-projection of P onto the class of factored distributions is simply the product of marginals of P (see proposition 8.3). We can compute these marginals by computing the marginals of the individual variables in the message. We note that the message is often not normalized, and is therefore not a distribution; however, we can normalize the message and treat it as a distribution without changing the final posterior obtained by the inference algorithm (see exercise 9.3).

Given the simplicity of M-projections, one might wonder why we used I-projections in our discussion so far. There are two main reasons. First, the theory of energy functionals allows us to use I-projections to lower-bound the partition function. Second, and more importantly, computing marginal probabilities for our distribution P_Φ is generally intractable. Thus, although the M-projection has a simple form, it is infeasible to apply when we have an intractable network structure. In our current setting, the situation is different. Here, our projection task is to approximate the outgoing message from a cluster $\boldsymbol{C}$. As discussed earlier, we assume that the product of the beliefs and the approximated messages for $\boldsymbol{C}$ is a factor set with tractable structure. This assumption allows us to use exact inference *within the cluster* $\boldsymbol{C}$ to compute the marginal probabilities needed for M-projections.

11.4.2 Approximate Message Computation

We now discuss in greater detail the task of computing factorized messages. To begin, consider the projection of $\delta_{2\to 3}$ into a fully factorized representation. According to proposition 8.3, to build the M-projection of $\delta_{2\to 3}$ onto a factored distribution, we need to compute the marginals of $A_{1,2}$, $A_{2,2}$, $A_{3,2}$, and $A_{4,2}$ in $\delta_{2\to 3}$. Since $\delta_{2\to 3}$ is defined to be the marginal of the factor set $\psi_2 \cdot \delta_{1\to 2}$, we need to compute terms such as $P_{\psi_2\cdot\delta_{1\to 2}}(A_{2,1})$, where we use $P_{\psi_2\cdot\delta_{1\to 2}}$ to denote the measure represented by the factor set. At an abstract level, this operation involves computing $\delta_{2\to 3}$ and then projecting it onto the simpler representation. However, if we examine

the structure of this factor set (see figure 11.14), we see that this computation can be done using standard exact inference on the factor set. In this particular case, the factor set is a tree network, and so inference is cheap. Similarly, we can compute the marginals over the variables in $\delta_{2\to 3}$. Thus, we can use the properties of M-projections and exact inference to compute the resulting projected message without ever explicitly enumerating the joint over the four variables in $\delta_{2\to 3}$. In an $n \times n$ grid, for example, we can perform these operations in time that is linear in n, whereas the explicit computation would have constructed a factor of exponential size. As we discussed, the more complex approximation of figure 11.13b also has a bounded tree-width of 2, and therefore also allows linear time inference.

Algorithm 11.4 Projecting a factor set to produce a set of marginals over a given set of scopes

```
Procedure Factored-Project (
    φ⃗,    // an input factor set
    𝒴,    // A set of desired output scopes
)
1   Build an inference data structure 𝒰
2   Initialize 𝒰 with factors in φ⃗
3   Perform inference in 𝒰
4   φ⃗^o ← ∅
5   for all Y_j ∈ 𝒴
6     Find a clique C_j in 𝒰 so that Y_j ⊆ C_j
7     ψ_j ← β_𝒰(Y_j)    // marginal of Y_j in 𝒰
8     φ⃗^o ← φ⃗^o ∪ {ψ_j}
9   return φ⃗^o
```

The overall structure of the algorithm is shown in algorithm 11.4. At a high level, we can view exact inference in the factor set as a "black box" and not concern ourselves with the exact implementation. However, it is also useful to consider how this type of projected message may be computed efficiently. Most often, the inference data structure $\mathcal{U}$ is a cluster graph or a clique tree, into which the initial factors $\vec{\phi}$ can easily be incorporated, and from which the target factors, of the appropriate scopes, can be easily extracted. To allow for that, we typically design the cluster graph to be family-preserving with respect to both sets of factors. Under that assumption, we can extract a factor ψ_j over the scope $\boldsymbol{Y}_j$ by identifying a cluster $\boldsymbol{C}_j$ in $\mathcal{U}$ whose scope contains $\boldsymbol{Y}_j$, and marginalizing it over $\boldsymbol{Y}_j$. As an alternative approach, we can construct an unconstrained clique tree, and use the out-of-clique inference algorithm of section 10.3.3.2 to extract from the graph the joint marginals of subsets of variables that are not together in a clique. (We note that out-of-clique inference is more challenging in the context of cluster graphs, since the path used to relate the clusters containing the query variables can affect the outcome of the computation; see exercise 11.22.)

If our representation of a message is simply a product of marginals over disjoint subsets of variables, this algorithm suffices to produce the output message: We produce a factor over each (disjoint) scope $\boldsymbol{Y}_j$; the product of these factors is the M-projection of the distribution. But for

richer approximations, the required operation is somewhat more complex.

Example 11.6

Consider again our grid example of figure 11.13b. Here, the clique needs to take in messages that involve factors over pairs of variables $A_{i,1}, A_{i+1,1}$ and produce messages over pairs of variables $A_{i,2}, A_{i+1,2}$. We can accomplish this goal by constructing, within this clique, a nested clique tree that has at least one clique containing each of these pairs. For this purpose, we can use any clique tree based on triangulating the structure inside the clique in figure 11.13b. For example, we can have a tree with the cliques $\{A_{1,1}, A_{1,2}, A_{2,2}\}$, $\{A_{1,1}, A_{2,1}, A_{2,2}\}$, $\{A_{2,1}, A_{2,2}, A_{3,2}\}$, $\{A_{2,1}, A_{3,1}, A_{3,2}\}$, $\{A_{3,1}, A_{3,2}, A_{4,2}\}$, $\{A_{3,1}, A_{4,1}, A_{4,2}\}$.

Our goal now is to produce a set of factors that is the M-projection of the true message onto the chain. Assume that we extract from the clique tree the pairwise marginals $P(A_{1,2}, A_{2,2})$, $P(A_{2,2}, A_{3,2})$, and $P(A_{3,2}, A_{4,2})$. However, we cannot directly encode the distribution using these factors as a factor set, since this approach would double-count the probability of the singletons $A_{2,2}$ and $A_{3,2}$, each of which appears in two factors. To produce the true M-projected distribution, we need to divide by the double-counted marginals $A_{2,2}$ and $A_{3,2}$.

We can achieve this correction easily by computing these node marginals and adding to our factor set representation two factors

$$\frac{1}{\tilde{\delta}_{2\to 3}(A_{2,2})}, \frac{1}{\tilde{\delta}_{2\to 3}(A_{3,2})}$$

that compensate for the double-counting. This factor set represents the M-projection of the distribution, and it can be used in subsequent message passing steps. Equivalently, we are representing the distribution as a calibrated clique tree over $A_{1,2}, A_{2,2}, A_{3,2}, A_{4,2}$, where the factors derived from the pairwise marginals are the clique beliefs, the factors derived from inverted node marginals are the sepset messages, and the overall distribution is encoded as in equation (10.11). ■

We can easily generalize this approach to more complex message representations. Most generally, assume that we choose to encode our message between cluster i and cluster j using a representation as in equation (11.35); more precisely, we have some set of factors $\{\beta_r(\boldsymbol{X}_r) : \boldsymbol{X}_r \in \mathbf{R}_{i,j}\}$, each raised to some power:

$$\tilde{\delta}_{i\to j} = \prod_{r\in\mathbf{R}_{i,j}} (\beta_r(\boldsymbol{X}_r))^{\kappa_r}. \tag{11.38}$$

We can compute this approximation in our procedure using the Factored-Project algorithm, using the set $\{\boldsymbol{X}_r \in \mathbf{R}_{i,j}\}$ as $\mathcal{Y}$. The output of this procedure is a set of calibrated marginals over these scopes, and can therefore be plugged into equation (11.38) to produce a message in the appropriate factored form. The same factors, each raised to its appropriate power, can be used as the factorized message passed to cluster j.

Importantly, we note that this approach does not always provide an exact M-projection of the true message. This property is guaranteed only when the set of regions forms a clique tree, allowing the M-projection to be calculated in closed form using equation (11.38); in other cases, we obtain only an approximation to the M-projection. In practice, we often choose some simple clique tree, as in example 11.6, to encode our distribution, allowing the M-projection to be performed easily and exactly. However, in some cases, a clique tree approximation, to stay tractable, is forced to make too many independence assumptions, leading to a poor

approximation of the message. Hence, the approach of an approximation M-projection into a richer class of distributions may provide a useful alternative in some cases.

In summary, we have shown how we can perform an approximate message passing step with structured messages. The algorithm maintains messages and beliefs in factored form. Factors are entered into a nested clique tree or cluster graph, and message propagation is used to compute the factors describing the output message. In the fully factored case, this representation is simply a set of factors, and the multiplication operation used in message passing is simply a union of factor sets. In the more complex setting, we may need to postprocess the set of marginals — exponentiating them by appropriate counting numbers — to eliminate double-counting. The resulting set of factors is the compact representation of the outgoing message.

11.4.3 Inference with Approximate Messages

Equipped with these operations on factor sets, we can consider how to use these tools within cluster tree propagation. Our algorithm will maintain the beliefs at each cluster i using a factor set $\vec{\phi}_i$. Initially, we are given a cluster tree $\mathcal{T}$ with an assignment of the original factors to clusters. We have also determined the factorized form for each sepset, in terms of a network structure $\mathcal{G}_{i,j}$ that describes the desired factorization for $\tilde{\delta}_{i \to j}$ and $\tilde{\delta}_{j \to i}$.

There are two main strategies for applying the ideas described in the previous section to define an approximate message passing algorithms in clique trees. One is based on a sum-product message passing scheme, and the other on belief update messages. As we will see, although these two strategies are equivalent in the exact inference setting, they lead to fairly different algorithms when we consider approximations.

For notational simplicity, we introduce the notation M-project-distr$_{i,j}$ to be the combined operation of marginalizing variables that do not appear in $S_{i,j}$ and performing the M-projection onto the set of distributions that can be factorized according to $\mathcal{G}_{i,j}$.

11.4.3.1 Sum-Product Propagation

Consider the application of sum-product propagation (CTree-SP-Calibrate, algorithm 10.2), to our grid example. In this case, we need to perform the following operations:

$$\begin{aligned}
\delta_{1 \to 2} &= \psi_1 \\
\delta_{2 \to 3} &= \sum_{A_{1,1}, \ldots, A_{4,1}} \psi_2 \cdot \delta_{1 \to 2} \\
\delta_{3 \to 4} &= \sum_{A_{1,2}, \ldots, A_{4,2}} \psi_3 \cdot \delta_{2 \to 3} \\
\delta_{4 \to 3} &= \sum_{A_{1,4}, \ldots, A_{4,4}} \psi_4 \\
\delta_{3 \to 2} &= \sum_{A_{1,3}, \ldots, A_{4,3}} \psi_3 \cdot \delta_{4 \to 3} \\
\delta_{2 \to 1} &= \sum_{A_{1,2}, \ldots, A_{4,2}} \psi_3 \cdot \delta_{3 \to 2}.
\end{aligned}$$

A straightforward application of approximation is to replace each of the messages by the M-projected version:

$$\begin{aligned}
\tilde{\delta}_{1\to 2} &= \text{M-project-distr}_{1,2}(\psi_1) \\
\tilde{\delta}_{2\to 3} &= \text{M-project-distr}_{2,3}(\psi_2 \cdot \tilde{\delta}_{1\to 2}) \\
\tilde{\delta}_{3\to 4} &= \text{M-project-distr}_{3,4}(\psi_3 \cdot \tilde{\delta}_{2\to 3}) \\
\tilde{\delta}_{4\to 3} &= \text{M-project-distr}_{3,4}(\psi_4) \\
\tilde{\delta}_{3\to 2} &= \text{M-project-distr}_{2,3}(\psi_3 \cdot \tilde{\delta}_{4\to 3}) \\
\tilde{\delta}_{2\to 1} &= \text{M-project-distr}_{1,2}(\psi_2 \cdot \tilde{\delta}_{3\to 2}).
\end{aligned}$$

Each of these projection operations can be performed using the procedure described in the previous section.

sum-product expectation propagation

More generally, our *sum-product expectation propagation* procedure is identical to sum-product propagation of section 10.2, except that we modify the basic propagation procedure SP-Message so that, rather than simply marginalizing the product of factors, it computes their M-projection. Otherwise, the general structure of the propagation procedure is maintained exactly as before. Each cluster collects the messages from its neighbors and when possible sends outgoing messages. As for the original sum-product message passing, this process converges after a single upward-and-downward pass over the clique tree structure. Thus, unlike most of the other approximations we discuss in this chapter, this procedure terminates in a fixed number of steps.

Note that, after performing the propagation, the final beliefs over the individual clusters in the tree are not an explicit representation of a joint distribution over the variables in the cluster. In this case, the final beliefs over a cluster $\boldsymbol{C}$ are represented in a factorized form, as a set of beliefs. In our running example, the computed beliefs over $\boldsymbol{C}_2$ have the form

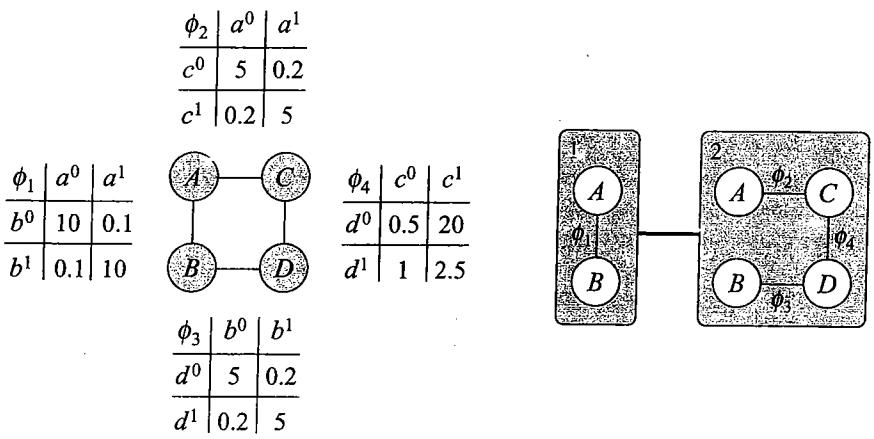
$$\tilde{\beta}_2 = \psi_2 \cdot \tilde{\delta}_{1\to 2}\tilde{\delta}_{3\to 2},$$

and the structure shown in figure 11.13a. Because this structure allows tractable inference, we can answer queries about the posterior distribution of these variables using standard inference methods, such as variable elimination or clique tree inference.

These computational benefits come at a price. The exact beliefs over $\boldsymbol{C}_2$ are not decomposable (see exercise 11.20). And thus, although forcing a particular independence structure on the marginal distribution has computational advantages, it does lose information. With this caveat in mind, we can still question whether the algorithm finds the best possible approximation within the set of constraints we are considering.

Example 11.7

To get a sense of the issues, consider the simple example shown in figure 11.15a. In this small network, the potential $\phi_1(A, B)$ introduces a strong coupling between A and B, with a strong preference for $A = B$. The potentials $\phi_2(A, C)$, $\phi_3(B, D)$ incur weaker coupling between A and C and B and D. Finally, the potential $\phi_4(C, D)$ has a strong preference for $C = c^1$, with a weaker preference for $C \neq D$.



(a) Markov network (b) cluster tree

Figure 11.15 Markov network used to demonstrate approximate message passing. (a) A simple Markov network with four pairwise potentials. (b) A clique tree for this network.

If we perform exact inference in this network, we find the following marginal posteriors:

$$\begin{aligned} P(a^0, b^0) &= 0.274 & P(c^0, d^0) &= 0.102 \\ P(a^0, b^1) &= 0.002 & P(c^0, d^1) &= 0.018 \\ P(a^1, b^0) &= 0.041 & P(c^1, d^0) &= 0.368 \\ P(a^1, b^1) &= 0.682 & P(c^1, d^1) &= 0.512. \end{aligned}$$

We see that the preference for $C = c^1$ is reflected in this distribution (with the marginal distribution $P(c^1) = 0.879$). In addition, the strong coupling between A and B is propagated through the network, which results in making $D = d^1$ more probable when $C = c^1$.

What happens when we perform inference using the cluster tree of figure 11.15b and use approximate messages that are products of marginals? It is easy to see that, because ϕ_1 is symmetric, we get that $\tilde{\delta}_{1\to 2}[a^1] = 0.5$, and $\tilde{\delta}_{1\to 2}[b^1] = 0.5$. We can compare the exact message and the approximate one

$$\begin{aligned} \delta_{1\to 2}(a^0, b^0) &= 0.495 & \tilde{\delta}_{1\to 2}(a^0, b^0) &= 0.5 * 0.5 = 0.25 \\ \delta_{1\to 2}(a^0, b^1) &= 0.005 & \tilde{\delta}_{1\to 2}(a^0, b^1) &= 0.5 * 0.5 = 0.25 \\ \delta_{1\to 2}(a^1, b^0) &= 0.005 & \tilde{\delta}_{1\to 2}(a^1, b^0) &= 0.5 * 0.5 = 0.25 \\ \delta_{1\to 2}(a^1, b^1) &= 0.495 & \tilde{\delta}_{1\to 2}(a^1, b^1) &= 0.5 * 0.5 = 0.25. \end{aligned}$$

Thus, the approximate message is one where each joint assignment to A and B is equiprobable. This approximation loses the coupling that ϕ_1 introduces between A and B, and therefore it is a poor approximation to the exact message.

If we multiply this approximate message into the clique C_2. The initial factor here is $\psi_2 =$

$\phi_2 \cdot \phi_3 \cdot \phi_4$, and after multiplying it with $\tilde{\delta}_{1\to 2}$ we get the beliefs

$$\begin{aligned}\tilde{\beta}_2(c^0, d^0) &= 0.021\\ \tilde{\beta}_2(c^0, d^1) &= 0.042\\ \tilde{\beta}_2(c^1, d^0) &= 0.833\\ \tilde{\beta}_2(c^1, d^1) &= 0.104.\end{aligned}$$

Note that this factor is essentially a normalization of ϕ_4, since the message $\tilde{\delta}_{1\to 2}$ puts a uniform distribution of A and B, and since ϕ_2 and ϕ_3 are symmetric. Because the information about the coupling between A and B is not propagated into this cluster, we lose the consequent coupling between C and D, and the resulting approximation to $P(C, D)$ is also quite poor.

By contrast, if we now compute $\tilde{\delta}_{2\to 1}$, we get that

$$\begin{aligned}\tilde{\delta}_{2\to 1}[a^1] &= 0.904\\ \tilde{\delta}_{2\to 1}[b^1] &= 0.173.\end{aligned}$$

This message ascribes high probability to a^1 and a low one to b^1. This is quite different from the original coupling introduced by ϕ_1. Thus, when we combine the two to get the approximated posterior over A and B, we get the following beliefs factor:

$$\begin{aligned}\tilde{\beta}_1(a^0, b^0) &= 0.326\\ \tilde{\beta}_1(a^0, b^1) &= 0.001\\ \tilde{\beta}_1(a^1, b^0) &= 0.031\\ \tilde{\beta}_1(a^1, b^1) &= 0.642.\end{aligned}$$

This approximation is fairly close to the exact marginal over A and B. ■

The problem in this example is that the approximation of $\tilde{\delta}_{1\to 2}$ is done blindly, not taking into account its effect on approximations on computations in downstream clusters. Thus, the message did not place sufficient emphasis on obtaining a correct approximation for the case $A = a^1$, which roughly corresponds to the "important" (high-probability) case $C = c^1$.

11.4.3.2 Belief Update Propagation

Example 11.7 shows a case where factorizing messages leads to a big error in the approximated beliefs. Let us examine this example in somewhat more detail. Given the update of $\tilde{\delta}_{2\to 1}$, the approximation of $P(A, B)$ is more or less on target. Can we use this information to improve our approximation of $P(C, D)$? To do this, we would need to revise $\tilde{\delta}_{1\to 2}$. The posterior over A and B informs us that most of the mass of the probability distribution is on a^1, b^1. With this information, we might want to change $\tilde{\delta}_{1\to 2}$ to reflect preferences for a^1 and b^1.

A priori, it appears that this idea is inherently problematic; after all, a key constraint for exact inference is to avoid feedback from $\delta_{2\to 1}$ to $\delta_{1\to 2}$, so as not to double-count evidence. For this reason, we took care, in the sum-product message-passing algorithm, *not* to multiply in the message $\delta_{2\to 1}$ when passing messages from $\boldsymbol{C}_1$ to $\boldsymbol{C}_2$.

However, recall that in section 10.3, we presented the sum-product-divide update rule and showed its equivalence to the sum-product rule. Briefly recapping, we can take the sum-product

update rule:

$$\delta_{i \to j} = \sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \psi_i \left(\prod_{k \in \text{Nb}_i - \{j\}} \delta_{k \to i}, \right)$$

and multiply and divide by $\delta_{j \to i}$, resulting in the rule:

$$\delta_{i \to j} \leftarrow \frac{\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \beta_i}{\delta_{j \to i}}.$$

These two rules are therefore equivalent *in the exact case.* However, when we consider approximate inference, the situation is more complex.

belief-update expectation propagation

Consider now performing *belief-update expectation propagation* message passing. Assume, as before, that the algorithm is maintaining a set of approximate beliefs $\tilde{\beta}_i$. We now have two possible stages in which to do the project. The first is:

$$\tilde{\delta}_{i \to j} \leftarrow \text{M-project-distr}_{i,j} \left(\frac{\sum_{\boldsymbol{C}_i - \boldsymbol{S}_{i,j}} \tilde{\beta}_i}{\tilde{\delta}_{j \to i}} \right).$$

This version is identical to the approximate sum-product we discussed before: The beliefs factor β_i accounts for all of the incoming messages; when we divide by the message $\tilde{\delta}_{j \to i}$ before projecting, we are projecting the product of all the other incoming message, which is precisely the sum-product message.

Alternatively, we can do the projection before we divide by $\tilde{\delta}_{j \to i}$. In this second approach, we first project $\tilde{\beta}_i$, and then divide by $\tilde{\delta}_{j \to i}$:

$$\begin{aligned} \tilde{\sigma}_{i \to j} &\leftarrow \text{M-project-distr}_{i,j}(\tilde{\beta}_i) \\ \tilde{\delta}_{i \to j} &\leftarrow \frac{\tilde{\sigma}_{i \to j}}{\tilde{\delta}_{j \to i}}. \end{aligned} \tag{11.39}$$

In this update, we first collect all messages into $\boldsymbol{C}_i$; we then compute the beliefs about $\boldsymbol{C}_i$ and project this to the required form of the message. As in the exact belief update algorithm, this term accounts for information sent from $\boldsymbol{C}_j$. Note that both $\tilde{\sigma}_{i \to j}$ and $\tilde{\delta}_{j \to i}$ have the same factorization, and hence so does their quotient $\tilde{\delta}_{i \to j}$.

This message, which is subsequently used to update $\boldsymbol{C}_j$, is very different from the sum-product update. This is because the incoming message was used in determining the approximation. **The approximation process is invariably a trade-off, in that a better approximation of some regions of the probability space results in a worse approximation in others. In the belief-update form of message passing, we can take into account the current approximation of the message from the target clique when deciding on our approximation, potentially focusing more of our attention on "more relevant" parts of the space.**

To integrate this update rule into the standard belief-update message passing algorithm, we simply replace BU-Message of algorithm 10.3 with EP-Message, shown in algorithm 11.5. We note that the data structures in this procedure are slightly different from those in the original algorithm. First, we maintain the cluster beliefs implicitly, as a factor set, and consider the product of these factors only as part of the M-projection operation. Second, we do not keep the

Algorithm 11.5 Modified version of BU-Message **that incorporates message projection**

Procedure EP-Message (
  i, // sending clique
  j // receiving clique
)
1 $\tilde{\sigma}_{i \to j} \leftarrow$ M-project-distr$_{\mathcal{Q}_{i,j}}(\vec{\phi}_i)$
2 // marginalize and project the clique over the sepset
3 Remove old $\tilde{\delta}_{i \to j}$ from $\vec{\phi}_j$
4 $\tilde{\delta}_{i \to j} \leftarrow \frac{\tilde{\sigma}_{i \to j}}{\tilde{\delta}_{j \to i}}$
5 // divide by the message from from C_j to C_i
6 Insert new $\tilde{\delta}_{i \to j}$ into $\vec{\phi}_j$

previous message sent over the edge, but rather keep the messages $\tilde{\delta}_{i \to j}$ sent in both directions; this more refined bookkeeping is necessary for dividing the correct message following the approximation. The "EP" in the name of the procedure stands for *expectation propagation*, which is the name for this type of algorithm; the reasons for this name are explained later.

expectation propagation

Example 11.8

To understand the behavior of this algorithm, consider the application of belief-update message propagation to example 11.7. Suppose we initialize all messages to 1 and use updates of the form equation (11.39). We start by propagating a message $\tilde{\sigma}_{1 \to 2} =$ M-project-distr$_{1,2}(\tilde{\beta}_1)$ *from* C_1 *to* C_2. *Because* $\tilde{\delta}_{1 \to 2}$ *at this stage is 1, the resulting update is exactly the one we discussed in example 11.7. If we now perform propagation from* C_2 *to* C_1, *we get the message* $\tilde{\delta}_{2 \to 1}$ *derived in example 11.7, multiplied by a constant (since* $\tilde{\delta}_{1 \to 2}$ *is uniform). At this point, as we discussed, the clique beliefs* β_1 *are a fairly reasonable approximation to the posterior.*

Using the revised update rule, we now project $\tilde{\beta}_1$, *and then divide by* $\tilde{\delta}_{2 \to 1}$:

$$\tilde{\delta}_{1 \to 2} \leftarrow \frac{\text{M-project-distr}_{1,2}(\tilde{\beta}_1)}{\tilde{\delta}_{2 \to 1}}.$$

This quotient, which is thensubsequently used to update C_2, *is very different from the previous update* $\tilde{\delta}_{2 \to 1}$. *Specifically, The marginal* $\tilde{\beta}_1$ *at this stage puts a posterior of* $0.642 + 0.031 = 0.673$ *on* a^1, *and* 0.642 *on* b^1. *To avoid double-counting the contribution of* $\tilde{\delta}_{2 \to 1}$, *we need to divide this marginal by this message. After we normalize messages, we obtain:*

$$\tilde{\delta}_{1 \to 2}[a^1] \leftarrow \frac{\frac{0.673}{0.904}}{\frac{0.673}{0.904} + \frac{0.327}{0.086}} = \frac{0.744}{4.15} = 0.179$$

$$\tilde{\delta}_{1 \to 2}[a^0] \leftarrow \frac{\frac{0.327}{0.086}}{\frac{0.673}{0.904} + \frac{0.327}{0.086}} = \frac{3.406}{4.15} = 0.821$$

$$\tilde{\delta}_{1 \to 2}[b^1] \leftarrow \frac{\frac{0.642}{0.173}}{\frac{0.642}{0.173} + \frac{0.358}{0.827}} = \frac{3.710}{4.413} = 0.895$$

$$\tilde{\delta}_{1 \to 2}[b^0] \leftarrow \frac{\frac{0.358}{0.827}}{\frac{0.642}{0.173} + \frac{0.358}{0.827}} = \frac{0.433}{4.144} = 0.105.$$

Recall that this message can be viewed as a "correction term" for the sepset marginals, relative to the message $\tilde{\delta}_{2\to 1}$. Thus, its effect is to reduce the support for a^1 (which was very high in $\tilde{\beta}_2$), and at the same time to increase the support for b^1.

Propagating this message to C_2 and updating the clique beliefs $\tilde{\beta}_2$, we get a normalized factor of:

$$\begin{aligned}
\tilde{\beta}_2(c^0, d^0) &\leftarrow 0.031 \\
\tilde{\beta}_2(c^0, d^1) &\leftarrow 0.397 \\
\tilde{\beta}_2(c^1, d^0) &\leftarrow 0.317 \\
\tilde{\beta}_2(c^1, d^1) &\leftarrow 0.254.
\end{aligned}$$

These beliefs are closer to the exact marginals in that they distribute some of the mass that was previously assigned to c^1, d^0 to two other cases. However, they are still quite far from the exact marginal. ■

This example demonstrates several issues. First, there is significant difference between the two update rules. After incorporating $\tilde{\delta}_{2\to 1}$, the message from C_1 to C_2 is readjusted to account for the new information, leading to a different approximation and hence a different update $\tilde{\delta}_{1\to 2}$. Second, unlike sum-product propagation, belief update propagation does not generally converge within two rounds, even in a clique tree. In fact, an immediate question is whether these iterations converge at all. And if they do, is there anything we can say about the convergence points? As we show in section 11.4.5, the answers to these questions are very similar to the answers we got in the case of cluster-graph belief propagation.

11.4.4 Expectation Propagation

So far, our discussion of approximate message passing has focused on a particular type of approximation: approximating a complex joint distribution as a product of small factors. However, the same ideas are applicable to a broad range of approximations. We now consider this process from a more general perspective, which will allow us to use a wider range of structured approximation in messages. Moreover, as we will see, this generalized form simplifies the variational analysis of this approach.

exponential family

The framework we use is based on the idea of the *exponential family*, as presented in chapter 8. As we discussed there, the exponential families are a general class of distributions, that contains many of the distributions of interest. Recall that a family of distributions $\mathcal{Q}$ is in the exponential family if it can be defined by two functions: the sufficient statistic function $\tau(\boldsymbol{x})$, and the natural parameter function $\mathrm{t}(\boldsymbol{\theta})$, so that any distribution Q in the family can be written as

$$Q(\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\{\langle \tau(\boldsymbol{x}), \mathrm{t}(\boldsymbol{\theta}) \rangle\},$$

where $\boldsymbol{\theta}$ is a set of parameters that specify the particular member of the family.

To simplify the discussion we will focus on linear exponential families, where $\mathrm{t}(\boldsymbol{\theta}) = \boldsymbol{\theta}$. Recall that linear exponential families include Markov networks (and consequently chordal Bayesian networks).

exponential family messages

As we now show, the approximate message passing approach described earlier applies when

Algorithm 11.6 The message passing step in the expectation propagation algorithm. The algorithm performs approximate message propagation by projecting expected sufficient statistics.

Procedure M-Project-Distr (
  $\mathcal{Q}$, // target exponential family for projection
  $\vec{\phi}$ // Factor set
)
1 $\boldsymbol{X} \leftarrow Scope[\vec{\phi}]$ // Variables in factor set
2 $\bar{\tau} \leftarrow \boldsymbol{E}_{\boldsymbol{x} \sim \prod_{\phi \in \vec{\phi}}} \left[\tau_{\mathcal{Q}_{i,j}}(\boldsymbol{x}) \right]$
3 // Compute expectation of sufficient statistics relative to distribution defined by product of factors
4 $\theta \leftarrow$ M-project($\bar{\tau}$)
5 **return** (θ)

we choose to approximate messages by distributions from (linear) exponential families. Specifically, assume that we restrict each sepset $\boldsymbol{S}_{i,j}$ to be represented within an exponential family $\mathcal{Q}_{i,j}$ defined by a sufficient statistics function $\tau_{i,j}$. When performing message passing from $\boldsymbol{C}_i$ to $\boldsymbol{C}_j$, we compute the marginal of $\tilde{\beta}_i$, usually represented as a factor set $\vec{\phi}_i$, and project it into $\mathcal{Q}_{i,j}$ using the *M-projection* operator M-project-distr$_{i,j}$. This computation is often done using inference procedure that takes into account the structure of $\tilde{\beta}_i$ as a factor set.

M-projection

It turns out that the entire message passing operation can be formulated cleanly within this framework. If we are using an exponential family to represent our messages, then both the approximate clique marginal $\tilde{\sigma}_{i \to j}$ and the previous message $\tilde{\delta}_{j \to i}$ can be represented in the exponential form. Thus, if we ignore normalization factors, we have:

$$\begin{aligned}
\tilde{\sigma}_{i \to j} &\propto \exp\left\{ \langle \theta_{\tilde{\sigma}_{i \to j}}, \tau_{i,j}(\boldsymbol{s}_{i,j}) \rangle \right\} \\
\tilde{\delta}_{j \to i} &\propto \exp\left\{ \langle \theta_{\tilde{\delta}_{j \to i}}, \tau_{i,j}(\boldsymbol{s}_{i,j}) \rangle \right\} \\
\tilde{\delta}_{i \to j} &= \frac{\tilde{\sigma}_{i \to j}}{\tilde{\delta}_{j \to i}} \propto \exp\left\{ \langle (\theta_{\tilde{\sigma}_{i \to j}} - \theta_{\tilde{\delta}_{j \to i}}), \tau_{i,j}(\boldsymbol{s}_{i,j}) \rangle \right\},
\end{aligned}$$

where $\theta_{\tilde{\sigma}_{i \to j}}$ and $\theta_{\tilde{\delta}_{i \to j}}$ are the parameters of the messages $\tilde{\sigma}_{i \to j}$ and $\tilde{\delta}_{i \to j}$ respectively.

Since these messages are in an exponential family, it suffices to represent each of them by the parameters that describe them. We can then view propagation steps as updating these parameters. Specifically, we can rewrite the update step in line 4 of EP-Message as

$$\theta_{\tilde{\delta}_{i \to j}} \leftarrow (\theta_{\tilde{\sigma}_{i \to j}} - \theta_{\tilde{\delta}_{j \to i}}). \tag{11.40}$$

Note that, in the case of exact inference in discrete networks, the original update and the one using the exponential family representation are essentially identical, since the exponential family representation of factors is of the same size as the factor. Indeed, the standard update is often performed in a logarithmic representation (for reasons of numerical stability; see box 10.A), which gives rise precisely to the exponential family update.

The final issue we must address is the construction of the exponential-family representation of $\tilde{\sigma}_{i \to j}$ in line 1 of the algorithm. Recall that this process involves the M-projection of $\tilde{\beta}_i$

onto $\mathcal{Q}_{i,j}$. As we discussed in section 8.5, the M-projection of a distribution P within an exponential family $\mathcal{Q}$ is the distribution $Q \in \mathcal{Q}$, which defines the same expectation over the sufficient statistics as defined by P. In that section, we described a two-phase procedure for computing this approximating distribution: We compute the expected sufficient statistics induced by P, and then find the parameters for a distribution $Q \in \mathcal{Q}$ that induces the same expected sufficient statistics. We can apply this approach to define a general procedure for performing the operation M-project-distr$_{i,j}(\vec{\phi}_i)$ for a general exponential family. We first compute the expected expectation of $\tau_{i,j}$ according to $\tilde{\beta}_i$. We then find the distribution within $\mathcal{Q}_{i,j}$ that gives rise to the same expected sufficient statistics. This step is accomplished by the application of the function M-project that takes a vector of sufficient statistics and returns a parameter vector in the exponential family that induces precisely the expected sufficient statistics $\bar{\tau}_{i,j}$. This function is shown in algorithm 11.6. The use of the expectation step in computing the messages is the basis for the name *expectation propagation*, which describes the general procedure for any member of the exponential family.

expectation propagation

As we have already discussed, dividing by the previous message corresponds to subtraction of the parameters. Thus, overall, we obtain the following *EP message passing* step:

EP message passing

$$\theta_{\tilde{\delta}_{i \to j}} \leftarrow \text{M-project}_{i,j}(\boldsymbol{E}_{\boldsymbol{S}_{i,j} \sim \tilde{\beta}_i}[\tau_{i,j}(\boldsymbol{S}_{i,j})]) - \theta_{\tilde{\delta}_{j \to i}}. \tag{11.41}$$

How expensive are the two key steps in the expectation-propagation message passing procedure? The first step is computing the expectation $\boldsymbol{E}_{\boldsymbol{S}_{i,j} \sim \tilde{\beta}_i}[\tau_{i,j}]$. In the case of discrete networks, this step might be as expensive as the number of possible values of $\boldsymbol{C}_i$. However, as we saw in the previous section, in many cases we can use the structure of the factors that make up $\tilde{\beta}_i$ to perform this expectation much more efficiently. The second step is computing M-project on these factors. For some exponential families, this step is trivial, and can be done using a plug-in formula. In other families, this is a complex problem by itself. We will return to these issues in much greater detail in later chapters (particularly chapter 19 and chapter 20). Usually, when we design an approximation algorithm, we choose an exponential family for which this second step is easy.

The factored distributions we discussed earlier are perhaps the simplest example of an exponential family that we can use in this algorithm. However, other representations also fall into this class.

Example 11.9

Consider using a chain network to approximate each message, as in figure 11.15b. In example 8.16 and exercise 8.6 we showed that this class of distributions is a linear family and constructed the function M-project *for it. Following the derivation, suppose the variables in the sepset are $X_1, \ldots, X_k$ and we want to represent messages using the network structure $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_k$. In example 8.16, we showed that the expected sufficient statistics are summarized in the vector of indicators comprising: $\mathbf{1}\{x_i^j\}$ for $i = 1, \ldots, k$ and $x_i^j \in Val(X_i)$; and $\mathbf{1}\{x_i^j, x_{i+1}^\ell\}$ for $i = 1, \ldots, k$, $x_i^j \in Val(X_i)$, $x_i^\ell \in Val(X_{i+1})$. Once we have the expected value of these statistics, we can reconstruct the distribution $Q(X_{i+1} \mid X_i)$ using the procedure described in exercise 8.6. Given these subprocedures, the remaining propagation steps have been described earlier.*

If we consider a chain approximation to grid network, we can use the chains as in figure 11.13b. In this case, the main cost of a propagation step is the projection. The messages incoming to a cluster consists of univariate beliefs and pairwise beliefs along the column. When combined with

the clique factors, these result in a ladder-like network (shown in figure 11.13b). As we discussed, we can a build a (nested) clique tree for this network that involves clusters of at most three variables. Thus, we can perform inference efficiently on this nested clique tree to compute the expectations on pairwise beliefs in the outgoing column, and then use those expectations to reconstruct parameters.■

The view of the expectation propagation algorithm in these terms allows us to understand the scope of this approach. We can apply this approach to any class of distributions in the linear exponential family for which the computation of expected sufficient statistics and the M-projection operation can be implemented effectively. Note that not every class of distributions we have discussed obeys these two restrictions. In particular, Markov networks are in the linear exponential family, but they do not have an effective M-projection procedure. Thus, a general Markov network structure is not often used to represent messages in the expectation propagation algorithm. Conversely, Bayesian networks are not in the linear exponential family (see section 8.3.2). One might argue that Bayesian networks should be usable, since the M-projection operation can be implemented analytically (see theorem 8.7), allowing the expectation propagation algorithm to be applied effectively.

This argument, however, brings up one important caveat regarding the expectation propagation update rule. In the rule, we subtract two sets of parameters: the parameters associated with the message from j to i are subtracted from the parameters obtained from M-projection operation for C_i. For the classes of distributions that we considered earlier, the space of legal parameters Θ was the entire real space $\mathbb{R}^K$; this space is closed under subtraction, guaranteeing that the result of this update rule is a valid distribution in our space. This property does not hold for every exponential family; in particular, it is not the case for Bayesian networks with immoralities. Thus, we may end up in situations where the resulting parameters do not actually define a legal distribution in our space. We return to this issue when we discuss the application of expectation propagation to Gaussians in section 14.3.3, where it can give rise to severe problems. Thus, the most commonly used class of distributions in the expectation propagation algorithm is the class of low tree-width chordal graphs. Because these graphs are both Bayesian networks and Markov networks, they are both in the linear exponential family and admit an effective M-projection operation. The example of the chain distribution we used earlier falls into this category. In more general cases, these distributions are represented as clique trees, allowing them to be represented using a smaller set of factors: clique beliefs and sepset messages.

11.4.5 Variational Analysis

To define a variational principle for expectation propagation, we take an approach similar to the one we discussed in the context of cluster-graph belief propagation. Again, we consider an approximation Q that consists of a set of pseudo marginals. In fact, we use the same energy functional $\tilde{F}[\tilde{P}_\Phi, Q]$.

The main difference from the case of cluster-graph belief propagation is that, in the current approximation, the cluster tree is *not* calibrated. As we project messages into an approximate form, we no longer ensure that beliefs of neighboring clusters agree on the joint distribution of the variables they share. Instead, we maintain a weaker property that depends on the nature of the approximation. For example, in the case where we use messages that are product of marginal distributions, intuition suggests that neighboring clusters will eventually agree on the marginal distributions of the variables in the sepset.

To gain better insight into the constraints we need, we start with reasoning about properties of convergence points of the algorithm. Suppose we iterate expectation-propagation belief update propagations until convergence. Now, consider two neighboring clusters i and j. Since the algorithm has converged, it follows that further updates do not change the cluster beliefs. Thus, the assignment of the expectation-propagation update rule of equation (11.41) becomes an equality for all clusters. We can then reason that

$$\text{M-project}_{i,j}(\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\tilde{\beta}_i}[\tau_{i,j}]) = \theta_{\tilde{\delta}_{i\to j}} + \theta_{\tilde{\delta}_{j\to i}}.$$

A similar argument holds for the M-projection of the j cluster beliefs, $\text{M-project}_{i,j}(\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\tilde{\beta}_j}[\tau_{i,j}])$. It follows that the projection of the two beliefs onto $\mathcal{Q}_{i,j}$ result in the same distribution.

expectation consistency constraint

This discussion suggests that we can pose the problem as optimizing the same objective as CTree-Optimize, except that we now replace the constraint equation (11.7) with an *expectation consistency constraint*:

$$\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\mu_{i,j}}[\tau_{i,j}] = \boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\beta_j}[\tau_{i,j}]. \tag{11.42}$$

In general, $\tau_{i,j}$ is a vector, and so this equation defines a vector of constraints.

We now can define the optimization problem. It is identical to the optimization problems we already encountered, except that we relax the marginal consistency constraints.

EP-Optimize:

Find $\boldsymbol{Q}$
maximizing $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$
subject to

$$\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\mu_{i,j}}[\tau_{i,j}] = \boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\beta_j}[\tau_{i,j}] \quad \forall (i\text{-}j) \in \mathcal{E}_\mathcal{T} \tag{11.43}$$

$$\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) = 1 \qquad \forall i \in \mathcal{V}_\mathcal{T} \tag{11.44}$$

$$\sum_{\boldsymbol{s}_{i,j}} \mu_{i,j}[\boldsymbol{s}_{i,j}] = 1 \qquad \forall (i\text{-}j) \in \mathcal{E}_\mathcal{T} \tag{11.45}$$

$$\beta_i(\boldsymbol{c}_i) \geq 0 \qquad \forall i \in \mathcal{V}_\mathcal{T}, \boldsymbol{c}_i \in Val(\boldsymbol{C}_i). \tag{11.46}$$

Like CGraph-Optimize, this optimization problem is an approximation to the problem of optimizing the energy functional in two ways. First, the space over which we are optimizing is the set of pseudo-marginals $\boldsymbol{Q}$. Because our marginalization constraint only requires that two neighboring clusters agree on their expectations, they will generally not agree on the full marginals. Thus, in general, a solution to this problem will generally *not* correspond to the marginals of an actual distribution Q, even in the context of a clique tree. Second, because we can define the true energy function $F[\tilde{P}_\Phi, Q]$ only for coherent joint distributions, we must resort here to optimizing its factored form $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$. Although the two forms are equivalent for distributions over clique trees, they are not equivalent in this setting (since the exact energy functional is not even defined outside the space of coherent distributions Q). Thus, as in the case of the CGBP algorithms in the previous section, we are approximating both the objective and the optimization space.

The generalization of theorem 11.3 for this relaxed optimization problem follows using more or less the same proof structure, where we characterize the stationary points of the constrained objective using a set of *fixed-point equations* that define a message passing algorithm.

fixed-point equations

Theorem 11.8

Let $\boldsymbol{Q}$ be a set of beliefs such that $\mu_{i,j}$ is in the exponential family $\mathcal{Q}_{i,j}$ for all $(i-j) \in \mathcal{E}_T$. Let M-project-distr$_{i,j}$ *be the M-projection operation into the family $\mathcal{Q}_{i,j}$. Then $\boldsymbol{Q}$ is a stationary point of* EP-Optimize *if and only if for every edge $(i-j) \in \mathcal{E}_T$ there are auxiliary beliefs $\delta_{i\to j}(\boldsymbol{S}_{i,j})$ and $\delta_{j\to i}(\boldsymbol{S}_{i,j})$ so that*

$$\begin{aligned} \delta_{i\to j} &= \frac{\text{M-project-distr}_{i,j}(\beta_i)}{\delta_{j\to i}} \\ \beta_i &\propto \psi_i \cdot \prod_{j\in \text{Nb}_i} \delta_{j\to i} \\ \mu_{i,j} &\propto \delta_{j\to i} \cdot \delta_{i\to j}. \end{aligned} \tag{11.47}$$

Proof As in previous proofs of this kind, we define a Lagrangian that consists of the (approximate) energy functional $\tilde{F}[\tilde{P}_\Phi, \boldsymbol{Q}]$ as well as Lagrange multiplier terms for each of the constraints. In our case, we have a vector of Lagrange multipliers for each of the constraints in equation (11.43).

$$\begin{aligned} \mathcal{J} = &\sum_{i\in\mathcal{V}_T} \boldsymbol{E}_{\boldsymbol{C}_i\sim\beta_i}[\ln\psi_i] + \sum_{i\in\mathcal{V}_T} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(i-j)\in\mathcal{E}_T} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}) \\ &- \sum_i \sum_{j\in\text{Nb}_i} \vec{\lambda}_{j\to i} \cdot \left(\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\mu_{i,j}}[\tau_{i,j}] - \boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\beta_j}[\tau_{i,j}]\right) \\ &- \sum_{i\in\mathcal{V}_T} \lambda_i \left(\sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) - 1\right) - \sum_{(i-j)\in\mathcal{E}_T} \lambda_{i,j} \left(\sum_{\boldsymbol{s}_{i,j}} \mu_{i,j}[\boldsymbol{s}_{i,j}] - 1\right). \end{aligned}$$

Taking partial derivatives of J with respect to $\beta_i(\boldsymbol{c}_i)$ and $\mu_{i,j}[\boldsymbol{s}_{i,j}]$ and equating these derivatives to zero, we get the following equalities that must hold at a stationary point:

$$\begin{aligned} \beta_i(\boldsymbol{c}_i) &\propto \psi_i(\boldsymbol{c}_i) \prod_{j\in\text{Nb}_i} \exp\left\{\vec{\lambda}_{j\to i} \cdot \tau_{i,j}(\boldsymbol{s}_{i,j})\right\} \\ \mu_{i,j}[\boldsymbol{s}_{i,j}] &\propto \exp\left\{(\vec{\lambda}_{j\to i} + \vec{\lambda}_{i\to j}) \cdot \tau_{i,j}(\boldsymbol{s}_{i,j})\right\}. \end{aligned}$$

Note that $(\vec{\lambda}_{j\to i} + \vec{\lambda}_{i\to j})$ serves as the natural parameters of $\mu_{i,j}$ in its exponential form representation.

Moreover, the constraint of equation (11.43) implies that

$$\boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\mu_{i,j}}[\tau_{i,j}] = \boldsymbol{E}_{\boldsymbol{S}_{i,j}\sim\beta_j}[\tau_{i,j}].$$

Thus, using theorem 8.6, we conclude that $\mu_{i,j} = \text{M-project-distr}_{i,j}(\beta_i)$. By defining

$$\delta_{i\to j}[\boldsymbol{s}_{i,j}] \propto \exp\left\{\vec{\lambda}_{i\to j} \cdot \tau_{i,}(\boldsymbol{s}_{i,j})\right\},$$

we can verify that the statement of the theorem is satisfied. ∎

Theorem 11.8 shows that, if we perform EP belief update propagation until convergence, then we reach a stationary point of EP-Optimize. Thus, this result provides an optimization semantics for expectation-propagation message passing.

Our discussion of expectation propagation and the proof were posed in the context of linear exponential families. The same ideas can be extended to nonlinear families but require additional subtleties that we do not discuss.

11.4.6 Discussion

In this section, we presented an alternative approach for inference in large graphical models. Rather than modifying the global structure of the inference object, we modify the structure of the messages and how they are computed. Although we focused on the application of this approximation to clique trees, it is equally applicable in the context of cluster graphs. The message passing algorithms (both the sum-product algorithm and the belief update algorithm) can be used for passing messages between clusters in general graphs. Moreover, the variational analysis of section 11.4.5 also applies, essentially without change, to cluster graphs, using the same derivation as in section 11.3.

Note that the expectation propagation algorithm suffers from the same caveats we discussed in the previous section: **Iterations of EP message propagation are not guaranteed to induce monotonic improvements to the objective function, and the algorithm does not always converge. Moreover, even when the algorithm does converge, the clusters are only approximately calibrated: their marginals agree on the expectations of the sufficient statistics (say the individual marginals), but not on other aspects of the distribution (say marginals over pairs of variables). As a consequence, if we want to answer a query using the network, it may make a difference from which cluster we extract the answer.**

We presented expectation propagation in the context of its application to factored messages. In the simplest case, of fully factored messages, the messages are simply cluster marginals over individual variables. The similarity between this variant of expectation propagation and belief propagation is quite striking; indeed, one can simulate expectation propagation with fully factored messages using cluster-graph belief propagation with a particular factor graph structure (see exercise 11.23). The more general case of messages that are not fully factored (for example, figure 11.13b) is more complex, and they cannot be mapped directly to belief propagation in cluster graphs. However, a mapping does exist between expectation propagation in discrete networks with factorized messages and cluster-graph belief propagation with region graphs.

More important, however, is the fact that expectation propagation provides a general approach for dealing with distributions in the exponential family. It therefore provides message passing algorithms for a broad class of models. For example, we will see an application of expectation propagation to hybrid (continuous/discrete) graphical models in section 14.3.3. Its broad applicability makes expectation propagation an important component in the approximate inference toolbox.

11.5 Structured Variational Approximations

In the previous two sections, we examined approximations based on belief propagation. As we saw, both methods can be viewed as optimizing an approximate energy functional over the

class of pseudo-marginals. These pseudo-marginals generally do not correspond to a globally coherent joint distribution Q. **The *structured variational* approach aims to optimize the energy functional over a family $\mathcal{Q}$ of *coherent* distributions Q. This family is chosen to be computationally tractable, and hence it is generally not sufficiently expressive to capture all of the information in P_Φ.**

structured variational

More precisely, we aim to address the following maximization problem:

Structured-Variational:
Find $Q \in \mathcal{Q}$
maximizing $F[\tilde{P}_\Phi, Q]$

where $\mathcal{Q}$ is a given family of distributions. In these methods, we are using the exact energy functional $F[\tilde{P}_\Phi, Q]$, which satisfies proposition 11.2. Thus, maximizing the energy functional corresponds directly to obtaining a better approximation to P_Φ (in terms of $\boldsymbol{D}(Q\|P_\Phi)$).

The main parameter in this maximization problem is the choice of family $\mathcal{Q}$. This choice induces a trade-off. On one hand, families that are "simpler," that is, that can be described by a Bayesian network or a Markov network with small tree-width, allow more efficient inference. As we will see, simpler families also allow us to solve the maximization problem efficiently. On the other hand, if the family $\mathcal{Q}$ is too restrictive, then it cannot represent distributions that are good approximations of P_Φ, giving rise to a poor approximation Q. In either case, this family is generally chosen to have enough structure that allows inference to be tractable, giving rise to the name *structured variational* approximation.

structured variational

As we will see, the methods of this type differ from generalized belief propagation in several ways. They are guaranteed to lower-bound the log-partition function, and they also are guaranteed to converge.

11.5.1 The Mean Field Approximation

mean field

The first approach we consider is called the *mean field* approximation. As we will see, in many respects, it resembles the algorithm obtained using the Bethe approximation to the energy functional. In particular, the resulting algorithm performs message passing where the messages are distributions over single variables. As we will see, however, the form of the updates is somewhat different.

11.5.1.1 The Mean Field Energy

Unlike our presentation in earlier sections, we begin our discussion with the energy functional, and we derive the algorithm directly from analyzing it. The mean field algorithm finds the distribution Q, which is closest to P_Φ in terms of $\boldsymbol{D}(Q\|P_\Phi)$ within the class of distributions representable as a product of independent marginals:

$$Q(\mathcal{X}) = \prod_i Q(X_i). \tag{11.48}$$

On the one hand, the approximation of P_Φ as a fully factored distribution is likely to lose a lot of information in the distribution. On the other hand, this approximation is computationally

attractive, since we can easily evaluate any query on Q by a product over terms that involve the variables in the scope of the query. Moreover, to represent Q, we need only to describe the marginal probabilities of each of the variables.

energy functional

As in previous sections, the mean field algorithm is derived by considering fixed points of the *energy functional*. We thus begin by considering the form of the energy functional in equation (11.3) when Q has the form of a product distribution as in equation (11.48). We can then characterize its fixed points and thereby derive an iterative algorithm to find such fixed points.

The functional contains two terms. The first is a sum of terms of the form $\boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\ln \phi]$, where we need to evaluate

$$\begin{aligned} \boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\ln \phi] &= \sum_{\boldsymbol{u}_\phi} Q(\boldsymbol{u}_\phi) \ln \phi(\boldsymbol{u}_\phi) \\ &= \sum_{\boldsymbol{u}_\phi} \left(\prod_{X_i \in \boldsymbol{U}_\phi} Q(x_i) \right) \ln \phi(\boldsymbol{u}_\phi). \end{aligned}$$

As shown, we can use the form of Q to compute $Q(\boldsymbol{u}_\phi)$ as a product of marginals, allowing the evaluation of this term to be performed in time linear in the number of values of $\boldsymbol{U}_\phi$. Because this cost is linear in the description size of the factors of P_Φ, we cannot expect to do much better.

As we saw in section 8.4.1, the term $\boldsymbol{H}_Q(\mathcal{X})$ also decomposes in this case.

Corollary 11.3

If $Q(\mathcal{X}) = \prod_i Q(X_i)$, then

$$\boldsymbol{H}_Q(\mathcal{X}) = \sum_i \boldsymbol{H}_Q(X_i). \tag{11.49}$$

Thus, the energy functional for a fully factored distribution Q can be rewritten simply as a sum of expectations, each one over a small set of variables. Importantly, the complexity of this expression depends on the size of the factors in P_Φ, and *not* on the topology of the network. Thus, the energy functional in this case can be represented and manipulated effectively, even in networks that would require exponential time for exact inference.

Example 11.10

Continuing our running example, consider the form of the mean field energy for a 4×4 grid network. Based on our discussion, we see that it has the form

$$\begin{aligned} F[\tilde{P}_\Phi, Q] = &\sum_{i \in \{1,2,3\}, j \in \{1,2,3,4\}} \boldsymbol{E}_Q[\ln \phi(A_{i,j}, A_{i+1,j})] \\ &+ \sum_{i \in \{1,2,3,4\}, j \in \{1,2,3\}} \boldsymbol{E}_Q[\ln \phi(A_{i,j}, A_{i,j+1})] \\ &+ \sum_{i \in \{1,2,3,4\}, j \in \{1,2,3,4\}} \boldsymbol{H}_Q(A_{i,j}). \end{aligned}$$

We see that the energy functional involves only expectations over single variables and pairs of neighboring variables. The expression has the same general form for an $n \times n$ grid. Thus, although the tree-width of an $n \times n$ grid is exponential in n, the energy functional can be represented and computed in cost $O(n^2)$; that is, in a time linear in the number of variables. ■

11.5.1.2 Maximizing the Energy Functional: Fixed-point Characterization

The next step is to consider the task of optimizing the energy function: finding the distribution Q for which this energy functional is maximized:

Mean-Field:
Find $\{Q(X_i)\}$
maximizing $F[\tilde{P}_\Phi, Q]$
subject to

$$Q(\mathcal{X}) = \prod_i Q(X_i) \tag{11.50}$$

$$\sum_{x_i} Q(x_i) = 1 \quad \forall i. \tag{11.51}$$

To simplify notation, from now on we use $\boldsymbol{X}_{-i}$ to denote $\mathcal{X} - \{X_i\}$.

Note that, unlike the cluster-graph belief propagation algorithms of section 11.3 and the expectation propagation algorithm of section 11.4, here we are not approximating the objective. We are approximating only the optimization space by selecting a space of distributions $\mathcal{Q}$ that generally does not contain our original distribution P_Φ.

As with the previous optimization problems in this chapter, we use the method of Lagrange multipliers to derive a characterization of the stationary points of $F[\tilde{P}_\Phi, Q]$. However, the structure of Q allows us to consider the optimal value of each component (that is, marginal distribution) given the rest. (This iterative optimization procedure was not feasible in cluster trees and graphs due to constraints that relate different beliefs.)

fixed-point equations

We now provide a set of *fixed-point equations* that characterize the stationary points of the mean field optimization problem:

Theorem 11.9 *The distribution $Q(X_i)$ is a local maximum of* Mean-Field *given $\{Q(X_j)\}_{j \neq i}$ if and only if*

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{ \sum_{\phi \in \Phi} \boldsymbol{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i] \right\}, \tag{11.52}$$

conditional expectation

where Z_i is a local normalizing constant and $\boldsymbol{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i]$ is the conditional expectation *given the value x_i*

$$\boldsymbol{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i] = \sum_{\boldsymbol{u}_\phi} Q(\boldsymbol{u}_\phi \mid x_i) \ln(\boldsymbol{u}_\phi).$$

PROOF The proof of this theorem relies on proving the fixed-point characterization of the individual marginal $Q(X_i)$ in terms of the other components, $Q(X_1), \ldots, Q(X_{i-1}), Q(X_{i+1}), \ldots, Q(X_n)$, as specified in equation (11.52).

We first consider the restriction of our objective $F[\tilde{P}_\Phi, Q]$ to those terms that involve $Q(X_i)$:

$$F_i[Q] = \sum_{\phi \in \Phi} \boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\ln \phi] + \boldsymbol{H}_Q(X_i). \tag{11.53}$$

To optimize $Q(X_i)$, we define the Lagrangian that consists of all terms in $F[\tilde{P}_\Phi, Q]$ that involve $Q(X_i)$

$$L_i[Q] = \sum_{\phi \in \Phi} \boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\ln \phi] + \boldsymbol{H}_Q(X_i) + \lambda \left(\sum_{x_i} Q(x_i) - 1 \right).$$

The Lagrange multiplier λ corresponds to the constraint that $Q(X_i)$ is a distribution. We now take derivatives with respect to $Q(x_i)$. The following result plays an important role in the remainder of the derivation:

Lemma 11.1

If $Q(\mathcal{X}) = \prod_i Q(X_i)$ then, for any function f with scope $\boldsymbol{U}$,

$$\frac{\partial}{\partial Q(x_i)} \boldsymbol{E}_{\boldsymbol{U} \sim Q}[f(\boldsymbol{U})] = \boldsymbol{E}_{\boldsymbol{U} \sim Q}[f(\boldsymbol{U}) \mid x_i].$$

The proof of this lemma is left as an exercise (see exercise 11.24).

Using this lemma, and standard derivatives of entropies, we see that

$$\frac{\partial}{\partial Q(x_i)} L_i = \sum_{\phi \in \Phi} \boldsymbol{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i] - \ln Q(x_i) - 1 + \lambda.$$

Setting the derivative to 0, and rearranging terms, we get that

$$\ln Q(x_i) = \lambda - 1 + \sum_{\phi \in \Phi} \boldsymbol{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i].$$

We take exponents of both sides and renormalize; because λ is constant relative to x_i, it drops out in the renormalization, so that we obtain the formula in equation (11.52).

This derivation, by itself, shows only that the solution of equation (11.52) is a stationary point of equation (11.53). To prove that it is a maximum, we note that equation (11.53) is a sum of two terms: $\sum_{\phi \in \Phi} \boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\ln \phi]$ is linear in $Q(X_i)$, given all the other components $Q(X_j)$; $\boldsymbol{H}_Q(X_i)$ is a concave function in $Q(X_i)$. As a whole, given the other components of Q, the function F_i is concave in $Q(X_i)$, and therefore has a unique global optimum, which is easily verified to be equation (11.52) rather than any of the extremal points. ■

From this it follows that:

Corollary 11.4

The distribution Q is a stationary point of Mean-Field *if and only if, for each X_i, equation (11.52) holds.*

In contrast to theorem 11.9, this result only provides a characterization of stationary points of the objective, and not necessarily of its optima. The stationary points include local maxima, local minima, and saddle points. The reason for the difference is that, although each "coordinate" $Q(X_i)$ is guaranteed to be locally maximal given the others, the direction that locally improves the objective may require a coordinated change in several components. We return to this point in section 11.5.1.3.

We now move to interpreting this characterization. The key term in equation (11.52) is the argument in the expectation. We can prove the following property.

Corollary 11.5

In the mean field approximation, $Q(X_i)$ is locally optimal only if

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{\mathbf{E}_{\mathbf{X}_{-i} \sim Q}[\ln P_\Phi(x_i \mid \mathbf{X}_{-i})]\right\} \tag{11.54}$$

where Z_i is a normalizing constant.

PROOF Recall that $\tilde{P}_\Phi = \prod_{\phi \in \Phi} \phi$ is the unnormalized measure defined by Φ. Due to the linearity of expectation:

$$\sum_{\phi \in \Phi} \mathbf{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i] = \mathbf{E}_{\mathcal{X} \sim Q}\left[\ln \tilde{P}_\Phi(X_i, \mathbf{X}_{-i}) \mid x_i\right].$$

Because Q is a product of marginals, we can rewrite $Q(\mathbf{X}_{-i} \mid x_i) = Q(\mathbf{X}_{-i})$, and get that:

$$\mathbf{E}_{\mathcal{X} \sim Q}\left[\ln \tilde{P}_\Phi(X_i, \mathbf{X}_{-i}) \mid x_i\right] = \mathbf{E}_{\mathbf{X}_{-i} \sim Q}\left[\ln \tilde{P}_\Phi(x_i, \mathbf{X}_{-i})\right].$$

Using properties of conditional distributions, it follows that:

$$\tilde{P}_\Phi(x_i, \mathbf{X}_{-i}) = Z P_\Phi(x_i, \mathbf{X}_{-i}) = Z P_\Phi(\mathbf{X}_{-i}) P_\Phi(x_i \mid \mathbf{X}_{-i}).$$

We conclude that

$$\sum_{\phi \in \Phi} \mathbf{E}_{\mathcal{X} \sim Q}[\ln \phi \mid x_i] = \mathbf{E}_{\mathbf{X}_{-i} \sim Q}[\ln P_\Phi(x_i \mid \mathbf{X}_{-i})] + \mathbf{E}_{\mathbf{X}_{-i} \sim Q}[\ln P_\Phi(\mathbf{X}_{-i}) Z].$$

Plugging this equality into the update equation equation (11.52), we get that

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{\mathbf{E}_{\mathbf{X}_{-i} \sim Q}[\ln P_\Phi(x_i \mid \mathbf{X}_{-i})]\right\} \exp\left\{\mathbf{E}_{\mathbf{X}_{-i} \sim Q}[\ln P_\Phi(\mathbf{X}_{-i}) Z]\right\}.$$

The term $\ln P_\Phi(\mathbf{X}_{-i}) Z$ does not depend on the value of x_i. Recall that when we multiply a belief by a constant factor, it does not change the distribution Q; in fact, as we renormalize the distribution at the end to sum to 1, this constant will be "absorbed" into the normalizing function, to achieve normalization. Thus, we can simply ignore this term, thereby achieving the desired conclusion. We note that this type of algebraic manipulation will prove useful multiple times throughout this section. ■

This corollary shows that $Q(x_i)$ is the *geometric average* of the conditional probability of x_i given all other variables in the domain. The average is based on the probability that Q assigns to all possible assignments to the variables in the domain. In this sense, the mean field approximation requires that the marginal of X_i be "consistent" with the marginals of other variables.

Note that, in P_Φ, we can also represent the marginal of X_i as an average:

$$P_\Phi(x_i) = \sum_{\boldsymbol{x}_{-i}} P_\Phi(\boldsymbol{x}_{-i}) P_\Phi(x_i \mid \boldsymbol{x}_{-i}) = \mathbf{E}_{\mathbf{X}_{-i} \sim P_\Phi}[P_\Phi(x_i \mid \mathbf{X}_{-i})]. \tag{11.55}$$

This average is an arithmetic average, whereas the one used in the mean field approximation is a geometric average. In general, the latter tends to lead to marginals that are more sharply peaked than the original marginals in P_Φ. More significant, however, is the fact that the expectations in equation (11.55) are taken relative to P_Φ, whereas the ones in equation (11.54) are taken relative to the approximation Q. Thus, this similarity does not imply as a consequence that our approximation in Q to the marginals in P_Φ is a good one.

11.5.1.3 Maximizing the Energy Functional: The Mean Field Algorithm

How do we convert the fixed-point equation of equation (11.52) into an update algorithm? We start by observing that if $X_i \notin Scope[\phi]$ then $\mathbf{E}_{\mathbf{U}_\phi \sim Q}[\ln \phi \mid x_i] = \mathbf{E}_{\mathbf{U}_\phi \sim Q}[\ln \phi]$. Thus, expectation terms on such factors are independent of the value of X_i. Consequently, we can absorb them into the normalization constant Z_i and get the following simplification.

Corollary 11.6

In the mean field approximation, $Q(X_i)$ is locally optimal only if

$$Q(x_i) = \frac{1}{Z_i} \exp\left\{ \sum_{\phi: X_i \in Scope[\phi]} \mathbf{E}_{(\mathbf{U}_\phi - \{X_i\}) \sim Q}[\ln \phi(\mathbf{U}_\phi, x_i)] \right\}. \tag{11.56}$$

where Z_i is a normalizing constant.

This representation shows that $Q(X_i)$ has to be consistent with the expectation of the potentials in which it appears. In our grid network example, this characterization implies that $Q(A_{i,j})$ is a product of four terms measuring its interaction with each of its four neighbors:

$$Q(a_{i,j}) = \frac{1}{Z_{i,j}} \exp\left\{ \begin{array}{l} \sum_{a_{i-1,j}} Q(a_{i-1,j}) \ln(\phi(a_{i-1,j}, a_{i,j})) + \\ \sum_{a_{i,j-1}} Q(a_{i,j-1}) \ln(\phi(a_{i,j-1}, a_{i,j})) + \\ \sum_{a_{i+1,j}} Q(a_{i+1,j}) \ln(\phi(a_{i,j}, a_{i+1,j})) + \\ \sum_{a_{i,j+1}} Q(a_{i,j+1}) \ln(\phi(a_{i,j}, a_{i,j+1})) \end{array} \right\}. \tag{11.57}$$

Each term is a (geometric) average of one of the potentials involving $A_{i,j}$. For example, the final term in the exponent represents a geometric average of the potential between $A_{i,j}$ and $A_{i,j+1}$, averaged using the distribution $Q(A_{i,j+1})$.

The characterization of corollary 11.6 provides tools for developing an algorithm to maximize $F[\tilde{P}_\Phi, Q]$. For example, examining equation (11.57), we see that we can easily evaluate the term within the exponential by considering each of $A_{i,j}$'s neighbors and computing the interaction between the values that neighbor can take and possible values of $A_{i,j}$. Moreover, in this example, we see that $Q(A_{i,j})$ does *not* appear on the right-hand side of the update rule. Thus, we can choose $Q(A_{i,j})$, which satisfies the required equality by assigning it to the term denoted by the right-hand side of the equation.

This last observation is true in general. All the terms on the right-hand side of equation (11.56) involve expectations of variables other than X_i, and do not depend on the choice of $Q(X_i)$. We can achieve equality simply by evaluating the exponential terms for each value x_i, normalizing the results to sum to 1, and then assigning them to $Q(X_i)$. As a consequence, we reach the optimal value of $Q(X_i)$ in one easy step.

This last statement must be interpreted with some care. The resulting value for $Q(X_i)$ is its optimal value *given* the choice of all other marginals. Thus, this step optimizes our function relative only to a single coordinate in the space — the marginal of $Q(X_i)$. To optimize the function in its entirety, we need to optimize relative to all of the coordinates. We can embed this step in an iterated *coordinate ascent* algorithm, which repeatedly optimizes a single marginal at a time, given fixed choices to all of the others. The resulting algorithm is shown in algorithm 11.7. Importantly, a single optimization of $Q(X_i)$ does not usually suffice: a subsequent modification

Algorithm 11.7 The Mean-Field approximation algorithm

```
Procedure Mean-Field (
  Φ,    // factors that define P_Φ
  Q_0   // Initial choice of Q
)
1   Q ← Q_0
2   Unprocessed ← X
3   while Unprocessed ≠ ∅
4     Choose X_i from Unprocessed
5     Q_old(X_i) ← Q(X_i)
6     for x_i ∈ Val(X_i) do
7       Q(x_i) ← exp{ Σ_{φ:X_i∈Scope[φ]} E_{(U_φ−{X_i})∼Q}[ln φ[x_i]] }
8     Normalize Q(X_i) to sum to one
9     if Q_old(X_i) ≠ Q(X_i) then
10      Unprocessed ← Unprocessed ∪ (∪_{φ:X_i∈Scope[φ]} Scope[φ])
11    Unprocessed ← Unprocessed − {X_i}
12  return Q
```

to another marginal $Q(X_j)$ may result in a different optimal parameterization for $Q(X_i)$. Thus, the algorithm repeats these steps until convergence. Note that, in practice, we do not test for equality in line 9, but rather for equality up to some fixed small-error tolerance.

A key property of the coordinate ascent procedure is that each step leads to an increase in the energy functional. **Thus, each iteration of** Mean-Field **results in a better approximation Q to the target density P_Φ, guaranteeing convergence.**

Theorem 11.10 *The* Mean-Field *iterations are guaranteed to converge. Moreover, the distribution Q^* returned by* Mean-Field *is a stationary point of $F[\tilde{P}_\Phi, Q]$, subject to the constraint that $Q(\mathcal{X}) = \prod_i Q(X_i)$ is a distribution.*

PROOF We showed earlier that each iteration of Mean-Field is monotonically nondecreasing in $F[\tilde{P}_\Phi, Q]$. Because the energy functional is bounded, the sequence of distributions represented by successive iterations of Mean-Field must converge. At the convergence point the fixed-point equations of theorem 11.9 hold for all the variables in the domain. As a consequence, the convergence point is a stationary point of the energy functional. ■

As we discussed, the distribution Q^* returned by Mean-Field is not necessarily a local optimum of the algorithm. However, local minima and saddle points are not stable convergence points of the algorithm, in the sense that a small perturbation of Q followed by optimization will lead to a better convergence point. Because the algorithm is unlikely to accidentally land precisely on the unstable point and get stuck there, in practice, the convergence points of the algorithm are local maxima.

In general, however, the result of the mean field approximation is a local maximum, and not necessarily a global one.

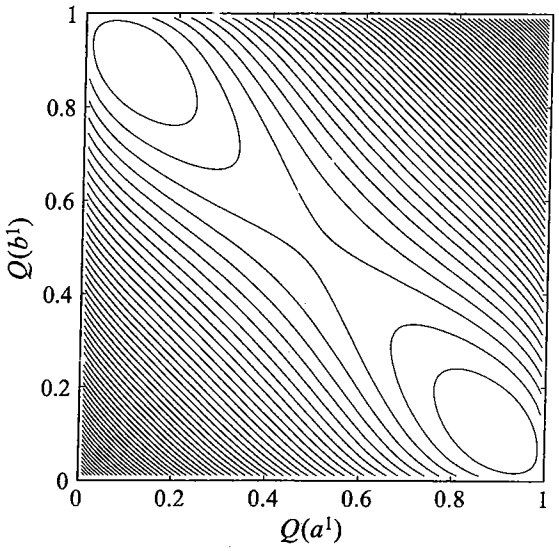


Figure 11.16 An example of a multimodal mean field energy functional landscape. In this network, $P(a, b) = 0.5 - \epsilon$ if $a \neq b$ and ϵ if $a = b$. The axes correspond to the mean field marginal for $Q(a^1)$ and $Q(b^1)$ and the contours show equi-values of the energy functional for different choices of these variational parameters. As we can see, there are two modes to the energy functional, one roughly corresponds to a^1, b^0 and the other one to a^0, b^1. In addition, there is a saddle point at $(0.5, 0.5)$.

Example 11.11

Consider a distribution P_Φ that is an approximate XOR (exclusive or) of two variables A and B, so that $P_\Phi(a, b) = 0.5 - \epsilon$ if $a \neq b$ and $P_\Phi(a, b) = \epsilon$ if $a = b$. Clearly, we cannot approximate P_Φ by a product of marginals, since such a product cannot capture the relationship between A and B. It turns out that if ϵ is sufficiently small, say 0.01, then the energy potential surface has two local maxima that correspond to the two cases where $a \neq b$. See figure 11.16. (For sufficiently large ϵ, such as 0.1, the mean field approximation has a single maximum point at the uniform distribution.) ■

We can use standard strategies, such as multiple random restarts, to try to avoid getting stuck in local maxima. However, these do not overcome the basic shortcoming of the mean field approximation, which is apparent in this example. The approximation cannot describe complex posteriors, such as the XOR posterior we discussed. And thus, we cannot expect it to give satisfactory approximations in these situations. To provide better approximations, we must use a richer class of distributions $\mathcal{Q}$, which has greater expressive power.

11.5.2 Structured Approximations

The mean field algorithm provides an easy approximation method. However, it is limited by forcing Q to be a very simple distribution. As we just saw, the fact that all variables are independent of each other in Q can lead to very poor approximations. Intuitively, if we

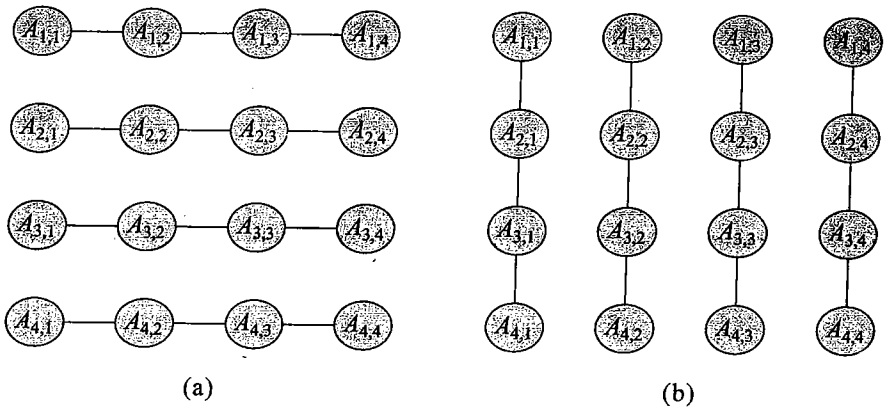


Figure 11.17 Two structures for variational approximation of a 4×4 grid network

use a distribution Q that can capture some of the dependencies in P_Φ, we can get a better approximation. Thus, we would like to explore the spectrum of approximations between the mean field approximation and exact inference.

A natural approach to get richer approximations that capture some of the dependencies in P_Φ is to use network structures of different complexity. By adding and removing edges from the network we can control the cost of inference in the approximating distribution and how well it captures dependencies in the target distribution. We can achieve this type of flexibility by using either Bayesian networks or Markov networks. Both types of networks lead to similar approximations, and so we focus on the undirected case, parameterized as Gibbs distributions (so that we are not restricted to factors over maximal cliques). Exercise 11.34 develops similar ideas using a Bayesian network approximation.

11.5.2.1 Fixed-Point Characterization

We now consider the form of the variational approximation when we are given a general form of Q as a Gibbs parametric family. Formally, we assume we are given a set of potential scopes $\{\boldsymbol{C}_j \subseteq \mathcal{X} : j = 1, \ldots, J\}$. We can then choose an approximation Q that has the form:

$$Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_{j=1}^{J} \psi_j, \tag{11.58}$$

where ψ_j is a factor with $Scope[\psi_j] = \boldsymbol{C}_j$.

Example 11.12 *Consider again the grid network example. There are many possible approximating network structures we can choose that allow for efficient inference. As a concrete example, we might choose potential scopes* $\{A_{1,1}, A_{1,2}\}$, $\{A_{1,2}, A_{1,3}\}, \ldots, \{A_{2,1}, A_{2,2}\}$, $\{A_{2,2}, A_{2,3}\}, \ldots$. *That is, we preserve the dependencies between variables in the same column, but ignore the ones that relate different columns. Alternatively, we can consider an approximation that preserves dependencies along rows and ignores the dependencies between columns. As we can see in figure 11.17, in both*

cases, the structure we use is a collection of independent chain structures. Exact inference with such structures is linear, and so the cost of inference is not much worse than in the mean field approximation. Clearly, we can also consider many other structures for the approximating distributions. These might introduce additional dependencies and can have higher cost in terms of inference. We will return to the question of what structure to use. ■

Assume that we decide on the form of the potentials for the approximating family $\mathcal{Q}$. As before, we consider the form of the energy functional for a distribution Q in this family. We then characterize the stationary points of the functional, and we use those to derive an iterative optimization algorithm.

As before, evaluating the terms that involve $\mathbf{E}_{U_\phi \sim Q}[\ln \phi]$ requires performing expectations with respect to the variables in $Scope[\phi]$. Unlike the case of mean field approximation, the complexity of computing this expectation depends on the structure of the approximating distribution. However, we assume that we can solve this problem by exact inference (in the network corresponding to Q), using the methods we discussed in previous chapters.

As discussed in section 8.4.1, the entropy term in the energy functional also reduces to computing a similar set of expectation terms:

Proposition 11.5

If $Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j$, *then*

$$\mathbf{H}_Q(\mathcal{X}) = -\sum_{j=1}^{J} \mathbf{E}_{\mathbf{C}_j \sim Q}[\ln \psi_j(\mathbf{C}_j)] + \ln Z_Q.$$

Overall, we obtain the following form for the energy functional, for distributions Q in the family $\mathcal{Q}$:

$$F[\tilde{P}_\Phi, Q] = \sum_{k=1}^{K} \mathbf{E}_Q[\ln \phi_k] - \sum_{j=1}^{J} \mathbf{E}_Q[\ln \psi_j] + \ln Z_Q. \tag{11.59}$$

As before, the hard question is how to optimize the potential to get the best approximation. We solve this problem using the same general strategy we discussed in the context of the mean field approximation. First, we derive the fixed-point equations that hold when the approximation is a local maximum (or, more precisely, a stationary point) of the energy functional. We then use these fixed-point equations to help derive an optimization algorithm.

fixed-point equations

We derive the *fixed-point equations* by taking derivatives of $F[\tilde{P}_\Phi, Q]$ with respect to parameters of the distribution Q. In our case, the parameters will be an entry $\psi_j(\mathbf{c}_j)$ in each of the factors that define the distribution. We then set those equations to zero, obtaining the following result:

Theorem 11.11

If $Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j$, *then the potential* ψ_j *is a stationary point of the energy functional if and only if*

$$\psi_j(\mathbf{c}_j) \propto \exp\left\{ \mathbf{E}_Q\left[\ln \tilde{P}_\Phi \mid \mathbf{c}_j\right] - \sum_{k \neq j} \mathbf{E}_Q[\ln \psi_k \mid \mathbf{c}_j] - F[\tilde{P}_\Phi, Q] \right\}. \tag{11.60}$$

The proof is straightforward algebraic manipulation and is left as an exercise (exercise 11.26).

This theorem establishes a characterization of the fixed point as the difference between the expected value of logarithm of the original potentials and the expected value of the logarithm of the approximating potentials. The last term in equation (11.60) is the energy functional $F[\tilde{P}_\Phi, Q]$, which is independent of the assignment $\boldsymbol{c}_j$; thus, as we discussed in the proof of corollary 11.5, we can absorb this term into the normalization constant of the distribution and ignore it.

Corollary 11.7 *If $Q(\mathcal{X}) = \frac{1}{Z_Q}\prod_j \psi_j$, then the potential ψ_j is a stationary point of the energy functional if and only if:*

$$\psi_j(\boldsymbol{c}_j) \propto \exp\left\{ \boldsymbol{E}_Q\left[\ln \tilde{P}_\Phi \mid \boldsymbol{c}_j\right] - \sum_{k \neq j} \boldsymbol{E}_Q[\ln \psi_k \mid \boldsymbol{c}_j] \right\}. \tag{11.61}$$

As we show in section 11.5.2.3 and section 11.5.2.4, we can often exploit additional structure in Q to reduce further the complexity of the fixed-point equations, and hence of the resulting update steps. The following discussion, which describes the procedure of applying the fixed-point equations to find a stationary point of the energy functional, is orthogonal to these simplifications.

11.5.2.2 Optimization

Given a set of fixed-point equations as in equation (11.61), our task is to find a distribution Q that satisfies them. As in section 11.5.1, our strategy is based on the key observation that the factor ψ_j does not affect the right-hand side of the fixed-point equations defining its value: The first expectation, $\boldsymbol{E}_Q\left[\ln \tilde{P}_\Phi \mid \boldsymbol{c}_j\right]$, is conditioned on $\boldsymbol{c}_j$ and therefore does not depend on the parameterization of ψ_j. The same observation holds for the second expectation, $\boldsymbol{E}_Q[\ln \psi_k \mid \boldsymbol{c}_j]$, for any $k \neq j$. (Importantly, there is no such term for $k = j$ in the right-hand side.) Thus, we can use the same general approach as in Mean-Field: We can optimize each potential ψ_j, *given values for the other potentials*, by simply selecting ψ_j to satisfy the fixed-point equation. As for the case of mean field, this step is guaranteed to increase (or not decrease) the value of the objective; thus, the overall process is guaranteed to converge to a stationary point of the objective.

This last step requires that we perform inference in the approximating distribution Q to compute the requisite expectations. Although this step was also present (implicitly) in the mean field approximation, there the structure of the approximating distribution was trivial, and so the inference step involved only individual marginals. Here, we need to collect the expectation of several factors, and each of these requires that we compute expectations given different assignments to the factor of interest. (See exercise 11.27 for a discussion of how these expectations can be computed efficiently.) For a general distribution Q, even one with tractable structure, running inference in the corresponding network $\mathcal{H}_Q$ can be costly, and we may want to reduce the number of calls to the inference subroutine.

This observation leads to a question of how best to perform updates for several factors in Q. We can consider two strategies. The *sequential update* strategy is similar to our strategy in the mean field algorithm: We choose a factor in ψ_j, apply the fixed-point equation to that

factor by running inference in $\mathcal{H}_Q$, update the distribution, and then repeat this process with another factor until convergence. The problematic aspect of this approach is that we need to perform inference after each update step. For example, if we are using cluster tree inference in the network $\mathcal{H}_Q$, the network parameterization changes after each update step, so we need to recalibrate the clique tree every time. Some of these steps can be made more efficient by selecting an appropriate order of updates and using dynamic programming (see exercise 11.27), but the process can still be quite expensive.

An alternative approach is the *parallel update* strategy, where we compute the right-hand side of our fixed-point equations (for example, equation (11.61)) simultaneously for each of the factors in Q. If we are using a cluster tree for inference, this process involves multiple queries from the same calibrated cluster tree. Thus, we can perform a single calibration step and use the resulting tree to reestimate all of our potentials; this process is less costly than recalibrating the clique tree J times. Using these results, we update all the factors at once, recalibrate the tree, and repeat.

The comparison of these two methods is not as clear-cut as in the comparison of parallel and asynchronous message scheduling for cluster-graph belief propagation (see box 11.B). On one hand, the parallel strategy requires an order of magnitude fewer inference steps in the network $\mathcal{H}_Q$, which can lead to significant savings, especially in cases where the approximation Q is reasonably expressive. Although the cost of the sequential method can be reduced using the dynamic programming approach mentioned earlier, the resulting algorithm is nontrivial to implement, and it is still generally more expensive than a single calibration.

On the other hand, the guarantees provided by these two update steps are different. For the sequential update strategy, we can prove that each update step is monotonic in the energy functional: each step maximizes the value of one potential given the values of all the others, and therefore is guaranteed not to decrease (and generally to increase) the energy functional. This monotonic improvement implies that iterations of sequential updates necessarily converge, generally to a local maximum. The issue of convergence is more complicated in the parallel update strategy. Because we update all the potentials at once, we have no guarantees that any fixed-point equation holds after the update; a value that was optimal for ψ_j with respect to the values of all other factors before the parallel update step is not necessarily optimal given their new values. As such, it is conceivable that parallel updates will not converge (for example, oscillate between two sets of values for the potentials). Such oscillations can generally be avoided using damped update steps, similar to these we discussed in the case of cluster-graph belief propagation (see box 11.B), but this modified procedure still does not guarantee convergence.

At this point, there is no generally accepted procedure for scheduling updates in variational methods, and different approaches are likely to be best for different applications.

11.5.2.3 Simplifying the Update Equations

Equation (11.61) provides a general characterization of the fixed points of the energy functional, for any approximating class of distributions $\mathcal{Q}$ obeying a particular factorization, as in equation (11.58). In many cases, we can exploit additional structure of the approximating class $\mathcal{Q}$ and of the distribution P_Φ to simplify significantly the form of these fixed-point equations and thereby make the update step more efficient.

The simplifications we describe take two forms. The first utilizes marginal independencies in $\mathcal{Q}$ to simplify the right-hand side of the fixed-point equation, equation (11.61), eliminating irrelevant terms. The second exploits interactions between the form of $\mathcal{Q}$ and the form of P_Φ to simplify the factorization of $\mathcal{Q}$, without loss in expressive power. Both simplifications allow the fixed-point updates to be performed more efficiently. We motivate each of the simplifications using an example, and then we present the general result.

Example 11.13

Once again, consider the 4×4 grid network. Assume that we approximate it by a "row" network that has the structure shown in figure 11.17a. This approximating network consists of four independent chains. Now we can apply the general form of the fixed-point equation equation (11.61) for a specific entry in our approximation, say:

$$\psi_{1,1}(a_{1,1}, a_{1,2}) \propto \exp\left\{ \begin{array}{l} \mathbf{E}_Q\left[\ln \tilde{P}_\Phi \mid a_{1,1}, a_{1,2}\right] \\ -\sum_{\substack{i=1,\ldots,4 \\ j=1,\ldots,3 \\ (i,j)\neq(1,1)}} \mathbf{E}_Q\left[\ln \psi_{(i,j)}(A_{i,j}, A_{i,j+1}) \mid a_{1,1}, a_{1,2}\right] \end{array} \right\}.$$

As in the proof of corollary 11.5, the expectation of $\ln \tilde{P}_\Phi$ is the sum of expectations of the logarithm of each of the potentials in Φ. Some of these terms, however, do not depend on the choice of value of $A_{1,1}, A_{1,2}$ we are evaluating. For example, because $A_{2,1}$ and $A_{2,2}$ are independent of $A_{1,1}, A_{1,2}$ in Q, we conclude that

$$\mathbf{E}_{\{A_{2,1},A_{2,2}\}\sim Q}[\ln \phi(A_{2,1}, A_{2,2}) \mid a_{1,1}, a_{1,2}] = \mathbf{E}_{\{A_{2,1},A_{2,2}\}\sim Q}[\ln \phi(A_{2,1}, A_{2,2})].$$

Thus, this term will contribute the same value to each of the entries of $\psi(A_{1,1}, A_{1,2})$, and can therefore be absorbed into the corresponding normalizing term. We can continue in this manner and remove all terms that are not dependent on the context of the factor we are interested in. Overall, we can remove any term $\mathbf{E}_Q[\ln \phi(A_{i,j}, A_{i,j+1}) \mid a_{1,1}, a_{1,2}]$ and any term $\mathbf{E}_Q[\ln \phi(A_{i,j}, A_{i+1,j}) \mid a_{1,1}, a_{1,2}]$ except those where $i = 1$. Similarly, we can remove any term $\mathbf{E}_Q\left[\ln \psi_{(i,j)}(A_{i,j}, A_{i,j+1}) \mid a_{1,1}, a_{1,2}\right]$ except those where $i = 1$. These simplifications result in the following update rule:

$$\psi_{1,1}(a_{1,1}, a_{1,2}) \propto \exp\left\{ \begin{array}{l} \sum_{j=1,\ldots,3} \mathbf{E}_{\{A_{1,j},A_{1,j+1}\}\sim Q}\left[\ln \phi_{(1,j)}(A_{1,j}, A_{1,j+1}) \mid a_{1,1}, a_{1,2}\right] \\ +\sum_{j=1,\ldots,4} \mathbf{E}_{\{A_{1,j},A_{2,j}\}\sim Q}\left[\ln \phi_{(1,j)}(A_{1,j}, A_{2,j}) \mid a_{1,1}, a_{1,2}\right] \\ -\sum_{j=2,3} \mathbf{E}_{\{A_{1,j},A_{1,j+1}\}\sim Q}\left[\ln \psi_{(1,j)}(A_{1,j}, A_{1,j+1}) \mid a_{1,1}, a_{1,2}\right] \end{array} \right\}.$$

■

We can generalize this analysis to arbitrary sets of factors:

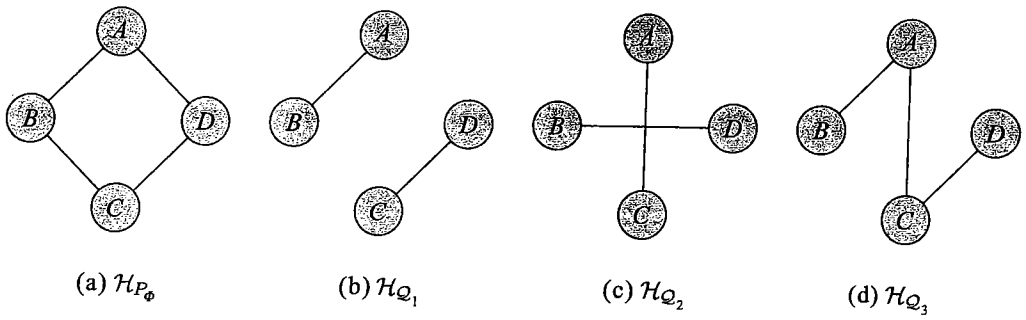
Theorem 11.12

If $Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j$, then the potential ψ_j is locally optimal only if

$$\ln \psi_j(\boldsymbol{c}_j) \propto \exp\left\{ \sum_{\phi \in A_j} \mathbf{E}_{\mathcal{X}\sim Q}[\ln \phi \mid \boldsymbol{c}_j] - \sum_{\psi_k \in B_j} \mathbf{E}_{\mathcal{X}\sim Q}[\ln \psi_k \mid \boldsymbol{c}_j] \right\}, \tag{11.62}$$

where

$$A_j = \{\phi \in \Phi : Q \not\models (\boldsymbol{U}_\phi \perp \boldsymbol{C}_j)\}$$



(a) $\mathcal{H}_{P_\Phi}$ (b) $\mathcal{H}_{Q_1}$ (c) $\mathcal{H}_{Q_2}$ (d) $\mathcal{H}_{Q_3}$

Figure 11.18 A diamond network and three possible approximating structures

and

$$B_j = \{\psi_k : Q \not\models (\boldsymbol{C}_k \perp \boldsymbol{C}_j)\} - \{\boldsymbol{C}_j\}.$$

Stated in words, this result shows that the parameterization of a factor $\psi_j(\boldsymbol{C}_j)$ depends only on factors in P_Φ and in Q whose scopes are not independent of $\boldsymbol{C}_j$ in in $\mathcal{Q}$. This result, applied to example 11.13, provides us precisely with the simplification shown: only factors whose scopes intersect with the first row are relevant to $\psi_{1,1}(A_{1,1}, A_{1,2})$. Thus, we can use independence properties of the approximating family $\mathcal{Q}$ to simplify the right-hand side of equation (11.61) by removing irrelevant terms.

11.5.2.4 Simplifying the Family $\mathcal{Q}$

It turns out that a similar analysis allows us to simplify the form of the approximating family $\mathcal{Q}$ without loss in the quality of the approximation.

We start by considering a simple example.

Example 11.14

Consider again the four-variable pairwise Markov network of figure 11.18a, which is parameterized by the pairwise factors:

$$P_\Phi(A, B, C, D) \propto \phi_{AB}(A, B) \cdot \phi_{BC}(B, C) \cdot \phi_{CD}(C, D) \cdot \phi_{AD}(A, D).$$

Consider applying the variational approximation with the distribution

$$Q(A, B, C, D) = \frac{1}{Z_Q} \psi_1(A, B) \cdot \psi_2(C, D) \tag{11.63}$$

that has the structure shown in figure 11.18b. Using equation (11.62), we conclude that the fixed-point characterization of ψ_1 is

$$\psi_1(a, b) \propto \exp\{\boldsymbol{E}_Q[\ln \phi_{AB}(A, B) \mid a, b] + \boldsymbol{E}_Q[\ln \phi_{BC}(B, C) \mid a, b] + \boldsymbol{E}_Q[\ln \phi_{AD}(A, D) \mid a, b]\}.$$

Can we further simplify this equation? Consider the first term. Clearly, $\mathbf{E}_Q[\ln \phi_{AB}(A,B) \mid a,b] = \ln \phi_{AB}(a,b)$. *What about the second term,* $\mathbf{E}_Q[\ln \phi_{BC}(B,C) \mid a,b]$? *To compute this expectation, we need to compute* $Q(B,C \mid a,b)$. *According to the structure of* Q, *we can see that*

$$Q(B,C \mid a,b) = \begin{cases} Q(C) & \text{If } B = b \\ 0 & \text{otherwise.} \end{cases}$$

Thus, we conclude that

$$\mathbf{E}_{A,B,C\sim Q}[\ln \phi_{BC}(B,C) \mid a,b] = \mathbf{E}_{C\sim Q}[\ln \phi_{BC}(b,C)].$$

We can simplify the third term in exactly the same way, concluding that:

$$\psi_1(a,b) \propto \exp\left\{\ln \phi_{AB}(a,b) + \mathbf{E}_{C\sim Q}[\ln \phi_{BC}(b,C)] + \mathbf{E}_{D\sim Q}[\ln \phi_{AD}(a,D)]\right\}.$$

Setting $\psi_1'(a) = \exp\{\mathbf{E}_{D\sim Q}[\ln \phi_{AD}(a,D)]\}$ *and* $\psi_1''(b) = \exp\{\mathbf{E}_{C\sim Q}[\ln \phi_{BC}(b,C)]\}$, *we conclude that the optimal* ψ_1 *factorizes as a product of three factors:*

$$\psi_1(A,B) = \phi_{AB}(A,B) \cdot \psi_1'(A) \cdot \psi_1''(B).$$

Have we gained anything from this decomposition? First, we see that Q *preserves the original pairwise interaction term* $\phi(A,B)$ *from* P_Φ. *Moreover, the effect of the interactions between these variables and the rest of the network (*C *and* D *in this example) is summarized by a univariate factor for each of the variables. Thus,* Q *does not change the interaction between* A *and* B.

Applying the same set of arguments to ψ_2, *we conclude that we can rewrite* Q *as*

$$Q'(A,B,C,D) = \frac{1}{Z_Q}\phi_{AB}(A,B) \cdot \phi_{CD}(C,D) \cdot \psi_1'(A) \cdot \psi_1''(B) \cdot \psi_2'(C) \cdot \psi_2''(D). \quad (11.64)$$

The preceding discussion shows that the best approximation to P_Φ *within* $\mathcal{Q}$ *can be rewritten in the form of* Q'. *Thus, there is no point in using the more complicated form of the approximating family of equation (11.63); we may as well use the form of* Q' *in equation (11.64). Note that the form of* Q' *involves a product of a subset of the original factors, which we keep intact without change, and a set of new factors, which we need to optimize. In this example, instead of estimating two pairwise potentials, we estimate four univariate potentials, which utilize a smaller number of parameters.*

Moreover, the update equations for Q' *are simpler. Consider, for example, applying equation (11.62) for* ψ_1':

$$\begin{aligned} \ln \psi_1'(a) \propto\ & \mathbf{E}_{B\sim Q'}[\ln \phi_{AB}(a,B) \mid a] + \mathbf{E}_{D\sim Q'}[\ln \phi_{AD}(a,D) \mid a] + \mathbf{E}_{B,C\sim Q'}[\ln \phi_{BC}(B,C) \mid a] \\ & - \mathbf{E}_{B\sim Q'}[\ln \phi_{AB}(a,B) \mid a] - \mathbf{E}_{B\sim Q'}[\ln \psi_1''(B) \mid a] \\ =\ & \mathbf{E}_{B\sim Q'}[\ln \phi_{AD}(a,B) \mid a] + \mathbf{E}_{B,C\sim Q'}[\ln \phi_{BC}(B,C) \mid a] - \mathbf{E}_{B\sim Q'}[\ln \psi_1''(B) \mid a]. \end{aligned}$$

The terms involving $\mathbf{E}_{Q'}[\ln \phi_{AB} \mid a]$ *appear twice, once as a factor in* P_Φ *and once as a factor in* Q'. *These two terms cancel out, and we are left with the simpler update equation. Although this equation does not explicitly mention* ϕ_{AB}, *this factor participates in the computation of* $Q'(B \mid a)$ *that implicitly appears in* $\mathbf{E}_{B\sim Q'}[\ln \psi_1''(B) \mid a]$. ■

Note that this result is somewhat counterintuitive, since it shows that the interactions between A and B are captured by the original potential in P_Φ. Intuitively, we would expect the chain of influence A—D—C—B to introduce additional interactions between A and B that should be represented in Q. This is not the only counterintuitive result.

Example 11.15

Consider another approximating family for the same network, using the network structure shown in figure 11.18c. In this approximation, we have two pairwise factors, $\psi_1(A, C)$, and $\psi_2(B, D)$. Applying the same set of arguments as before, we can show that the update equation can be written as

$$\begin{aligned}\ln \psi_1(a, c) \quad \propto \quad & \boldsymbol{E}_{B \sim Q}[\ln \phi_{AB}(a, B)] + \boldsymbol{E}_{D \sim Q}[\ln \phi_{AD}(a, D)] \\ & + \boldsymbol{E}_{B \sim Q}[\ln \phi_{BC}(B, c)] + \boldsymbol{E}_{D \sim Q}[\ln \phi_{CD}(c, D)].\end{aligned}$$

Thus, we can factorize ψ_1 into two factors, one with a scope of A and the other with C

$$\psi_1(A, C) = \psi_1'(A) \cdot \psi_1''(C).$$

In other words, the approximation in this case is equivalent to the mean field approximation. This result shows that, in some cases, we can remove spurious dependencies in the approximating distribution. However, this result is surprising, since it holds regardless of the actual values of the potentials in P_Φ. And so, we can imagine a network where there are very strong interactions between A and C and between B and D in P_Φ, and yet the variational approximation with a network structure of figure 11.18c will not capture these dependencies. This is a consequence of using I-projections. Had we used an M-projection that minimizes $\boldsymbol{D}(P_\Phi \| Q)$, then we would have represented the dependencies between A and C; see exercise 11.30. ■

These two examples suggest that we can use the fixed-point characterization to refine an initial approximating network by factorizing its factors into a product of, possibly smaller, factors and potentials from P_Φ. We now consider the general theory of such factorizations and then discuss its implications.

We start with a simple definition and a proposition that form the basis of the simplifications we consider.

Definition 11.8

interface

*Let $\mathcal{H}$ be a Markov network structure and let $\boldsymbol{X}, \boldsymbol{Y} \subseteq \mathcal{X}$. We define the $\boldsymbol{Y}$-*interface *of $\boldsymbol{X}$, denoted $Interface_{\mathcal{H}}(\boldsymbol{X}; \boldsymbol{Y})$, to be the minimal subset of $\boldsymbol{X}$ such that $\text{sep}_{\mathcal{H}}(\boldsymbol{X}; \boldsymbol{Y} \mid Interface_{\mathcal{H}}(\boldsymbol{X}; \boldsymbol{Y}))$.* ■

That is, the $\boldsymbol{Y}$-interface of $\boldsymbol{X}$ is the subset of $\boldsymbol{X}$ that suffices to separate it from $\boldsymbol{Y}$.

Example 11.16

The $\{A, D\}$-interface of $\{A, B\}$ in $\mathcal{H}_{P_\Phi}$ of figure 11.18 is $\{A, B\}$, since neither A is separated from $\{A, D\}$ given B, nor is B is separated from $\{A, D\}$ given A. In $\mathcal{H}_{\mathcal{Q}_1}$, we have that B is separated from $\{A, D\}$ given A, so that $Interface_{\mathcal{H}_{\mathcal{Q}_1}}(\{A, B\}; \{A, D\})$ is $\{A\}$. The same holds in $\mathcal{H}_{\mathcal{Q}_3}$. In $\mathcal{H}_{\mathcal{Q}_2}$, we have that, again, neither A nor B suffices to separate the other from $\{A, D\}$, and hence, $Interface_{\mathcal{H}_{\mathcal{Q}_2}}(\{A, B\}; \{A, D\}) = \{A, B\}$. ■

The definition of interface can be used to reduce the scope of the conditional expectations in the fixed-point equations:

Proposition 11.6

If $\mathcal{H}$ is an I-map of $Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j$ and ϕ is a potential with scope $\boldsymbol{U}_\phi$. Then,

$$\boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\phi \mid \boldsymbol{c}_j] = \boldsymbol{E}_{\boldsymbol{U}_\phi \sim Q}[\phi \mid \boldsymbol{c}_j \langle Interface_{\mathcal{H}}(\boldsymbol{C}_j; \boldsymbol{U}_\phi) \rangle].$$

The proof follows immediately from the definition of conditional independence.

This proposition provides a principled approach for reformulating terms on the right-hand side of the fixed-point equation.

☞ **We can use this simplification result to define a two-phase strategy for designing approximation. First, we define a "rough" outline for approximation by defining Q over factors with a fairly large scope. We use this outline to obtain a set of update equations, as implied by equation (11.62) on Q. We then derive a finer-grained representation by factorizing each of these factors using proposition 11.6. This process results in a finer-grained approximation that is provably equivalent to the one with which we started.**

Theorem 11.13 **(Factorization)** *Let $\mathcal{Q}$ be an approximating family defined in terms of factors $\{\psi_j(\boldsymbol{C}_k)\}$, which induce a Markov network structure $\mathcal{H}_{\mathcal{Q}}$. Let $Q \in \mathcal{Q}$ be a stationary point of the energy functional $F[\tilde{P}_\Phi, Q]$ subject to the given factorization. Then, factors in Q are factorized as*

$$\psi_j(\boldsymbol{C}_j) = \prod_{\phi \in \Phi_j} \phi \prod_{\boldsymbol{D}_l \in \mathcal{D}_j} \psi_{j,l}(\boldsymbol{D}_l), \tag{11.65}$$

where

$$\Phi_j = \{\phi \in \Phi : Scope[\phi] \subseteq \boldsymbol{C}_j\}$$

and

$$\mathcal{D}_j = \{Interface_{\mathcal{H}_{\mathcal{Q}}}(\boldsymbol{C}_j; \boldsymbol{X}) : \boldsymbol{X} \in \{Scope[\phi] : \phi \in \Phi - \Phi_j\} \cup \{Scope[\psi_k] : k \neq j\}\}.$$

This theorem states that ψ_j can be written as the product of two sets of factors. The first set contains factors in the original distribution P_Φ whose scope is a subset of the scope of ψ_j. The factors in the second set are the interfaces of ψ_j with other factors that appear in the update equation. These include factors in P_Φ that are partially "covered" by the scope of ψ_k, and other factors in Q. The set $\mathcal{D}_k$ defines the set of interfaces between ψ_k and these factors.

To gain a better understanding of this theorem, let us consider various approximations in two concrete examples. The first example serves to demonstrate the ease with which this theorem allows us to determine the form of the factorization of Q.

Example 11.17 *Let us return to example 11.14. In example 11.16, we have already shown the interfaces of $\{A, B\}$ with $\{A, D\}$ in $\mathcal{H}_1$. This analysis, together with theorem 11.13, directly imply the reduced factorization of example 11.14 In particular, for $\psi_1(\{A, B\})$, we have that Φ_1 contains only the factor $\phi(\{A, B\})$ in P_Φ, which therefore constitutes the first term in the factorization of equation (11.65). The second set of terms in the equation corresponds to the interfaces of $\{A, B\}$ with other factors in both $\mathcal{H}_{P_\Phi}$ and in $\mathcal{H}_{\mathcal{Q}_1}$. We get two such interfaces: one with scope $\{A\}$ from the factor $\phi(\{A, D\})$ in P_Φ, and one with scope $\{B\}$ from the factor $\phi(\{B, C\})$.*

Assume that we add the edge A—C, as in figure 11.18d. Now, $Interface_{\mathcal{H}_{\mathcal{Q}_3}}(\{A, B\}; \{B, C\})$ is the entire set $\{A, B\}$, since B no longer separates C from A. Thus, in this case, the second set of terms in the factorization of ψ also contains a new pairwise interaction factor $\psi_{1,\{A,B\}}$. As a consequence, the pairwise interaction of A, B is no longer the same in Q and in P_Φ. This result is somewhat counterintuitive: In the simpler network $\mathcal{H}_{\mathcal{Q}_1}$, which contained no factors allowing any interaction between the A, B pair and the C, D pair, the A, B interaction was the same in P_Φ and

in Q. But if we enrich our approximation (presumably allowing a better fit via the introduction of the A, C factor), the pairwise interaction term does change.

Finally, $\mathcal{H}_{Q_2}$ does not contain an $\{A, B\}$ factor. Here, $\Phi_j = \emptyset$ for both factors in $\mathcal{H}_{Q_2}$, and each $\mathcal{D}_j$ consists solely of singleton scopes; for example, $\text{Interface}_{\mathcal{H}_{Q_2}}(\{A, C\}; \{A, D\}) = \{A\}$. ■

Our second example serves to illustrate the two-phase strategy described earlier, where we first select a "rough" approximation containing a few large factors and then use the theorem to refine them.

Example 11.18

Consider again our running example of the 4×4 grid. Suppose we select an approximation where each factor consists of the variables in a single row in the grid. Thus, for example, $\boldsymbol{C}_1 = \{A_{1,1}, \ldots, A_{1,4}\}$. Note that this approximation is not the one shown in figure 11.17a, since the structure in our approximation here is a full clique over each row. We now apply theorem 11.13. What is the factorization of $\boldsymbol{C}_1$? First, we search for factors in Φ_1. We see that the factors $\phi(A_{1,1}, A_{1,2})$, $\phi(A_{1,2}, A_{1,3})$, and $\phi(A_{1,3}, A_{1,4})$ have a scope that is a subset of $\boldsymbol{C}_1$. Next, we consider the interfaces between $\boldsymbol{C}_1$ and other factors in P_Φ and Q. For example, the interface with $\phi(A_{1,1}, A_{2,1})$ is $\{A_{1,1}\}$. Similarly, $\{A_{1,2}\}$, $\{A_{1,3}\}$, and $\{A_{1,4}\}$ are interfaces with other factors in P_Φ. It is easy to convince ourselves that these are the only non-empty interfaces in $\mathcal{I}_1$. Thus, by applying theorem 11.13, we get the following factorization:

$$\begin{aligned}\psi_1(A_{1,1}, \ldots, A_{1,4}) &= \phi(A_{1,1}, A_{1,2}) \cdot \phi(A_{1,2}, A_{1,3}) \cdot \phi(A_{1,3}, A_{1,4}) \\ &\quad \psi_{1,1}(A_{1,1}) \cdot \psi_{1,2}(A_{1,2}) \cdot \psi_{1,3}(A_{1,3}) \cdot \psi_{1,4}(A_{1,4}).\end{aligned}$$

We conclude that, once we decide that the approximation should decouple the rows in the group, we might as well work with an approximation where we keep all original potentials along each row and introduce univariate potentials only to capture interactions along columns. Additional potentials, such as a potential between $A_{1,1}$ and $A_{1,3}$, would not improve the approximation. Thus, while we started with an approximation containing full cliques on each of the rows, we ended up with an approximation whose structure is that of figure 11.17a, and where we have only the original factors and new factors over single variables.

We can work directly with this new factorized form of Q, ignoring our original factorization entirely. More precisely, we define Q' to be

$$\begin{aligned}Q'(\mathcal{X}) &= \phi(A_{1,1}, A_{1,2}) \cdot \phi(A_{1,2}, A_{1,3}) \cdot \phi(A_{1,3}, A_{1,4}) \\ &\quad \cdots \\ &\quad \phi(A_{4,1}, A_{4,2}) \cdot \phi(A_{4,2}, A_{4,3}) \cdot \phi(A_{4,3}, A_{4,4}) \\ &\quad \psi_{1,1}(A_{1,1}) \cdot \cdots \psi_{4,4}(A_{4,4}).\end{aligned}$$

In this new form, we fix the value of all the pairwise potentials, and so we have to define an update rule only for the new singleton potentials. For example, consider the fixed-point equation for $\psi_{1,1}(A_{1,1})$. Applying theorem 11.12 we get that

$$\begin{aligned}&\ln \psi_{1,1}(a_{1,1}) \propto \\ &\quad + \boldsymbol{E}_{Q'}[\ln \phi(A_{1,1}, A_{2,1}) \mid a_{1,1}] + \boldsymbol{E}_{Q'}[\ln \phi(A_{1,2}, A_{2,2}) \mid a_{1,1}] \\ &\quad + \boldsymbol{E}_{Q'}[\ln \phi(A_{1,3}, A_{2,3}) \mid a_{1,1}] + \boldsymbol{E}_{Q'}[\ln \phi(A_{1,4}, A_{2,4}) \mid a_{1,1}] \\ &\quad - \boldsymbol{E}_{Q'}[\ln \psi_{1,2}(A_{1,3}) \mid a_{1,1}] - \boldsymbol{E}_{Q'}[\ln \psi_{1,3}(A_{1,3}) \mid a_{1,1}] - \boldsymbol{E}_{Q'}[\ln \psi_{1,3}(A_{1,3}) \mid a_{1,1}]\end{aligned}$$

where we have exploited the fact that the terms involving factors such as $\phi(A_{1,1}, A_{1,2})$ appear in both P_Φ and Q, and so cancel out of the equation. Note that to compute terms such as $E_{Q'}[\ln \phi(A_{1,2}, A_{2,2}) \mid a_{1,1}]$ we need to evaluate $Q'(A_{1,2}, A_{2,2} \mid a_{1,1}) = Q'(A_{1,2} \mid a_{1,1}) \cdot Q'(A_{2,2})$ (where we used the independencies in Q' to simplify the joint marginal). Note that $Q'(A_{2,2})$ does not change when we update factors in the first row, such as $\psi_{1,1}(A_{1,1})$. Thus, we can cache the computation of this marginal when updating the factors $\psi_{1,1}(A_{1,1}), \ldots, \psi_{1,4}(A_{1,4})$. When performing inference in a large model this can result in dramatic effect. ■

cluster mean field

This example is a special case of an approximation approach called *cluster mean field.* In this case, our initial approximation has the form

$$Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j(\boldsymbol{C}_j),$$

where the scopes $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_K$ are partition of $\mathcal{X}$. That is, each pair of factors have disjoint scopes, and each variable in $\mathcal{X}$ appears in one factor. This approximation resembles the mean field approximation, except that it is clusters, rather than individual variables, that are marginally independent. We can now apply theorem 11.13 to refine the approximation. Because the factors are all disjoint, there are no chains of influence, and so the interfaces take a particularly simple form:

Proposition 11.7

Let $Q(\mathcal{X}) = \frac{1}{Z_Q} \prod_j \psi_j(\boldsymbol{C}_j)$ be a cluster mean field approximation to a set of factors P_Φ, and let ψ_j be a factor of Q. Then, the set $\mathcal{D}_j$ of theorem 11.13 can be written as

$$\mathcal{D}_j = \{\boldsymbol{C}_j \cap Scope[\phi] : \phi \in \Phi - \Phi_j\} - \{\emptyset\}.$$

The proof follows directly from the independence properties in Q, and is left as an exercise (exercise 11.31).

In words, this result states that the interfaces of a cluster are simply the places where the cluster scope intersects potentials in Φ that are not fully contained in the cluster. In our grid example, when we choose the clusters to be the individual columns, the interfaces are the intersections with the row potentials, which are precisely the singleton variables that we discussed in example 11.18.

We conclude this discussion with a slightly more elaborate example, demonstrating again the strength of this result:

Example 11.19

Consider again our 4×4 grid, and the "comb" approximation whose structure is shown in figure 11.19a. In this structure, we have a fully connected clique over each of the columns, and a "backbone" connecting the columns to each other. Consider again the factorization of the potential over $\boldsymbol{C}_1 = \{A_{1,1}, \ldots, A_{4,1}\}$. As in the previous example, the first term in the new factorization contains the pairwise factors $\phi(A_{1,1}, A_{2,1})$, $\phi(A_{2,1}, A_{3,1})$, and $\phi(A_{3,1}, A_{4,1})$. The second set of terms contains the interfaces with other factors in P_Φ and Q. Due to the structure of the approximation, the Q interfaces introduce only singleton potentials. The factors in P_Φ, however, are more interesting. Consider, for example, the factor $\phi(A_{4,1}, A_{4,2})$. The interface of $\boldsymbol{C}_1$ with $\{A_{4,1}, A_{4,2}\}$ is $A_{1,1}, A_{4,1}$ — the variable $A_{4,1}$ separates $\boldsymbol{C}_1$ from itself, and the variable $A_{1,1}$ from $A_{4,2}$. Now, consider a factor $\phi(A_{2,3}, A_{3,3})$; in this case, the interface is simply $A_{1,1}$, which separates the first column from both of these variables. Continuing this argument, it follows that all other factors in

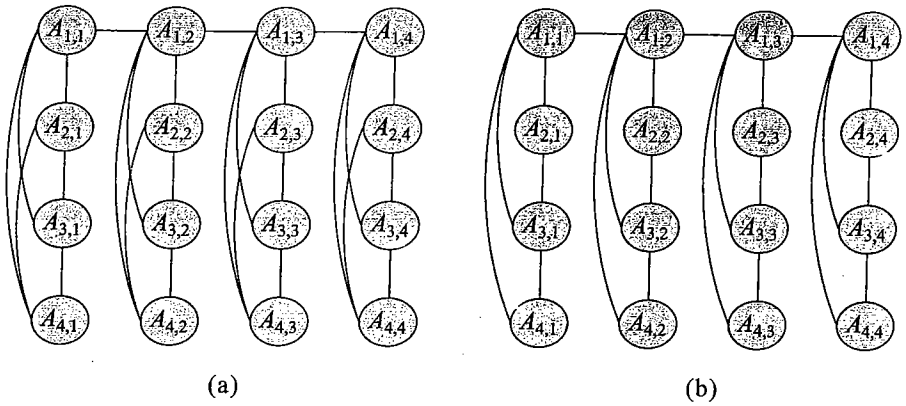


Figure 11.19 Simplification of approximating structure in cluster mean field. (a) Example of an approximating structure we can use in a variational approximation of a 4×4 grid network. (b) Simplification of that network using theorem 11.13.

P_Φ induce an interface containing a variable at the head of the column and (possibly) another variable in the column. Thus, we can eliminate any (new) pairwise interaction terms between any other pair of variables. For a general $n \times n$ grid, this reduces the overall number of (new) pairwise potentials from $n \cdot \binom{n}{2}$ to $n \times (n-1)$. ■

11.5.2.5 Selecting the Approximation

In general, both the quality and the computational complexity of the variational approximation depend on the structure of P_Φ and the structure of the approximating family $\mathcal{Q}$. There are several guiding intuitions. First, we want to be able to perform efficient inference in the approximating network. In example 11.18, the approximating structure was a chain of variables, where we can perform inference in linear time (as a function of the number of variables in the chain). In general, we often select our network so that the resulting factorization leads to a tractable network (that is, one of low tree-width).

It is important to note, however, that the structure of the original distribution is not the only aspect in determining the complexity of inference in Q. We also need to take into account factors that correspond to the interfaces of the cluster. In our grid example, these interfaces involved a single variable at time, and so they did not add to the network complexity. However, in more complex networks, these factors can have a significant effect.

Another consideration besides computational complexity is the quality of our approximation. Intuitively, we should design $\mathcal{Q}$ so as to preserve the strong dependencies in P_Φ. By preserving such dependencies we maintain the main effects in the distribution we want to apply.

These intuitions provide some guidelines in choosing the approximating distribution. However, these choices are far from an exact science at this stage. The theory we described here allows to automate two parts of the process: defining the form of the approximation given some initial rough set of (disjoint or overlapping) clusters; and defining the fixed-point iterations to

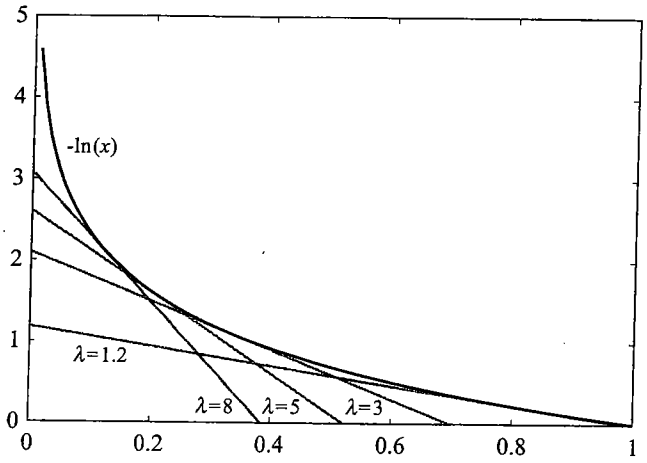


Figure 11.20 Illustration of the variational bound $-\ln(x) \geq -\lambda x + \ln(\lambda) + 1$. For any x, the bound holds for all λ, and is tight for some value of λ.

optimize such an approximation. The current tools do not provide for an automated way for determining what are reasonable sets of clusters to achieve a desired degree of approximation.

11.5.3 Local Variational Methods ⋆

variational method

lower bound

The general method that we used throughout this chapter is an instance of a general class of methods known as *variational methods*. In this class of methods, we take a complex objective function $f_{\text{obj}}(\boldsymbol{x})$, and lower or upper bound it using a parameterized family of functions $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\lambda})$. Focusing, for concreteness, on the case of a *lower bound*, this family has the property that $f_{\text{obj}}(\boldsymbol{x}) \geq \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\lambda})$ for any value of $\boldsymbol{\lambda}$, and that, for any $\boldsymbol{x}$, the bound is tight for some value of $\boldsymbol{\lambda}$ (a different one for every $\boldsymbol{x}$).

variational lower bound

As an example, we can show the *variational lower bound*:

Lemma 11.2

For any choice of λ and x

$$-\ln(x) \geq -\lambda x + \ln(\lambda) + 1,$$

and, for any x, this bound is tight for some value of λ.

Proof Consider the tangent of $\ln(x)$ at the point x_0

$$f_{\text{obj}}(x : x_0) = \ln(x_0) + (x - x_0)\frac{1}{x_0} = \frac{x}{x_0} + \ln(x_0) - 1.$$

Since $\ln(x)$ is a concave function, it is upper bounded by each of its tangents. And so, $-\ln(x) \geq -f_{\text{obj}}(x : x_0)$ for any choice of x and x_0. Setting $x_0 = \lambda^{-1}$ leads to the desired result. ■

convex duality

This result is illustrated in figure 11.20. It is a special case of a general result in the field of *convex duality*, which guarantees the existence of such bounds for a broad class of functions.

variational parameter

This lower bound allows to approximate a nonlinear function $-\ln(x)$ with a term that is linear in x. This simplification comes at the price of introducing a new *variational parameter* λ, whose value is undetermined. If we optimize λ exactly for each value of x, we obtain a tight lower bound, but a bound is obtained for any value of λ.

The techniques we have used in this chapter so far also fall into this category. Equation (11.5) shows that the energy functional is a lower bound on the log-partition function for any distribution Q. Thus, we can take f_{obj} to be the partition function, $\boldsymbol{x}$ to correspond to the parameters of the true distribution P_Φ, and $\boldsymbol{\lambda}$ to correspond to the parameters of the approximating distribution Q. Although the lower bound is tight when Q precisely represents P_Φ, for reasons of efficiency, we generally optimize Q in a restricted space that provides a bound, but not a tight one, on the log-partition function.

This general approach of introducing auxiliary variational parameters that help in simplifying a complex objective function appears in many other domains. While it is beyond our scope to introduce a general theory of variational methods, we now briefly describe one other application of variational methods that is relevant to probabilistic inference and does not fall directly within the scope of optimizing the energy functional. This application arises in the context of exact inference using an algorithm such as variable elimination. Here, we use variational bounds to avoid creating large factors that can lead to exponential complexity in the algorithm, giving rise to an approximate *variational variable elimination* algorithm. Such simplifications can be achieved in several ways; we describe two.

variational variable elimination

11.5.3.1 Variational Bounds

Consider, for example, the diamond network of figure 11.18a. Assume that we run variable elimination to sum out the variable B, which we assume for convenience is binary-valued. The elimination of B introduces a new factor:

$$\phi_B(A, C) = \sum_b \phi_1(A, b)\phi_2(b, C)$$

Coupling A and C in a single factor may be expensive, for example, if A and C have many values. In more complex networks, this type of coupling can induce complexity if an elimination step couples a larger set of variables, or if the local coupling leads to additional cost later in the computation, when we eliminate A or C.

As we now show, we can use a variational bound to avoid this coupling. Consider the following bound:

Proposition 11.8

If $0 \leq \lambda \leq 1$, *then*

$$\ln(1 + e^x) \leq \lambda x + \boldsymbol{H}(\lambda),$$

where $\boldsymbol{H}(\lambda) = -\lambda \ln \lambda - (1 - \lambda)\ln(1 - \lambda)$.

This bound implies that

$$1 + e^x \leq e^{\lambda x + \boldsymbol{H}(\lambda)}.$$

Why is this useful? Using some algebraic manipulation, we can bound each of the entries in our newly generated factor:

$$\begin{aligned}
\phi_B(a,c) &= \phi_1(b^0,a)\phi_2(b^0,c) + \phi_1(b^1,a)\phi_2(b^1,c) \\
&= \phi_1(b^0,a)\phi_2(b^0,c)\left[1 + \exp\left\{\ln \frac{\phi_1(b^1,a)\phi_2(b^1,c)}{\phi_1(b^0,a)\phi_2(b^0,c)}\right\}\right] \\
&\geq \phi_1(b^0,a)\phi_2(b^0,c)\exp\left\{\lambda_{a,c}\ln \frac{\phi_1(b^1,a)\phi_2(b^1,c)}{\phi_1(b^0,a)\phi_2(b^0,c)} + \boldsymbol{H}(\lambda_{a,c})\right\} \\
&= \left(\phi_1(b^0,a)^{1-\lambda_{a,c}}\phi_1(b^1,a)^{\lambda_{a,c}}\right) \cdot \\
&\quad \left(\phi_2(b^0,c)^{1-\lambda_{a,c}}\phi_2(b^1,c)^{\lambda_{a,c}}\right) \cdot e^{\boldsymbol{H}(\lambda_{a,c})}.
\end{aligned} \tag{11.66}$$

Thus, we can replace a factor that couples A and C by a product of three terms: an expression involving only factors of A, an expression involving only factors of C, and the final factor $e^{\boldsymbol{H}(\lambda_{a,c})}$. However, all three terms also involve the variational parameter $\lambda_{a,c}$ and therefore also depend on both A and C. At this point, it is unclear what we gain from the transformation.

However, we can choose the same λ for all joint assignments to A, C. In doing so, we replace four variational parameters by a single parameter λ. This operation relaxes the bound, which is no longer tight. On the other hand, it also decouples A and C, leading to a product of terms none of which depends on both variables:

$$\left(\phi_1(b^0,a)^{1-\lambda}\phi_1(b^1,a)^{\lambda}\right) \cdot \left(\phi_2(b^0,c)^{1-\lambda}\phi_2(b^1,c)^{\lambda}\right) \cdot e^{\boldsymbol{H}(\lambda)} = \tilde{\phi}_1(a,\lambda)\tilde{\phi}_2(c,\lambda)e^{\boldsymbol{H}(\lambda)}. \tag{11.67}$$

Thus, if we use this approximation, we have effectively eliminated B without coupling A and C. As we saw in chapter 9, this type of simplification can circumvent the need for coupling yet more variables in later stages in variable elimination, potentially leading to significant savings.

It is interesting to observe how λ decouples the two factors. Each factor is replaced by a geometric average over the values of B. The variational parameter specifies the weight we assign to each of the two cases. Note that the original bound in equation (11.66) is tight; thus, if we pick the "right" variational parameter $\lambda_{a,c}$ for each assignment a, c, we reproduce the correct factor $\phi_B(A, C)$. However, these variational parameters are generally different for each assignment a, c, and hence, a single variational parameter cannot optimize all of the terms. Our choice of λ effectively determines the quality of our approximation for each of the terms $\phi_B(a, c)$. Thus, the overall quality of our approximation for a particular choice of λ depends on the importance of these different terms in the variable elimination computation as a whole.

Other variational approximations exploit specific parametric forms of CPDs in the network. For example, consider networks with sigmoid CPDs (see section 5.4.2). Recall that a logistic CPD $P(X \mid \boldsymbol{U})$ has the parametric form:

$$P(x^1 \mid \boldsymbol{u}) = \text{sigmoid}\left(\sum_i w_i u_i + w_0\right),$$

where $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$. The observation of X couples the parents $\boldsymbol{U}$. Can we decouple these parents using an approximation? Using proposition 11.8 we can find an upper bound:

$$\ln \text{sigmoid}\left(\sum_i w_i u_i + w_0\right) \leq \lambda\left(\sum_i w_i u_i + w_0\right) - \boldsymbol{H}(\lambda). \tag{11.68}$$

Similarly to our earlier example, such an approximation allows us to replace a factor over several variables by a product of smaller factors. In this case, all the parents of X are decoupled by the approximate form.

11.5.3.2 Variational Variable Elimination

How do we use this approximation in the course of inference? Note that equation (11.67) provides a lower bound to $\phi_B(a, c)$ for every value of a, c. Assume that, in the course of running variable elimination, rather than generating $\phi_B(A, C)$, we introduce the expression in equation (11.67) and continue the variable elimination process with these decoupled factors. From a graph-theoretic perspective, the result of this approximation when applied to a variable B is analogous to the effect of conditioning on B, as described in section 9.5.3: it removes from the graph B and all its adjacent edges. However, unlike conditioning, we do not enumerate and perform inference for all values of B (of course, at the cost of obtaining an approximate result).

More generally, in an execution of variable elimination, there may be some set of elimination steps that create large factors that couple many variables. This variational approximation can allow us to avoid this coupling. Like conditioning (see section 9.5.4.1), we can perform such an approximation step not only at the very beginning, but also in a way that is interleaved with variable elimination, allowing us to reuse computation. This class of algorithms is called *variational variable elimination*.

variational variable elimination

What is the result of this approximation? Each of the entries in the approximated factor is replaced with a lower bound; thus, each entry in every subsequent factor produced by the algorithm is also a lower bound to the original entry. If we proceed to eliminate all variables, either exactly or using additional variational approximation steps for other intermediate factors, the outcome of this process is a lower bound to the partition function. If we do not eliminate all of the variables, the result is an approximate factor in which every entry is a lower bound to the original. Of course, once we renormalize the factor to produce a distribution, we can make no guarantees about the direction of the approximation for any given entry. Nevertheless, the resulting factor might be a reasonable approximation to the original.

The quality of our approximation depends on the choice of variational parameters introduced during the course of the variable elimination algorithm. How do we select them? One approach is simply to select the variational parameter at each step to optimize the quality of our approximation at that step. However, this high-level goal is not fully defined. For example, we can choose λ so as to make $\tilde{\phi}_1(a^1, \lambda)\tilde{\phi}_2(b^1, \lambda)e^{\boldsymbol{H}(\lambda)}$ as close as possible to $\phi_B(a^1, c^1)$; or, we can focus on a^1, c^0. The decision of where to focus our "approximation effort" depends on the impact of these components of the factor on the final outcome of the computation. Thus, a more correct approach is to identify the actual expression that we are trying to estimate — for example, the partition function — and to try to maximize our bound to that expression. In our simple example, we can write down the partition function as a function of the variational parameter λ introduced when eliminating B:

$$\tilde{Z}(\lambda) = e^{\boldsymbol{H}(\lambda)} \sum_a \sum_c \phi_3(a, d)\phi_4(c, d) \sum_d \tilde{\phi}_1(a, \lambda)\tilde{\phi}_2(c, \lambda).$$

This expression is a function of λ; we can then try to identify the best bound by maximizing $\max_\lambda \tilde{Z}(\lambda)$, say, using gradient ascent or another numerical optimization method.

However, as we discussed, in most cases we would use several approximate elimination steps within the variable elimination algorithm. In our example, the elimination of D also couples A and C, a situation we may wish to avoid. Thus, we could apply the same type of variational bound to the internal summation:

$$\phi_D(A, C) = \sum_d \tilde{\phi}_1(a, \lambda)\tilde{\phi}_2(c, \lambda),$$

giving rise to the bound $\tilde{\phi}_3(a, \lambda')\tilde{\phi}_4(b, \lambda')e^{\boldsymbol{H}(\lambda')}$. The resulting approximate partition function now has the form

$$\tilde{Z}(\lambda, \lambda') = e^{\boldsymbol{H}(\lambda)}e^{\boldsymbol{H}(\lambda')} \sum_a \tilde{\phi}_1(a, \lambda)\tilde{\phi}_3(a, \lambda') \sum_c \tilde{\phi}_2(c, \lambda)\tilde{\phi}_4(c, \lambda').$$

We can now maximize $\max_{\lambda,\lambda'} \tilde{Z}(\lambda, \lambda')$; the higher the value we find, the better our approximation to the true partition function.

In general, our approximate partition function will be a function $\tilde{Z}(\boldsymbol{\lambda})$, where $\boldsymbol{\lambda}$ is the vector of all variational parameters λ produced during the different approximation steps in the algorithm. One approach is to reformulate the variable elimination algorithm so that it produces factors that are not purely numerical, but rather symbolic expressions in the variables $\boldsymbol{\lambda}$; see exercise 11.32. Given the multivariate function $\tilde{Z}(\boldsymbol{\lambda})$, we can optimize it numerically to produce the best possible bound. A similar principle applies when we use the variational bound for sigmoid CPDs; see exercise 11.36 for an example. While we only sketch the basic idea here, this approach can be used as the basis for an algorithm that interleaves variational approximation steps with exact elimination steps to form a variational variable elimination algorithm.

11.6 Summary and Discussion

In this chapter, we have described a general class of methods for performing approximate inference in a distribution P_Φ defined by a graphical model. These methods all attempt to construct a representation Q within some approximation class $\mathcal{Q}$ that best approximates P_Φ. The key issue that must be tackled in this task is the construction of such an approximation without performing inference on the (intractable) P_Φ. The methods that we described all take a similar approach of using the I-projection framework, whereby we minimize the KL-divergence $\boldsymbol{D}(Q\|P_\Phi)$; these methods all reformulate this problem as one of maximizing the energy functional.

The different methods that we described all follow a similar template. We optimize the free energy, or an approximation thereof, over a class of representations $\mathcal{Q}$. We provided an optimization-based view for four different methods, each of which makes a particular choice regarding the objective and the constraints. We now recap these choices and their repercussions.

Clique tree calibration optimizes the factored energy functional, which is exact for clique trees. The optimization is performed over the space of calibrated clique potentials. For clique trees, any set of calibrated clique potentials must arise from a real distribution, and so this space is precisely the marginal polytope. As a consequence, both our objective and our constraint space are exact, so our solution represents the exact posterior.

Cluster-graph (loopy) belief propagation optimizes the factored energy functional, which is approximate for loopy graphs. The optimization is performed over the space of locally consistent

pseudo-marginals, which is a relaxation of the marginal polytope. Thus, both the objective and the constraint space are approximate.

Expectation propagation over clique trees optimizes the factored energy functional, which is the exact objective in this case. However, the constraints define a space of pseudo-marginals that are not even entirely consistent — only their moments are required to match. Thus, the constraint space here is a relaxation of our constraints, even when the structure is a tree. Expectation propagation over cluster graphs adds, on top of these, all of the approximations induced by belief propagation.

Finally, structured variational methods optimize the exact factored energy functional, for a class of distributions that is (generally) less expressive than necessary to encode P_Φ. As a consequence, the constraint space is actually a tightening of our original constraint space. Because the objective function is exact, the result of this optimization provides a lower bound on the value of the exact optimization problem. As we will see, such bounds can play an important role in the context of learning.

In the approaches we discussed, the optimization method was based on the method of Lagrange multipliers. This derivation gave rise to a series of fixed-point equations for the solution, where one variable is defined in terms of the others. As we showed, an iterative solution for these fixed-point equations gives rise to a suite of message passing algorithms that generalize the clique-tree message passing algorithms of chapter 10.

This general framework opens the door to the development of many other approaches. **Each of the methods that we described here involves a design choice in three dimensions: the objective function that we aim to optimize, the space of (pseudo-)distributions over which we perform our optimization, and the algorithm that we choose to use in order to perform the optimization. Although these three decisions affect each other, these dimensions are sufficiently independent that we can improve each one separately.** Some recent work takes exactly this approach. For example, we have already seen some work that focuses on better approximations to the energy functional; other work (see section 11.7) focuses on identifying constraints over the space of pseudo-marginals that make it a tighter relaxation to the (exact) marginal polytope; yet other work aims to find better (for example, more convergent) algorithms for a general class of optimization problems.

Many of these improvements aimed to address a fundamental problem arising in these algorithms: the possible lack of convergence to a stable solution. The issue of convergence is one on which some progress has been made. Some recently developed methods have better convergence properties in practice, and others are even guaranteed to converge. There are also theoretical analyses that can help determine when the algorithms converge.

A second key question, and one on which relatively little progress has been made, is the quality of the approximation. There is very little work that provides guarantees on the error made in the answers (for example, individual marginals) by any of these methods. As a consequence, there is almost no work that provides help in choosing a low-error approximation for a particular problem. Thus, the problem of applying these methods in practice is still a combination of manual tuning on one hand, and luck on the other. The development of automated methods that can help guide the selection of an approximating class for a particular problem is an important direction for future work.

11.7 Relevant Literature

Variational approximation methods are applicable in a wide variety of settings, including quantum mechanics (Sakurai 1985), statistical mechanics (Parisi 1988), neural networks (Elfadel 1995), and statistics (Rustagi 1976). The literature on the use of these approximation methods for inference problems in graphical models is heavily influenced by ideas in statistical mechanics, although the connections are beyond the scope of this book.

The rules for belief propagation were developed in Pearl's (1986b; 1988) analysis of inference in singly connected networks. In his book, Pearl noted that these propagation rules lead to wrong answers in multiconnected networks. Interest in "loopy" inference on pairwise networks was raised in several publications (Frey 1998; MacKay and Neal 1996; Weiss 1996) that reported good empirical results when running Pearl's algorithm on networks with loops. The most impressive success was in the error-correcting code scheme known as turbocodes (Berrou et al. 1993; McEliece et al. 1995). These decoding algorithms were shown to be equivalent to belief propagation in a network with loops (Kschischang and Frey 1998; McEliece et al. 1998; Frey and MacKay 1997). Researchers using these ad hoc algorithms reported that although they did not compute the correct posteriors, they often did compute the correct MAP assignment.

These empirical successes led to examination of both the reasons for the success of such methods and evaluated their performance in other domains. Weiss (2000) showed that when a network has a single loop, an analytic relationship exists between the belief computed and the true marginals, and it used that relationship to characterize network topologies for which the MAP solution is provably correct. Weiss and Freeman (2001a) then showed that if loopy belief propagation converges in a linear Gaussian network, then it computes correct posterior means; see section 14.7 for more references on results for the Gaussian case. These initial analytic results were supplemented by promising empirical results (Murphy et al. 1999; Horn and McEliece 1997; Weiss 2001).

Several alternative formulations of loopy belief propagation appeared in the literature. These include factor graphs (Kschischang et al. 2001b), tree-based reparameterization (Wainwright et al. 2003a), and algebraic formulations (Aji and McEliece 2000). In parallel, several authors (Yedidia et al. 2000; Dechter et al. 2002) proposed extending the idea of belief propagation to more generalized cluster graphs (or region graphs). Welling (2004) examined methods for automatically choosing the regions for this approximation.

A major advance in our understanding of this general class of approximation algorithms came with the analysis of Yedidia et al. (2000, 2005), showing that these methods can be posed as attempting to maximize an approximate energy functional. This result connected the algorithmic developments in the field with literature on free-energy approximations developed in statistical mechanics (Bethe 1935; Kikuchi 1951). This insight is the basis for our discussion of the Bethe energy functional in section 11.3.6. This result also provided a connection between belief propagation algorithms and other approximation procedures, such as structured variational methods. In addition, it led to the development of new types of approximate inference procedures. These include direct optimization of the Bethe energy functional (Welling and Teh 2001) using gradient-based methods, as well as provably convergent "double loop" variants of belief propagation (Yuille 2002; Heskes et al. 2003). Another class of algorithms combined exact inference on subtrees of the cluster graph within cluster-graph belief propagation (Wainwright et al. 2001, 2002a). This combination leads to faster convergence and introduces new directions for characterizing the

errors of the approximation. (Wainwright and Jordan 2003) provides a comprehensive review of the connections between belief propagation and optimization problems based on the energy functional.

An active area of research is analyzing the convergence properties of generalized belief propagation algorithms. Key directions include the identification of a set of sufficient conditions for the existence of unique local maxima of the Bethe energy functional (Pakzad and Anantharam 2002; Heskes 2002), as well as conditions that ensure a unique convergence point of belief propagation (Tatikonda and Jordan 2002; Ihler et al. 2003; Heskes 2004; Ihler et al. 2005; Mooij and Kappen 2007). Our discussion in section 11.3.4 is based on that of Mooij and Kappen (2007). A related and challenging question is estimating the error of the approximate marginal probabilities; see Leisink and Kappen (2003); Ihler (2007) for some results.

A different trajectory in generalizing the class of belief-propagation algorithms is by introduction of variants of the energy functional that have desired properties. For example, if the energy functional is convex, then it is guaranteed to have a single maximum. An initial convexified free-energy functional was introduced in Wainwright et al. (2002b). This functional has the additional property of providing an upper bound on the partition function. Recently, there has been significant work that provides a more detailed characterization of convex energy functionals (Wainwright and Jordan 2003; Heskes 2006; Wainwright and Jordan 2004; Sontag and Jaakkola 2007). Although the convexity of energy functional implies a unique maximum, it does not generally guarantee that a message passing algorithm will converge. Recent alternative algorithms provide guaranteed convergence for such energy functionals (Heskes 2006; Globerson and Jaakkola 2007a; Hazan and Shashua 2008).

A recent extension is the combination of belief propagation with particle-based methods. The basic idea is to use particle sampling to perform the basic belief propagation steps (Sudderth et al. 2003; Isard 2003). This combination allows us to use cluster-graph belief propagation on networks with continuous non-Gaussian variables, which appear in applications in vision and signal processing; see also section 14.7.

The idea of factored messages appeared in several early contexts (Dechter and Rish 1997; Dechter 1997; Boyen and Koller 1998b; Murphy and Weiss 2001). Similar ideas that involve projection of messages onto "simple" representations (such as Gaussians) are common in the control and tracking literature (Bar-Shalom 1992). The generalization of these ideas in the form of expectation propagation was introduced by Minka (2001b). The connection between expectation propagation and the Bethe energy functional was made by several authors (Minka 2001b; Heskes and Zoeter 2002; Heskes et al. 2005). The connection between expectation propagation and generalized belief propagation in the case of discrete variables is explored by Welling et al. (2005).

One of the early applications of variational methods to probabilistic methods was the application of mean field approximation for Boltzmann machines (Peterson and Anderson 1987). The use of mean field for Markov networks was common in computer vision (see, for example, Li (2001)). This methodology was introduced into the field of directed graphical model by Saul et al. (1996), following ideas that appeared in the context of a more complex architecture of Helmholtz machines (Hinton et al. 1995).

The mean field approximation cannot capture strong dependencies in the posterior distribution. Saul and Jordan (1996) suggested to circumvent this problem by using structured variational methods. These ideas were extended for different forms of approximating distributions

and target networks. Ghahramani and Jordan (1997) used independent hidden Markov chains to approximate factorial HMMs, a specific form of a dynamic Bayesian network. Barber and Wiegerinck (1998) use a Boltzmann machine approximation. Wiegerinck (2000) uses Markov networks and cluster trees as approximate distribution. Xing et al. (2003) describe cluster mean field and suggest efficient implementations. Geiger et al. (2006) further extend Wiegerinck's procedure and introduce efficient propagation schemes to maximize parameters in the approximating distribution.

The idea of using a mixture of mean field approximation was developed by Jaakkola and Jordan (1998). These ideas were extended to general networks with auxiliary variables by El-Hay and Friedman (2001).

Jaakkola and Jordan (1996b,a) introduced local variational approximations to compute both upper bounds and lower bounds of the log-likelihood function (that is, of $\log Z$). They demonstrated these methods by a large scale study of inference in the QMR-DT network (Jaakkola and Jordan 1999), where they show that variational methods are more effective than particle based methods. Additional extension on these methods appeared in Ng and Jordan (2000).

Tutorials discussing the use of structured and local methods appear in Jordan et al. (1998); Jaakkola (2001).

11.8 Exercises

Exercise 11.1

entropy

Show that the derivative of the *entropy* is:

$$\frac{\partial}{\partial Q(\boldsymbol{x})} H_Q(\boldsymbol{X}) = -\ln Q(\boldsymbol{x}) - 1.$$

Exercise 11.2⋆

local consistency polytope

Consider the set of locally consistent distributions, as in the *local consistency polytope* of equation (11.16).

marginal polytope

a. For a cluster graph $\mathcal{U}$ that is a clique tree $\mathcal{T}$ (satisfying the running intersection property), show that the set of distributions satisfying these constraints is precisely the *marginal polytope* — the set of legal distributions Q that can be represented over $\mathcal{T}$.

b. Show that, for a cluster graph that is not a tree, there are parameterizations that satisfy these constraints but are not marginals of any legal probability distribution.

Exercise 11.3⋆

In the text, we showed that CTree-Optimize has a unique global optimum. Show that it also has a unique fixed point.

Exercise 11.4⋆

Consider the network of figure 11.1c. We have shown that

$$P_{\mathcal{T}}(A,B,C,D) \propto P_{\Phi}(A,B,C,D)\frac{\mu_{3,4}[D]\mu_{1,4}[A]}{\beta_4(A,D)},$$

where $\mathcal{T}$ is the cluster tree we get if we remove $C_4 = \{A, D\}$. Show how to use this result on the *residual* to bound the error in the estimation of marginals in this cluster graph.

cluster graph residual

Exercise 11.5⋆

Prove proposition 11.2.