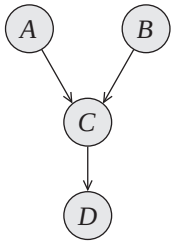


Figure 19.6 A simple network used to illustrate learning algorithms for missing data

When $\boldsymbol{o}$ is inconsistent with x or $\boldsymbol{u}$, then the gradient is 0, since the probability $P(x, \boldsymbol{u}, \boldsymbol{o})$ is 0 in this case. When $\boldsymbol{o}$ is consistent with x and $\boldsymbol{u}$, the gradient is the ratio between the probability $P(x, \boldsymbol{u}, \boldsymbol{o})$ and the parameter $P(x \mid \boldsymbol{u})$. Intuitively, this ratio takes into account the weight of the cases where $P(x \mid \boldsymbol{u})$ is "used" in the computation of $P(\boldsymbol{o})$. Increasing $P(x \mid \boldsymbol{u})$ by a small amount will increase the probability of these cases by a multiplicative factor.

The lemma does not deal with the case where $P(x \mid \boldsymbol{u}) = 0$, since we cannot divide by 0. Note, however, that the proof shows that this division is mainly a neat manner of writing the product of all terms *except* $P(x \mid \boldsymbol{u})$. Thus, even in this extreme case we can use a similar proof to compute the gradient, although writing the term explicitly is less elegant. Since in learning we usually try to avoid extreme parameter assignments, we will continue our discussion with the assumption that $P(x \mid \boldsymbol{u}) > 0$.

An immediate consequence of lemma 19.1 is the form of the gradient of the log-likelihood function.

Theorem 19.2

Let $\mathcal{G}$ be a Bayesian network structure over $\mathcal{X}$, and let $\mathcal{D} = \{\boldsymbol{o}[1], \ldots, \boldsymbol{o}[M]\}$ be a partially observable data set. Let X be a variable and $\boldsymbol{U}$ its parents in $\mathcal{G}$. Then

$$\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(x \mid \boldsymbol{u})} = \frac{1}{P(x \mid \boldsymbol{u})} \sum_{m=1}^{M} P(x, \boldsymbol{u} \mid \boldsymbol{o}[m], \boldsymbol{\theta}).$$

The proof is left as an exercise (exercise 19.5).

chain rule of derivatives

This theorem provides the form of the gradient for table-CPDs. For other CPDs, such as noisy-or CPDs, we can use the *chain rule of derivatives* to compute the gradient. Suppose that the CPD entries of $P(X \mid \boldsymbol{U})$ are written as functions of some set of parameters $\boldsymbol{\theta}$. Then, for a specific parameter $\theta \in \boldsymbol{\theta}$, we have

$$\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial \theta} = \sum_{x, \boldsymbol{u}} \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(x \mid \boldsymbol{u})} \frac{\partial P(x \mid \boldsymbol{u})}{\partial \theta},$$

where the first term is the derivative of the log-likelihood function when parameterized in terms of the table-CPDs induced by $\boldsymbol{\theta}$. For structured CPDs, we can use this formula to compute the gradient with respect to the CPD parameters. For some CPDs, however, this may not be the most efficient way of computing these gradients; see exercise 19.4.

19.2.1.2 An Example

We consider a simple example to clarify the concept. Consider the network shown in figure 19.6, and a partially specified data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0 \rangle$.

We want to compute the gradient of one family of parameters $P(D \mid c^0)$ given the observation $\boldsymbol{o}$. Using theorem 19.2, we know that

$$\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^0)} = \frac{P(d^0, c^0 \mid \boldsymbol{o})}{P(d^0 \mid c^0)},$$

and similarly for other values of D and C.

Assume that our current $\boldsymbol{\theta}$ is:

$$\begin{aligned}
\boldsymbol{\theta}_{a^1} &= 0.3 \\
\boldsymbol{\theta}_{b^1} &= 0.9 \\
\boldsymbol{\theta}_{c^1 \mid a^0, b^0} &= 0.83 \\
\boldsymbol{\theta}_{c^1 \mid a^0, b^1} &= 0.09 \\
\boldsymbol{\theta}_{c^1 \mid a^1, b^0} &= 0.6 \\
\boldsymbol{\theta}_{c^1 \mid a^1, b^1} &= 0.2 \\
\boldsymbol{\theta}_{d^1 \mid c^0} &= 0.1 \\
\boldsymbol{\theta}_{d^1 \mid c^1} &= 0.8.
\end{aligned}$$

In this case, the probabilities of the four data cases that are consistent with $\boldsymbol{o}$ are

$$\begin{aligned}
P(\langle a^1, b^1, c^1, d^0 \rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.2 = 0.0108 \\
P(\langle a^1, b^1, c^0, d^0 \rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.9 = 0.1944 \\
P(\langle a^1, b^0, c^1, d^0 \rangle) &= 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.2 = 0.0036 \\
P(\langle a^1, b^0, c^0, d^0 \rangle) &= 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.9 = 0.0108.
\end{aligned}$$

To compute the *posterior* probability of these instances given the partial observation $\boldsymbol{o}$, we divide the probability of each instance with the total probability, which is 0.2196, that is,

$$\begin{aligned}
P(\langle a^1, b^1, c^1, d^0 \rangle \mid \boldsymbol{o}) &= 0.0492 \\
P(\langle a^1, b^1, c^0, d^0 \rangle \mid \boldsymbol{o}) &= 0.8852 \\
P(\langle a^1, b^0, c^1, d^0 \rangle \mid \boldsymbol{o}) &= 0.0164 \\
P(\langle a^1, b^0, c^0, d^0 \rangle \mid \boldsymbol{o}) &= 0.0492.
\end{aligned}$$

Using these computations, we see that

$$\begin{aligned}
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^1 \mid c^0)} &= \frac{P(d^1, c^0 \mid \boldsymbol{o})}{P(d^1 \mid c^0)} = \frac{0}{0.1} = 0 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^0)} &= \frac{P(d^0, c^0 \mid \boldsymbol{o})}{P(d^0 \mid c^0)} = \frac{0.8852 + 0.0492}{0.9} = 1.0382 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^1 \mid c^1)} &= \frac{P(d^1, c^1 \mid \boldsymbol{o})}{P(d^1 \mid c^1)} = \frac{0}{0.8} = 0 \\
\frac{\partial \log P(\boldsymbol{o})}{\partial P(d^0 \mid c^1)} &= \frac{P(d^0, c^1 \mid \boldsymbol{o})}{P(d^0 \mid c^1)} = \frac{0.0492 + 0.0164}{0.2} = 0.328.
\end{aligned}$$

These computations show that we can increase the probability of the observations $\boldsymbol{o}$ by either increasing $P(d^0 \mid c^0)$ or $P(d^0 \mid c^1)$. Moreover, increasing the former parameter will lead to a bigger change in the probability of $\boldsymbol{o}$ than a similar increase in the latter parameter.

Now suppose we have an observation $\boldsymbol{o}' = \langle a^0, ?, ?, d^1 \rangle$. We can repeat the same computation as before and see that

$$\begin{aligned}
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^1 \mid c^0)} &= \frac{P(d^1, c^0 \mid \boldsymbol{o}')}{P(d^1 \mid c^0)} = \frac{0.2836}{0.1} = 2.8358 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^0 \mid c^0)} &= \frac{P(d^0, c^0 \mid \boldsymbol{o}')}{P(d^0 \mid c^0)} = \frac{0}{0.9} = 0 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^1 \mid c^1)} &= \frac{P(d^1, c^1 \mid \boldsymbol{o}')}{P(d^1 \mid c^1)} = \frac{0.7164}{0.8} = 0.8955 \\
\frac{\partial \log P(\boldsymbol{o}')}{\partial P(d^0 \mid c^1)} &= \frac{P(d^0, c^1 \mid \boldsymbol{o}')}{P(d^0 \mid c^1)} = \frac{0}{0.2} = 0.
\end{aligned}$$

Suppose our data set consists only of these two instances. The gradient of the log-likelihood function is the sum of the gradient with respect to the two instances. We get that

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} &= 2.8358 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} &= 1.0382 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^1)} &= 0.8955 \\
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^1)} &= 0.328.
\end{aligned}$$

Note that all the gradients are nonnegative. Thus, increasing any of the parameters in the CPD $P(D \mid C)$ will increase the likelihood of the data. It is clear, however, that we cannot increase both $P(d^1 \mid c^0)$ and $P(d^0 \mid c^0)$ at the same time, since this will lead to an illegal conditional probability. One way of solving this is to use a single parameter $\theta_{d^1 \mid c^0}$ and write

$$P(d^1 \mid c^0) = \theta_{d^1 \mid c^0} \qquad P(d^0 \mid c^0) = 1 - \theta_{d^1 \mid c^0}.$$

Using the chain rule of conditional probabilities, we have that

$$\begin{aligned}
\frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial \theta_{d^1 \mid c^0}} &= \frac{\partial P(d^1 \mid c^0)}{\partial \theta_{d^1 \mid c^0}} \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} + \frac{\partial P(d^0 \mid c^0)}{\partial \theta_{d^1 \mid c^0}} \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} \\
&= \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^1 \mid c^0)} - \frac{\partial \ell(\boldsymbol{\theta} : \mathcal{D})}{\partial P(d^0 \mid c^0)} \\
&= 2.8358 - 1.0382 = 1.7976.
\end{aligned}$$

Thus, in this case, we prefer to increase $P(d^1 \mid c^0)$ and decrease $P(d^0 \mid c^0)$, since the resulting increase in the probability of $\boldsymbol{o}'$ will be larger than the decrease in the probability of $\boldsymbol{o}$.

Algorithm 19.1 Computing the gradient in a network with table-CPDs

Procedure Compute-Gradient (
  $\mathcal{G}$, // Bayesian network structure over $X_1, \ldots, X_n$
  $\boldsymbol{\theta}$, // Set of parameters for $\mathcal{G}$
  $\mathcal{D}$ // Partially observed data set
)

// Initialize data structures
for each $i = 1, \ldots, n$
  for each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
    $\bar{M}[x_i, \boldsymbol{u}_i] \leftarrow 0$
  // Collect probabilities from all instances
for each $m = 1 \ldots M$
  Run clique tree calibration on $\langle \mathcal{G}, \boldsymbol{\theta} \rangle$ using evidence $\boldsymbol{o}[m]$
  for each $i = 1, \ldots, n$
    for each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
      $\bar{M}[x_i, \boldsymbol{u}_i] \leftarrow \bar{M}[x_i, \boldsymbol{u}_i] + P(x_i, \boldsymbol{u}_i \mid \boldsymbol{o}[m])$
  // Compute components of the gradient vector
for each $i = 1, \ldots, n$
  for each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
    $\delta_{x_i \mid \boldsymbol{u}_i} \leftarrow \frac{1}{\theta_{x_i \mid \boldsymbol{u}_i}} \bar{M}[x_i, \boldsymbol{u}_i]$
return $\{\delta_{x_i, \mid \boldsymbol{u}_i} : \forall i = 1, \ldots, n, \forall (x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})\}$

19.2.1.3 Gradient Ascent Algorithm

We now generalize these ideas to case of an arbitrary network. For now we focus on the case of table-CPDs. In this case, the gradient is given by theorem 19.2. To compute the gradient for the CPD $P(X \mid \boldsymbol{U})$, we need to compute the joint probability of x and $\boldsymbol{u}$ relative to our current parameter setting $\boldsymbol{\theta}$ and each observed instance $\boldsymbol{o}[m]$. In other words, we need to compute the joint distribution $P(X[m], \boldsymbol{U}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$ for each m. We can do this by running an inference procedure for each data case. Importantly, **we can do all of the required inference for each data case using one clique tree calibration, since the family preservation property guarantees that X and its parents $\boldsymbol{U}$ will be together in some clique in the tree.** Procedure Compute-Gradient, shown in algorithm 19.1, performs these computations.

Once we have a procedure for computing the gradient, it seems that we can simply plug it into a standard package for gradient ascent and optimize the parameters. As we have illustrated, however, there is one issue that we need to deal with. It is not hard to confirm that all components of the gradient vector are nonnegative. This is natural, since increasing each of the parameters will lead to higher likelihood. Thus, a step in the gradient direction will *increase* all the parameters. Remember, however, that we want to ensure that our parameters describe a legal probability distribution. That is, the parameters for each conditional probability are nonnegative and sum to one.

In the preceding example, we saw one approach that works well when we have binary variables. In general networks, there are two common approaches to deal with this issue. The first approach is to modify the gradient ascent procedure we use (for example, conjugate gradient) to respect these constraints. First, we must project each gradient vector onto the hyperplane that satisfies the linear constraints on the parameters; this step is fairly straightforward (see exercise 19.6). Second, we must ensure that parameters are nonnegative; this requires restricting possible steps to avoid stepping out of the allowed bounds.

reparameterization

The second approach is to *reparameterize* the problem. Suppose we introduce new parameters $\lambda_{x|\boldsymbol{u}}$, and define

$$P(x \mid \boldsymbol{u}) = \frac{e^{\lambda_{x|\boldsymbol{u}}}}{\sum_{x' \in Val(X)} e^{\lambda_{x'|\boldsymbol{u}}}}, \tag{19.3}$$

for each X and its parents $\boldsymbol{U}$. Now, any choice of values for the λ parameters will lead to legal conditional probabilities. We can compute the gradient of the log-likelihood with respect to the λ parameters using the chain rule of partial derivatives, and then use standard (unmodified) conjugate gradient ascent procedure. See exercise 19.7.

Lagrange multipliers

Another way of dealing with the constraints implied by conditional probabilities is to use the method of *Lagrange multipliers*, reviewed in appendix A.5.3. Applying this method to the optimization of the log-likelihood leads to the method we discuss in the next section, and we defer this discussion; see also exercise 19.8.

Having dealt with this subtlety, we can now apply any gradient ascent procedure to find a local maximum of the likelihood function. As discussed, in most missing value problems, the likelihood function has many local maxima. Unfortunately, gradient ascent procedures are guaranteed to achieve only a local maximum of the function. Many of the techniques we discussed earlier in the book can be used to avoid local maxima and increase our chances of finding a global maximum, or at least a better local maximum: the general-purpose methods of appendix A.4.2, such as multiple random starting points, or applying random perturbations to convergence points; and the more specialized data perturbation methods of algorithm 18.1.

19.2.2 Expectation Maximization (EM)

expectation maximization

An alternative algorithm for optimizing a likelihood function is the *expectation maximization* algorithm. Unlike gradient ascent, EM is not a general-purpose algorithm for nonlinear function optimization. Rather, it is tailored specifically to optimizing likelihood functions, attempting to build on the tools we had for solving the problem with complete data.

19.2.2.1 Intuition

Recall that when learning from complete data, we can collect sufficient statistics for each CPD. We can then estimate parameters that maximize the likelihood with respect to these statistics. As we saw, in the case of missing data, we do not have access to the full sufficient statistics. Thus, we cannot use the same strategy for our problem. For example, in a simple $X \rightarrow Y$ network, if we see the training instance $\langle ?, y^1 \rangle$, then we do not know whether to count this instance toward the count $M[x^1, y^1]$ or toward the count $M[x^0, y^1]$.

A simple approach is to "fill in" the missing values arbitrarily. For example, there are strategies that fill in missing values with "default values" (say *false*) or by randomly choosing a value. Once we fill in all the missing values, we can use standard, complete data learning procedure. Such approaches are called called *data imputation* methods in statistics.

data imputation

The problem with such an approach is that the procedure we use for filling in the missing values introduces a bias that will be reflected in the parameters we learn. For example, if we fill all missing values with *false*, then our estimate will be skewed toward higher (conditional) probability of *false*. Similarly, if we use a randomized procedure for filling in values, then the probabilities we estimate will be skewed toward the distribution from which we sample missing values. This might be better than a skew toward one value, but it still presents a problem. Moreover, when we consider learning with hidden variables, it is clear that an imputation procedure will not help us. The values we fill in for the hidden variable are conditionally independent from the values of the other variables, and thus, using the imputed values, we will not learn any dependencies between the hidden variable and the other variables in the network.

A different approach to filling in data takes the perspective that, when learning with missing data, we are actually trying to solve two problems at once: learning the parameters, and hypothesizing values for the unobserved variables in each of the data cases. Each of these tasks is fairly easy when we have the solution to the other. Given complete data, we have the statistics, and we can estimate parameters using the MLE formulas we discussed in chapter 17. Conversely, given a choice of parameters, we can use probabilistic inference to hypothesize the likely values (or the distribution over possible values) for unobserved variables. Unfortunately, because we have neither, the problem is difficult.

The EM algorithm solves this "chicken and egg" problem using a bootstrap approach. We start out with some arbitrary starting point. This can be either a choice of parameters, or some initial assignment to the hidden variables; these assignments can be either random, or selected using some heuristic approach. Assuming, for concreteness, that we begin with a parameter assignment, the algorithm then repeats two steps. First, we use our current parameters to *complete* the data, using probabilistic inference. We then treat the completed data as if it were observed and learn a new set of parameters.

data completion

More precisely, suppose we have a guess $\boldsymbol{\theta}^0$ about the parameters of the network. The resulting model defines a joint distribution over all the variables in the domain. Given a partial instance, we can compute the posterior (using our putative parameters) over all possible assignments to the missing values in that instance. The EM algorithm uses this probabilistic completion of the different data instances to estimate the *expected* value of the sufficient statistics. It then finds the parameters $\boldsymbol{\theta}^1$ that maximize the likelihood with respect to these statistics.

Somewhat surprisingly, this sequence of steps provably improves our parameters. In fact, as we will prove formally, unless our parameters have not changed due to these steps (such that $\boldsymbol{\theta}^0 = \boldsymbol{\theta}^1$), our new parameters $\boldsymbol{\theta}^1$ necessarily have a higher likelihood than $\boldsymbol{\theta}^0$. But now we can iteratively repeat this process, using $\boldsymbol{\theta}^1$ as our new starting point. Each of these operations can be thought of as taking an "uphill" step in our search space. More precisely, we will show (under very benign assumptions) that: each iteration is guaranteed to improve the log-likelihood function; that this process is guaranteed to converge; and that the convergence point is a fixed point of the likelihood function, which is essentially always a local maximum. Thus, the guarantees of the EM algorithm are similar to those of gradient ascent.

19.2.2.2 An Example

We start with a simple example to clarify the concepts. Consider the simple network shown in figure 19.6. In the fully observable case, our maximum likelihood parameter estimate for the parameter $\hat{\theta}_{d^1|c^0}$ is:

$$\hat{\theta}_{d^1|c^0} = \frac{M[d^1, c^0]}{M[c^0]} = \frac{\sum_{m=1}^{M} \mathbf{1}\{\xi[m]\langle D, C\rangle = \langle d^1, c^0\rangle\}}{\sum_{m=1}^{M} \mathbf{1}\{\xi[m]\langle C\rangle = c^0\}},$$

where $\xi[m]$ is the m'th training example. In the fully observable case, we knew exactly whether the indicator variables were 0 or 1. Now, however, we do not have complete data cases, so we no longer know the value of the indicator variables.

Consider a partially specified data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$. There are four possible instantiations to the missing variables B, C which could have given rise to this partial data case: $\langle b^1, c^1\rangle$, $\langle b^1, c^0\rangle$, $\langle b^0, c^1\rangle$, $\langle b^0, c^0\rangle$. We do not know which of them is true, or even which of them is more likely.

However, assume that we have some estimate $\boldsymbol{\theta}$ of the values of the parameters in the model. In this case, we can compute how likely each of these completions is, given our distribution. That is, we can define a distribution $Q(B, C) = P(B, C \mid \boldsymbol{o}, \boldsymbol{\theta})$ that induces a distribution over the four data cases. For example, if our parameters $\boldsymbol{\theta}$ are:

$$\begin{array}{llll} \boldsymbol{\theta}_{a^1} & = 0.3 & \boldsymbol{\theta}_{b^1} & = 0.9 \\ \boldsymbol{\theta}_{d^1|c^0} & = 0.1 & \boldsymbol{\theta}_{d^1|c^1} & = 0.8 \\ \boldsymbol{\theta}_{c^1|a^0,b^0} & = 0.83 & \boldsymbol{\theta}_{c^1|a^1,b^0} & = 0.6 \\ \boldsymbol{\theta}_{c^1|a^0,b^1} & = 0.09 & \boldsymbol{\theta}_{c^1|a^1,b^1} & = 0.2, \end{array}$$

then $Q(B, C) = P(B, C \mid a^1, d^0, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned} Q(\langle b^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.2/0.2196 = 0.0492 \\ Q(\langle b^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.9/0.2196 = 0.8852 \\ Q(\langle b^0, c^1\rangle) &= 0.3 \cdot 0.1 \cdot 0.6 \cdot 0.2/0.2196 = 0.0164 \\ Q(\langle b^0, c^0\rangle) &= 0.3 \cdot 0.1 \cdot 0.4 \cdot 0.9/0.2196 = 0.0492, \end{aligned}$$

where 0.2196 is a normalizing factor, equal to $P(a^1, d^0 \mid \boldsymbol{\theta})$.

If we have another example $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then $Q'(A, C) = P(A, C \mid b^1, d^1, \boldsymbol{\theta})$ is defined as:

$$\begin{aligned} Q'(\langle a^1, c^1\rangle) &= 0.3 \cdot 0.9 \cdot 0.2 \cdot 0.8/0.1675 = 0.2579 \\ Q'(\langle a^1, c^0\rangle) &= 0.3 \cdot 0.9 \cdot 0.8 \cdot 0.1/0.1675 = 0.1290 \\ Q'(\langle a^0, c^1\rangle) &= 0.7 \cdot 0.9 \cdot 0.09 \cdot 0.8/0.1675 = 0.2708 \\ Q'(\langle a^0, c^0\rangle) &= 0.7 \cdot 0.9 \cdot 0.91 \cdot 0.1/0.1675 = 0.3423. \end{aligned}$$

weighted data instances

Intuitively, now that we have estimates for how likely each of the cases is, we can treat these estimates as truth. That is, we view our partially observed data case $\langle a^1, ?, ?, d^0\rangle$ as consisting of four complete data cases, each of which has some *weight* lower than 1. The weights correspond to our estimate, based on our current parameters, on how likely is this particular completion of the partial instance. (This approach is somewhat reminiscent of the weighted particles in the likelihood weighting algorithm.) Importantly, as we will discuss, we do

not usually explicitly generate these completed data cases; however, this perspective is the basis for the more sophisticated methods.

More generally, let $\boldsymbol{H}[m]$ denote the variables whose values are missing in the data instance $\boldsymbol{o}[m]$. We now have a data set $\mathcal{D}^+$ consisting of

$$\cup_m \{\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle \ : \ \boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])\},$$

where each data case $\langle \boldsymbol{o}[m], \boldsymbol{h}[m]\rangle$ has weight $Q(\boldsymbol{h}[m]) = P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$.

expected sufficient statistics

We can now do standard maximum likelihood estimation using these completed data cases. We compute the *expected sufficient statistics*:

$$\bar{M}_{\boldsymbol{\theta}}[\boldsymbol{y}] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \mathbf{1}\{\xi[m]\langle \boldsymbol{Y}\rangle = \boldsymbol{y}\}.$$

We then use these expected sufficient statistics as if they were real in the MLE formula. For example:

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{\bar{M}_{\boldsymbol{\theta}}[d^1, c^0]}{\bar{M}_{\boldsymbol{\theta}}[c^0]}.$$

In our example, suppose the data consist of the two instances $\boldsymbol{o} = \langle a^1, ?, ?, d^0\rangle$ and $\boldsymbol{o}' = \langle ?, b^1, ?, d^1\rangle$. Then, using the calculated Q and Q' from above, we have that

$$\begin{aligned}
\bar{M}_{\boldsymbol{\theta}}[d^1, c^0] &= Q'(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\
&= 0.1290 + 0.3423 = 0.4713 \\
\bar{M}_{\boldsymbol{\theta}}[c^0] &= Q(\langle b^1, c^0\rangle) + Q(\langle b^0, c^0\rangle) + Q'(\langle a^1, c^0\rangle) + Q'(\langle a^0, c^0\rangle) \\
&= 0.8852 + 0.0492 + 0.1290 + 0.3423 = 1.4057.
\end{aligned}$$

Thus, in this example, using these particular parameters to compute expected sufficient statistics, we get

$$\tilde{\boldsymbol{\theta}}_{d^1|c^0} = \frac{0.4713}{1.4057} = 0.3353.$$

Note that this estimate is quite different from the parameter $\theta_{d^1|c^0} = 0.1$ that we used in our estimate of the expected counts. The initial parameter and the estimate are different due to the incorporation of the observations in the data.

This intuition seems nice. However, it may require an unreasonable amount of computation. To compute the expected sufficient statistics, we must sum over all the completed data cases. The number of these completed data cases is much larger than the original data set. For each $\boldsymbol{o}[m]$, the number of completions is exponential in the number of missing values. Thus, if we have more than few missing values in an instances, an implementation of this approach will not be able to finish computing the expected sufficient statistics.

Fortunately, it turns out that there is a better approach to computing the expected sufficient statistic than simply summing over all possible completions. Let us reexamine the formula for an expected sufficient statistic, for example, $\bar{M}_{\boldsymbol{\theta}}[c^1]$. We have that

$$\bar{M}_{\boldsymbol{\theta}}[c^1] = \sum_{m=1}^{M} \sum_{\boldsymbol{h}[m] \in Val(\boldsymbol{H}[m])} Q(\boldsymbol{h}[m]) \mathbf{1}\{\xi[m]\langle C\rangle = c^1\}.$$

Let us consider the internal summation, say for a data case $\boldsymbol{o} = \langle a^1, ?, ?, d^0 \rangle$. We have four possible completions, as before, but we are only summing over the two that are consistent with c^1, that is, $Q(b^1, c^1) + Q(b^0, c^1)$. This expression is equal to $Q(c^1) = P(c^1 \mid a^1, d^0, \boldsymbol{\theta}) = P(c^1 \mid \boldsymbol{o}[1], \boldsymbol{\theta})$. This idea clearly generalizes to our other data cases. Thus, we have that

$$\bar{M}_{\boldsymbol{\theta}}[c^1] = \sum_{m=1}^{M} P(c^1 \mid \boldsymbol{o}[m], \boldsymbol{\theta}).$$

Now, recall our formula for sufficient statistics in the fully observable case:

$$M[c^1] = \sum_{m=1}^{M} \boldsymbol{I}\{\xi[m]\langle C \rangle = c^1\}.$$

Our new formula is identical, except that we have substituted our indicator variable — either 0 or 1 — with a probability that is somewhere between 0 and 1. Clearly, if in a certain data case we get to observe C, the indicator variable and the probability are the same. Thus, we can view the expected sufficient statistics as filling in soft estimates for hard data when the hard data are not available.

We stress that we use *posterior* probabilities in computing expected sufficient statistics. Thus, although our choice of $\boldsymbol{\theta}$ clearly influences the result, the data also play a central role. This is in contrast to the probabilistic completion we discussed earlier that used a prior probability to fill in values, regardless of the evidence on the other variables in the same instances.

19.2.2.3 The EM Algorithm for Bayesian Networks

We now present the basic EM algorithm and describe the guarantees that it provides.

Networks with Table-CPDs Consider the application of the EM algorithm to a general Bayesian network with table-CPDs. Assume that the algorithm begins with some initial parameter assignment $\boldsymbol{\theta}^0$, which can be chosen either randomly or using some other approach. (The case where we begin with some assignment to the missing data is analogous.) The algorithm then repeatedly executes the following phases, for $t = 0, 1, \ldots$

expected sufficient statistics

Expectation (E-step): The algorithm uses the current parameters $\boldsymbol{\theta}^t$ to compute the *expected sufficient statistics*.

- For each data case $\boldsymbol{o}[m]$ and each family $X, \boldsymbol{U}$, compute the joint distribution $P(X, \boldsymbol{U} \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t)$.
- Compute the expected sufficient statistics for each $x, \boldsymbol{u}$ as:

$$\bar{M}_{\boldsymbol{\theta}^t}[x, \boldsymbol{u}] = \sum_{m} P(x, \boldsymbol{u} \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t).$$

E-step

This phase is called the *E-step* (*expectation step*) because the counts used in the formula are the expected sufficient statistics, where the expectation is with respect to the current set of parameters.

Algorithm 19.2 Expectation-maximization algorithm for BN with table-CPDs

```
Procedure Compute-ESS (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ,    // Set of parameters for G
    D     // Partially observed data set
)
    // Initialize data structures
    for each i = 1, ..., n
      for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
        M̄[x_i, u_i] ← 0
    // Collect probabilities from all instances
    for each m = 1 ... M
      Run inference on ⟨G, θ⟩ using evidence o[m]
      for each i = 1, ..., n
        for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
          M̄[x_i, u_i] ← M̄[x_i, u_i] + P(x_i, u_i | o[m])
    return {M̄[x_i, u_i] : ∀i = 1, ..., n, ∀x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)}

Procedure Expectation-Maximization (
    G,    // Bayesian network structure over X_1, ..., X_n
    θ^0,  // Initial set of parameters for G
    D     // Partially observed data set
)
    for each t = 0, 1 ..., until convergence
        // E-step
      {M̄_t[x_i, u_i]} ← Compute-ESS(G, θ^t, D)
        // M-step
      for each i = 1, ..., n
        for each x_i, u_i ∈ Val(X_i, Pa_{X_i}^G)
          θ^{t+1}_{x_i|u_i} ← M̄_t[x_i, u_i] / M̄_t[u_i]
    return θ^t
```

Maximization (M-step): Treat the expected sufficient statistics as observed, and perform maximum likelihood estimation, with respect to them, to derive a new set of parameters. In other words, set

$$\boldsymbol{\theta}^{t+1}_{x|\boldsymbol{u}} = \frac{\bar{M}_{\boldsymbol{\theta}^t}[x, \boldsymbol{u}]}{\bar{M}_{\boldsymbol{\theta}^t}[\boldsymbol{u}]}.$$

M-step

This phase is called the *M-step* (*maximization step*), because we are maximizing the likelihood relative to the expected sufficient statistics.

A formal version of the algorithm is shown fully in algorithm 19.2.

The maximization step is straightforward. The more difficult step is the expectation step. How do we compute expected sufficient statistics? We must resort to Bayesian network inference over the network $\langle \mathcal{G}, \boldsymbol{\theta}^t \rangle$. Note that, as in the case of gradient ascent, the only expected sufficient statistics that we need involve a variable and its parents. Although one can use a variety of different inference methods to perform the inference task required for the E-step, we can, as in the case of gradient ascent, use the clique tree or cluster graph algorithm. Recall that the family-preservation property guarantees that X and its parents $\boldsymbol{U}$ will be together in some cluster in the tree or graph. **Thus, once again, we can do all of the required inference for each data case using one run of message-passing calibration.**

General Exponential Family ⋆ The same idea generalizes to other distributions where the likelihood has sufficient statistics, in particular, all models in the *exponential family* (see definition 8.1). Recall that such families have a sufficient statistic function $\tau(\xi)$ that maps a complete instance to a vector of sufficient statistics. When learning parameters of such a model, we can summarize the data using the sufficient statistic function τ. For a complete data set $\mathcal{D}^+$, we define

exponential family

$$\tau(\mathcal{D}^+) = \sum_m \tau(\boldsymbol{o}[m], \boldsymbol{h}[m]).$$

We can now define the same E and M-steps described earlier for this more general case.

E-step

Expectation (*E-step*): For each data case $\boldsymbol{o}[m]$, the algorithm uses the current parameters $\boldsymbol{\theta}^t$ to define a model, and a posterior distribution:

$$Q(\boldsymbol{H}[m]) = P(\boldsymbol{H}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t).$$

expected sufficient statistics

It then uses inference in this distribution to compute the *expected sufficient statistics*:

$$\boldsymbol{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_m \boldsymbol{E}_Q[\tau(\boldsymbol{o}[m], \boldsymbol{h}[m])]. \tag{19.4}$$

M-step

Maximization (*M-step*): As in the case of table-CPDs, once we have the expected sufficient statistics, the algorithm treats them as if they were real and uses them as the basis for maximum likelihood estimation, using the appropriate form of the ML estimator for this family.

Convergence Results Somewhat surprisingly, this simple algorithm can be shown to have several important properties. We now state somewhat simplified versions of the relevant results, deferring a more precise statement to the next section.

The first result states that each iteration is guaranteed to improve the log-likelihood of the current set of parameters.

Theorem 19.3

During iterations of the EM procedure of algorithm 19.2, we have

$$\ell(\boldsymbol{\theta}^t : \mathcal{D}) \leq \ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}).$$

Thus, the EM procedure is constantly increasing the log-likelihood objective function. Because the objective function can be shown to be bounded (under mild assumptions), this procedure is guaranteed to converge. By itself, this result does not imply that we converge to a maximum of the objective function. Indeed, this result is only "almost true":

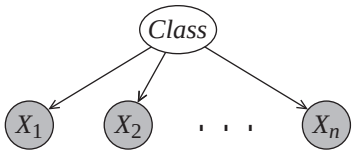


Figure 19.7 The naive Bayes clustering model. In this model each observed variables X_i is independent of the other observed variables given the value of the (unobserved) cluster variable C.

Theorem 19.4 *Suppose that θ^t is such that $\theta^{t+1} = \theta^t$ during EM, and θ^t is also an interior point of the allowed parameter space. Then θ^t is a stationary point of the log-likelihood function.*

This result shows that EM converges to a stationary point of the likelihood function. Recall that a stationary point can be a local maximum, local minimum, or a saddle point. Although it seems counterintuitive that by taking upward steps we reach a local minimum, it is possible to construct examples where EM converges to such a point. **However, nonmaximal convergence points can only be reached from very specific starting points, and are moreover not stable, since even small perturbations to the parameters are likely to move the algorithm away from this point. Thus, in practice, EM generally converges to a local maximum of the likelihood function.**

19.2.2.4 Bayesian Clustering Using EM

clustering

One important application of learning with incomplete data, and EM in particular, is to the problem of *clustering*. Here, we have a set of data points in some feature space $\boldsymbol{X}$. Let us even assume that they are fully observable. We want to classify these data points into coherent categories, that is, categories of points that seem to share similar statistical properties.

Bayesian clustering

The *Bayesian clustering* paradigm views this task as a learning problem with a single hidden variable C that denotes the category or class from which an instance comes. Each class is associated with a probability distribution over the features of the instances in the class. In most cases, we assume that the instances in each class c come from some coherent, fairly simple, distribution. In other words, we postulate a particular form for the *class-conditional distribution* $P(\boldsymbol{x} \mid c)$. For example, in the case of real-valued data, we typically assume that the class-conditional distribution is a multivariate Gaussian (see section 7.1). In discrete settings, we typically assume that the class-conditional distribution is a naive Bayes structure (section 3.1.3), where each feature is independent of the rest given the class variable. Overall, this approach views the data as coming from a *mixture distribution* and attempts to use the hidden variable to separate out the mixture into its components.

mixture distribution

naive Bayes

Suppose we consider the case of a *naive Bayes* model (figure 19.7) where the hidden class variable is the single parent of all the observed feature. In this particular learning scenario, the E-step involves computing the probability of different values of the class variables for each instance. Thus, we can think of EM as performing a soft classification of the instances, that is, each data instance belongs, to some degree, to multiple classes.

In the M-step we compute the parameters for the CPDs in the form $P(X \mid C)$ and the prior

$P(C)$ over the classes. These estimates depends on our expected sufficient statistics. These are:

$$\begin{aligned}\bar{M}_{\boldsymbol{\theta}}[c] &\leftarrow \sum_m P(c \mid x_1[m], \ldots, x_n[m], \boldsymbol{\theta}^t) \\ \bar{M}_{\boldsymbol{\theta}}[x_i, c] &\leftarrow \sum_m P(c, x_i \mid x_1[m], \ldots, x_n[m], \boldsymbol{\theta}^t).\end{aligned}$$

We see that an instance helps determine the parameters for all of the classes that it participates in (that is, ones where $P(c \mid \boldsymbol{x}[m])$ is bigger than 0). Stated a bit differently, each instance "votes" about the parameters of each cluster by contributing to the statistics of the conditional distribution given that value of the cluster variable. However, the weight of this vote depends on the probability with which we assign the instance to the particular cluster.

Once we have computed the expected sufficient statistics, the M-step is, as usual, simple. The parameters for the class variable CPD are

$$\theta_c^{t+1} \leftarrow \frac{\bar{M}_{\boldsymbol{\theta}}[c]}{M},$$

and for the conditional CPD are

$$\theta_{x_i|c}^{t+1} \leftarrow \frac{\bar{M}_{\boldsymbol{\theta}}[x_i, c]}{\bar{M}_{\boldsymbol{\theta}}[c]}.$$

We can develop similar formulas for the case where some of the observed variables are continuous, and we use a conditional Gaussian distribution (a special case of definition 5.15) to model the CPD $P(X_i \mid C)$. The application of EM to this specific model results in a simple and efficient algorithm.

We can think of the clustering problem with continuous observations from a geometrical perspective, where each observed variable X_i represents one coordinate, and instances correspond to points. The parameters in this case represent the distribution of coordinate values in each of the classes. Thus, each class corresponds to a *cloud* of points in the input data. In each iteration, we reestimate the location of these clouds. In general, depending on the particular starting point, EM will proceed to assign each class to a dense cloud.

The EM algorithm for clustering uses a "soft" cluster assignment, allowing each instance to contribute part of its weight to multiple clusters, proportionately to its probability of belonging to each of them. As another alternative, we can consider "hard clustering," where each instance contributes all of its weight to the cluster to which it is most likely to belong. This variant, called *hard-assigment EM* proceeds by performing the following steps.

hard-assignment EM

- Given parameters $\boldsymbol{\theta}^t$, we assign $c[m] = \arg\max_c P(c \mid \boldsymbol{x}[m], \boldsymbol{\theta}^t)$ for each instance m. If we let $\mathcal{H}^t$ comprise all of the assignments $c[m]$, this results in a complete data set $(\mathcal{D}^+)^t = \langle \mathcal{D}, \mathcal{H}^t \rangle$.
- Set $\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta} : (\mathcal{D}^+)^t)$. This step requires collecting sufficient statistics from $(\mathcal{D}^+)^t$, and then choosing MLE parameters based on these.

This approach is often used where the class-conditional distributions $P(\boldsymbol{X} \mid c)$ are all "round" Gaussian distributions with unit variance. Thus, each class c has its own mean vector $\boldsymbol{\mu}_c$, but a unit covariance matrix. In this case, the most likely class for an instance $\boldsymbol{x}$ is simply the

class c such that the Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{\mu}_c$ is smallest. In other words, each point gravitates to the class to which it is "closest." The reestimation step is also simple. It simply selects the mean of the class to be at the center of the cloud of points that have aligned themselves with it. This process iterates until convergence. This algorithm is called *k-means*.

k-means

Although hard-assignment EM is often used for clustering, it can be defined more broadly; we return to it in greater detail in section 19.2.2.6.

---

collaborative filtering

Box 19.A — Case Study: Discovering User Clusters. *In box 18.C, we discussed the* collaborative filtering *problem, and the use of Bayesian network structure learning to address it. A different application of Bayesian network learning to the collaborative filtering data task, proposed by Breese et al. (1998), utilized a Bayesian clustering approach. Here, one can introduce a cluster variable C denoting subpopulations of customers. In a simple model, the individual purchases X_i of each user are taken to be conditionally independent given the user's cluster assignment C. Thus, we have a naive Bayes clustering model, to which we can apply the EM algorithm. (As in box 18.C, items i that the user did not purchase are assigned $X_i = x_i^0$.)*

This learned model can be used in several ways. Most obviously, we can use inference to compute the probability that the user will purchase item i, given a set of purchases S. Empirical studies show that this approach achieves lower performance than the structure learning approach of box 18.C, probably because the "user cluster" variable simply cannot capture the complex preference patterns over a large number of items. However, this model can provide significant insight into the types of users present in a population, allowing, for example, a more informed design of advertising campaigns.

As one example, Bayesian clustering was applied to a data set of people browsing the MSNBC website. Each article was associated with a binary random variable X_i, which took the value x_i^1 if the user followed the link to the article. Figure 19.A.1 shows the four largest clusters produced by Bayesian clustering applied to this data set. Cluster 1 appears to represent readers of commerce and technology news (a large segment of the reader population at that period, when Internet news was in its early stages). Cluster 2 are people who mostly read the top-promoted stories in the main page. Cluster 3 are readers of sports news. In all three of these cases, the user population was known in advance, and the website contained a page targeting these readers, from which the articles shown in the table were all linked. The fourth cluster was more surprising. It appears to contain readers interested in "softer" news. The articles read by this population were scattered all over the website, and users often browsed several pages to find them. Thus, the clustering algorithm revealed an unexpected pattern in the data, one that may be useful for redesigning the website.

19.2.2.5 Theoretical Foundations ⋆

So far, we used an intuitive argument to derive the details of the EM algorithm. We now formally analyze this algorithm and prove the results regarding its convergence properties.

At each iteration, EM maintains the "current" set of parameters. Thus, we can view it as a local learning algorithm. Each iteration amounts to taking a step in the parameter space from

Cluster 1 (36 percent)	**Cluster 2** (29 percent)
E-mail delivery isn't exactly guaranteed	757 Crashes at sea
Should you buy a DVD player?	Israel, Palestinians agree to direct talks
Price low, demand high for Nintendo	Fuhrman pleads innocent to perjury
Cluster 3 (19 percent)	**Cluster 4** (12 percent)
Umps refusing to work is the right thing	The truth about what things cost
Cowboys are reborn in win over eagles	Fuhrman pleads innocent to perjury
Did Orioles spend money wisely?	Real astrology

Figure 19.A.1 — Application of Bayesian clustering to collaborative filtering. Four largest clusters found by Bayesian clustering applied to MSNBC news browsing data. For each cluster, the table shows the three news articles whose probability of being browsed is highest.

$\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}^{t+1}$. This is similar to gradient-based algorithms, except that in those algorithms we have good understanding of the nature of the step, since each step attempts to go uphill in the steepest direction. Can we find a similar justification for the EM iterations?

The basic outline of the analysis proceeds as follows. We will show that each iteration can be viewed as maximizing an *auxiliary function*, rather than the actual likelihood function. The choice of auxiliary function depends on the current parameters at the beginning of the iteration. The auxiliary function is nice in the sense that it is similar to the likelihood function in complete data problems. The crucial part of the analysis is to show how the auxiliary function relates to the likelihood function we are trying to maximize. As we will show, the relation is such that we can show that the parameters that maximize the auxiliary function in an iteration also have better likelihood than the parameters with which we started the iteration.

The Expected Log-Likelihood Function Assume we are given a data set $\mathcal{D}$ that consists of partial observations. Recall that $\mathcal{H}$ denotes a possible assignment to all the missing values in our data set. The combination of $\mathcal{D}, \mathcal{H}$ defines a complete data set $\mathcal{D}^+ = \langle \mathcal{D}, \mathcal{H} \rangle = \{\boldsymbol{o}[m], \boldsymbol{h}[m]\}_m$, where in each instance we now have a full assignment to all the variables. We denote by $\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$ the log-likelihood of the parameters $\boldsymbol{\theta}$ with respect to this completed data set.

Suppose we are not sure about the true value of $\mathcal{H}$. Rather, we have a probabilistic estimate that we denote by a distribution Q that assigns a probability to each possible value of $\mathcal{H}$. Note that Q is a joint distribution over full assignments to all of the missing values in the entire data set. Thus, for example, in our earlier network, if $\mathcal{D}$ contains two instances $\boldsymbol{o}[1] = \langle a^1, ?, ?, d^0 \rangle$ and $\boldsymbol{o}[2] = \langle ?, b^1, ?, d^1 \rangle$, then Q is a joint distribution over $B[1], C[1], A[2], C[2]$.

In the fully observed case, our score for a set of parameters was the log-likelihood. In this case, given Q, we can use it to define an average score, which takes into account the different possible completions of the data and their probabilities. Specifically, we define the *expected log-likelihood* as:

expected log-likelihood

$$\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{\mathcal{H}} Q(\mathcal{H}) \ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$$

This function has appealing characteristics that are important in the derivation of EM.

The first key property is a consequence of the linearity of expectation. Recall that when learning table-CPDs, we showed that

$$\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle) = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i] \log \theta_{x_i | \boldsymbol{u}_i}.$$

Because the only terms in this sum that depend on $\langle \mathcal{D}, \mathcal{H} \rangle$ are the counts $M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i]$, and these appear within a linear function, we can use linearity of expectations to show that

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} \boldsymbol{E}_Q \left[M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i] \right] \log \theta_{x_i | \boldsymbol{u}_i}.$$

If we now generalize our notation to define

$$\bar{M}_Q[x_i, \boldsymbol{u}_i] = \boldsymbol{E}_{\mathcal{H} \sim Q} \left[M_{\langle \mathcal{D}, \mathcal{H} \rangle}[x_i, \boldsymbol{u}_i] \right] \tag{19.5}$$

we obtain

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \sum_{i=1}^{n} \sum_{(x_i, \boldsymbol{u}_i) \in Val(X_i, \mathrm{Pa}_{X_i})} \bar{M}_Q[x_i, \boldsymbol{u}_i] \log \theta_{x_i | \boldsymbol{u}_i}.$$

This expression has precisely the same form as the log-likelihood function in the complete data case, but using the expected counts rather than the exact full-data counts. The implication is that instead of summing over all possible completions of the data, we can evaluate the expected log-likelihood based on the expected counts.

The crucial point here is that the log-likelihood function of complete data is *linear* in the counts. This allows us to use linearity of expectations to write the expected likelihood as a function of the expected counts.

The same idea generalizes to any model in the exponential family, which we defined in chapter 8. Recall that a model is in the exponential family if we can write:

$$P(\xi \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} A(\xi) \exp \left\{ \langle \mathtt{t}(\boldsymbol{\theta}), \tau(\xi) \rangle \right\},$$

where $\langle \cdot, \cdot \rangle$ is the inner product, $A(\xi)$, $\mathtt{t}(\boldsymbol{\theta})$, and $Z(\boldsymbol{\theta})$ are functions that define the family, and $\tau(\xi)$ is the sufficient statistics function that maps a complete instance to a vector of sufficient statistics.

As discussed in section 17.2.5, when learning parameters of such a model, we can summarize the data using the sufficient statistic function τ. We define

$$\tau(\langle \mathcal{D}, \mathcal{H} \rangle) = \sum_{m} \tau(\boldsymbol{o}[m], \boldsymbol{h}[m]).$$

Because the model is in the exponential family, we can write the log-likelihood $\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$ as a *linear function* of $\tau(\langle \mathcal{D}, \mathcal{H} \rangle)$

$$\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle) = \langle \mathtt{t}(\boldsymbol{\theta}), \tau(\langle \mathcal{D}, \mathcal{H} \rangle) \rangle + \sum_{m} \log A(\boldsymbol{o}[m], \boldsymbol{h}[m]) - M \log Z(\boldsymbol{\theta}).$$

Using the linearity of expectation, we see that

$$\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \langle \mathsf{t}(\boldsymbol{\theta}), \mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] \rangle + \sum_m \mathbf{E}_Q[\log A(\boldsymbol{o}[m], \boldsymbol{h}[m])] - M \log Z(\boldsymbol{\theta}).$$

Because $A(\boldsymbol{o}[m], \boldsymbol{h}[m])$ does not depend on the choice of $\boldsymbol{\theta}$, we can ignore it. We are left with maximizing the function:

$$\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] = \langle \mathsf{t}(\boldsymbol{\theta}), \mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)] \rangle - M \log Z(\boldsymbol{\theta}) + \text{const}. \tag{19.6}$$

In summary, the derivation here is directly analogous to the one for table-CPDs. The expected log-likelihood is a linear function of the expected sufficient statistics $\mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)]$. We can compute these as in equation (19.4), by aggregating their expectation in each instance in the training data. Now, maximizing the right-hand side of equation (19.6) is equivalent to maximum likelihood estimation in a *complete* data set where the sum of the sufficient statistics coincides with the expected sufficient statistics $\mathbf{E}_Q[\tau(\langle \mathcal{D}, \mathcal{H} \rangle)]$. These two steps are exactly the E-step and M-step we take in each iteration of the EM procedure shown in algorithm 19.2. In the procedure, the distribution Q that we are using is $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. Because instances are assumed to be independent given the parameters, it follows that

$$P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t) = \prod_m P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t),$$

where $\boldsymbol{h}[m]$ are the missing variables in the m'th data instance, and $\boldsymbol{o}[m]$ are the observations in the m'th instance. Thus, we see that in the t'th iteration of the EM procedure, we choose $\boldsymbol{\theta}^{t+1}$ to be the ones that maximize $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ with $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. This discussion allows us to understand a single iteration as an (implicit) optimization step of a well-defined target function.

Choosing Q The discussion so far has showed that we can use properties of exponential models to efficiently maximize the expected log-likelihood function. Moreover, we have seen that the t'th EM iteration can be viewed as maximizing $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ where Q is the conditional probability $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. This discussion, however, does not provide us with guidance as to why we choose this particular auxiliary distribution Q. Note that each iteration uses a different Q distribution, and thus we cannot relate the optimization taken in one iteration to the ones made in the subsequent one. We now show why the choice $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ allows us to prove that each EM iteration improves the likelihood function.

To do this, we will define a new function that will be the target of our optimization. Recall that our ultimate goal is to maximize the log-likelihood function. The log-likelihood is a function only of $\boldsymbol{\theta}$; however, in intermediate steps, we also have the current choice of Q. Therefore, we will define a new function that accounts for both $\boldsymbol{\theta}$ and Q, and view each step in the algorithm as maximizing this function.

We already encountered a similar problem in our discussion of approximate inference in chapter 11. Recall that in that setting we had a known distribution P and attempted to find an approximating distribution Q. This problem is similar to the one we face, except that in learning we also change the parameters of target distribution P to maximize the probability of the data.

Let us briefly summarize the main idea that we used in chapter 11. Suppose that $P = \tilde{P}/Z$ is some distribution, where $\tilde{P}$ is an unnormalized part of the distribution, specified by a product

of factors, and Z is the partition function that ensures that P sums up to one. We defined the energy functional *energy functional* as

$$F[P, Q] = \mathbf{E}_Q\left[\log \tilde{P}\right] + \mathbf{H}_Q(\mathcal{X}).$$

We then showed that the logarithm of the partition function can be rewritten as:

$$\log Z = F[P, Q] + \mathbf{D}(Q \| P).$$

How does this apply to the case of learning from missing data? We can choose

$$P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}) = P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta}) / P(\mathcal{D} \mid \boldsymbol{\theta})$$

as our distribution over $\mathcal{H}$ (we hold $\mathcal{D}$ and $\boldsymbol{\theta}$ fixed for now). With this choice, the partition function $Z(\boldsymbol{\theta})$ is the data likelihood $P(\mathcal{D} \mid \boldsymbol{\theta})$ and $\tilde{P}$ is the joint probability $P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})$, so that $\log \tilde{P} = \ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)$. Rewriting the energy functional for this new setting, we obtain:

$$F_{\mathcal{D}}[\boldsymbol{\theta}, Q] = \mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \mathbf{H}_Q(\mathcal{H}).$$

expected log-likelihood Note that the first term is precisely the *expected log-likelihood* relative to Q. Applying our earlier analysis, we now can prove

Corollary 19.1

For any Q,

$$\begin{aligned} \ell(\boldsymbol{\theta} : \mathcal{D}) &= F_{\mathcal{D}}[\boldsymbol{\theta}, Q] + \mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})) \\ &= \mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \mathbf{H}_Q(\mathcal{H}) + \mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})). \end{aligned}$$

Both equalities have important ramifications. Starting from the second equality, since both the entropy $\mathbf{H}_Q(\mathcal{H})$ and the relative entropy $\mathbf{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}))$ are nonnegative, we conclude that the expected log-likelihood $\mathbf{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$ is a lower bound on $\ell(\boldsymbol{\theta} : \mathcal{D})$. This result is data completion true for any choice of distribution Q. If we select $Q(\mathcal{H})$ to be the *data completion distribution* $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, the relative entropy term becomes zero. In this case, the remaining term $\mathbf{H}_Q(\mathcal{H})$ captures to a certain extent the difference between the expected log-likelihood and the real log-likelihood. Intuitively, when Q is close to being deterministic, the expected value is close to the actual value.

The first equality, for the same reasons, shows that, for any distribution Q, the F function is a lower bound on the log-likelihood. Moreover, this lower bound is tight for every choice of $\boldsymbol{\theta}$: if we choose $Q = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, the two functions have the same value. Thus, if we maximize the F function, we are bound to maximize the log-likelihood.

There many possible ways to optimize this target function. We now show that the EM procedure we described can be viewed as *implicitly* optimizing the EM functional F using a coordinate ascent particular optimization strategy. The strategy we are going to utilize is a *coordinate ascent* optimization. We start with some choice $\boldsymbol{\theta}$ of parameters. We then search for Q that maximizes $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$ while keeping $\boldsymbol{\theta}$ fixed. Next, we fix Q and search for parameters that maximize $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$. We continue in this manner until convergence.

We now consider each of these steps.

- **Optimizing** Q. Suppose that $\boldsymbol{\theta}$ are fixed, and we are searching for $\arg\max_Q F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$. Using corollary 19.1, we know that, if $Q^* = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, then

$$F_{\mathcal{D}}[\boldsymbol{\theta}, Q^*] = \ell(\boldsymbol{\theta} : \mathcal{D}) \geq F_{\mathcal{D}}[\boldsymbol{\theta}, Q].$$

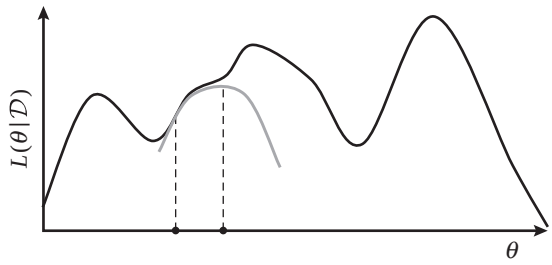


Figure 19.8 An illustration of the hill-climbing process performed by the EM algorithm. The black line represents the log-likelihood function; the point on the left represents $\boldsymbol{\theta}^t$; the gray line represents the expected log-likelihood derived from $\boldsymbol{\theta}^t$; and the point on the right represents the parameters $\boldsymbol{\theta}^{t+1}$ that maximize this expected log-likelihood.

Thus, we maximize the EM functional by choosing the auxiliary distribution Q^*. In other words, we can view the E-step as implicitly optimizing Q by using $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ in computing the expected sufficient statistics.

- **Optimizing $\boldsymbol{\theta}$.** Suppose Q is fixed, and that we wish to find $\arg\max_{\boldsymbol{\theta}} F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$. Because the only term in F that involves $\boldsymbol{\theta}$ is $\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)]$, the maximization is equivalent to maximizing the expected log-likelihood. As we saw, we can find the maximum by computing expected sufficient statistics and then solving the MLE given these expected sufficient statistics.

Convergence of EM The discussion so far shows that the EM procedure can be viewed as maximizing an objective function; because the objective function can be shown to be bounded, this procedure is guaranteed to converge. However, it is not clear what can be said about the convergence points of this procedure. We now analyze the convergence points of this procedure in terms of our true objective: the log-likelihood function. Intuitively, as our procedure is optimizing the energy functional, which is a tight lower bound of the log-likelihood function, each step of this optimization also improves the log-likelihood. This intuition is illustrated in figure 19.8. In more detail, the E-step is selecting, at the current set of parameters, the distribution Q^t for which the energy functional is a tight lower bound to $\ell(\boldsymbol{\theta} : \mathcal{D})$. The energy functional, which is a well-behaved concave function in $\boldsymbol{\theta}$, can be maximized effectively via the M-step, taking us to the parameters $\boldsymbol{\theta}^{t+1}$. Since the energy functional is guaranteed to remain below the log-likelihood function, this step is guaranteed to improve the log-likelihood. Moreover, the improvement is guaranteed to be at least as large as the improvement in the energy functional. More formally, using corollary 19.1, we can now prove the following generalization of theorem 19.3:

Theorem 19.5

During iterations of the EM procedure of algorithm 19.2, we have that

$$\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) - \ell(\boldsymbol{\theta}^t : \mathcal{D}) \geq \boldsymbol{E}_{P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)}\left[\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}, \mathcal{H})\right] - \boldsymbol{E}_{P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)}\left[\ell(\boldsymbol{\theta}^t : \mathcal{D}, \mathcal{H})\right].$$

As a consequence, we obtain that:

$$\ell(\boldsymbol{\theta}^t : \mathcal{D}) \leq \ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}).$$

PROOF We begin with the first statement. Using corollary 19.1, with the distribution $Q^t(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ we have that

$$\begin{aligned}
\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) &= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t+1} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}) + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t+1})) \\
\ell(\boldsymbol{\theta}^{t} : \mathcal{D}) &= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}) + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t})) \\
&= \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_{Q^t}(\mathcal{H}).
\end{aligned}$$

The last step is justified by our choice of $Q^t(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$. Subtracting these two terms, we have that

$$\begin{aligned}
&\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) - \ell(\boldsymbol{\theta}^{t} : \mathcal{D}) = \\
&\quad \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}, \mathcal{H})\right] - \boldsymbol{E}_{Q^t}\left[\ell(\boldsymbol{\theta}^{t} : \mathcal{D}, \mathcal{H})\right] + \boldsymbol{D}(Q^t(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^{t+1})).
\end{aligned}$$

Because the last term is nonnegative, we get the desired inequality.

To prove the second statement of the theorem, we note that $\boldsymbol{\theta}^{t+1}$ is the value of $\boldsymbol{\theta}$ that maximizes $\boldsymbol{E}_{P(\mathcal{H}|\mathcal{D},\boldsymbol{\theta}^t)}[\ell(\boldsymbol{\theta} : \mathcal{D}, \mathcal{H})]$. Hence the value obtained for this expression for $\boldsymbol{\theta}^{t+1}$ is at least as large as the value obtained for any other set of parameters, including $\boldsymbol{\theta}^t$. It follows that the right-hand side of the inequality is nonnegative, which implies the second statement. ■

☞ **We conclude that EM performs a variant of hill climbing, in the sense that it improves the log-likelihood at each step. Moreover, the M-step can be understood as maximizing a lower-bound on the improvement in the likelihood. Thus, in a sense we can view the algorithm as searching for the largest possible improvement, when using the expected log-likelihood as a proxy for the actual log-likelihood.**

For most learning problems, we know that the log-likelihood is upper bounded. For example, if we have discrete data, then the maximal likelihood we can assign to the data is 1. Thus, the log-likelihood is bounded by 0. If we have a continuous model, we can construct examples where the likelihood can grow unboundedly; however, we can often introduce constraints on the parameters that guarantee a bound on the likelihood (see exercise 19.10). If the log-likelihood is bounded, and the EM iterations are nondecreasing in the log-likelihood, then the sequence of log-likelihoods at successive iterations must converge.

The question is what can be said about this convergence point. Ideally, we would like to guarantee convergence to the maximum value of our log-likelihood function. Unfortunately, as we mentioned earlier, we cannot provide this guarantee; however, we can now prove theorem 19.4, which shows convergence to a fixed point of the log-likelihood function, that is, one where the gradient is zero. We restate the theorem for convenience:

Theorem 19.6 *Suppose that $\boldsymbol{\theta}^t$ is such that $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$ during EM, and $\boldsymbol{\theta}^t$ is also an interior point of the allowed parameter space. Then $\boldsymbol{\theta}^t$ is a stationary point of the log-likelihood function.*

PROOF We start by rewriting the log-likelihood function using corollary 19.1.

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H}\rangle)] + \boldsymbol{H}_Q(\mathcal{H}) + \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})).$$

We now consider the gradient of $\ell(\boldsymbol{\theta} : \mathcal{D})$ with respect to $\boldsymbol{\theta}$. Since the term $\boldsymbol{H}_Q(\mathcal{H})$ does not depend on $\boldsymbol{\theta}$, we get that

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta} : \mathcal{D}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \nabla_{\boldsymbol{\theta}} \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})).$$

This observation is true for any choice of Q. Now suppose we are in an EM iteration. In this case, we set $Q = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)$ and evaluate the gradient at $\boldsymbol{\theta}^t$.

A somewhat simplified proof runs as follows. Because $\boldsymbol{\theta} = \boldsymbol{\theta}^t$ is a minimum of the KL-divergence term, we know that $\nabla_{\boldsymbol{\theta}} \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t))$ is 0. This implies that

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^t : \mathcal{D}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{E}_Q \left[\ell(\boldsymbol{\theta}^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right].$$

Or, in other words, $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^t : \mathcal{D}) = 0$ if and only if $\nabla_{\boldsymbol{\theta}} \boldsymbol{E}_Q \left[\ell(\boldsymbol{\theta}^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right] = 0$.

Recall that $\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} \boldsymbol{E}_Q \left[\ell(\boldsymbol{\theta}^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right]$. Hence the gradient of the expected likelihood at $\boldsymbol{\theta}^{t+1}$ is 0. Thus, we conclude that $\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t$ only if $\nabla_{\boldsymbol{\theta}} \boldsymbol{E}_Q \left[\ell(\boldsymbol{\theta}^t : \langle \mathcal{D}, \mathcal{H} \rangle)\right] = 0$. And so, at this point, $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^t : \mathcal{D}) = 0$. This implies that this set of parameters is a stationary point of the log-likelihood function.

The actual argument has to be somewhat more careful. Recall that the parameters must lie within some allowable set. For example, the parameters of a discrete random variable must sum up to one. Thus, we are searching within a constrained space of parameters. When we have constraints, we often do not have zero gradient. Instead, we get to a stationary point when the gradient is orthogonal to the constraints (that is, local changes within the allowed space do not improve the likelihood). The arguments we have stated apply equally well when we replace statements about equality to 0 with orthogonality to the constraints on the parameter space. ■

19.2.2.6 Hard-Assignment EM

In section 19.2.2.4, we briefly mentioned the idea of using a hard assignment to the hidden variables, in the context of applying EM to Bayesian clustering. We now generalize this simple idea to the case of arbitrary Bayesian networks.

hard-assignment EM

This algorithm, called *hard-assignment EM*, also iterates over two steps: one in which it completes the data given the current parameters $\boldsymbol{\theta}^t$, and the other in which it uses the completion to estimate new parameters $\boldsymbol{\theta}^{t+1}$. However, rather than using a soft completion of the data, as in standard EM, it selects for each data instance $\boldsymbol{o}[m]$ the single assignment $\boldsymbol{h}[m]$ that maximizes $P(\boldsymbol{h} \mid \boldsymbol{o}[m], \boldsymbol{\theta}^t)$.

Although hard-assignment EM is similar in outline to EM, there are important differences. In fact, hard-assignment EM can be described as optimizing a different objective function, one that involves both the learned parameters and the learned assignment to the hidden variables. This objective is to maximize the likelihood of the complete data $\langle \mathcal{D}, \mathcal{H} \rangle$, given the parameters:

$$\max_{\boldsymbol{\theta}, \mathcal{H}} \ell(\boldsymbol{\theta} : \mathcal{H}, \mathcal{D}).$$

See exercise 19.14. Compare this objective to the EM objective, which attempts to maximize $\ell(\boldsymbol{\theta} : \mathcal{D})$, averaging over all possible completions of the data.

Does this observation provide us insight on these two learning procedures? The intuition is that these two objectives are similar if $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$ assigns most of the probability mass to

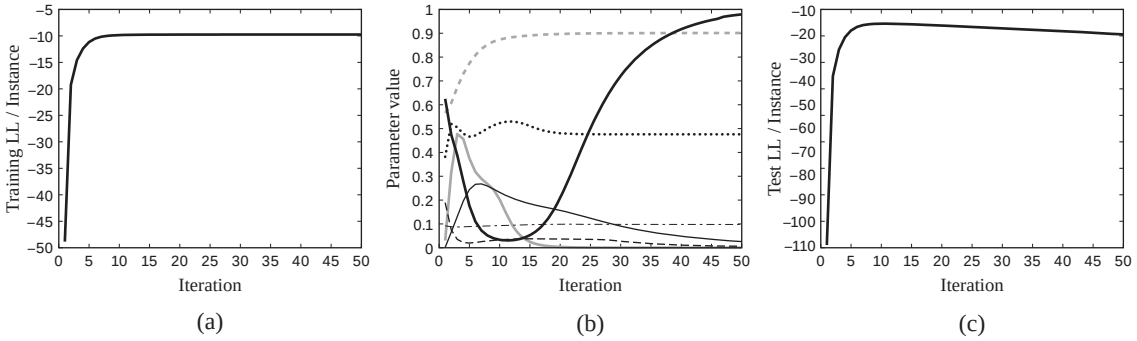


Figure 19.B.1 — Convergence of EM run on the ICU Alarm network. (a) Training log-likelihood. (b) Progress of several sample parameters. (c) Test data log-likelihood.

one completion of the data. In such a case, EM will effectively perform hard assignment during the E-step. However, if $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$ is diffuse, the two algorithms will lead to very different solutions. In clustering, the hard-assignment version tends to increase the contrast between different classes, since assignments have to choose between them. In contrast, EM can learn classes that are overlapping, by having many instances contributing to two or more classes.

Another difference between the two EM variants is in the way they progress during the learning. Note that for a given data set, at the end of an iteration, the hard-assignment EM can be in one of a finite number of parameter values. Namely, there is only one parameter assignment for each possible assignment to $\mathcal{H}$. Thus, hard-assignment EM traverses a path in the combinatorial space of assignments to $\mathcal{H}$. The soft-assignment EM, on the other hand, traverses the continuous space of parameter assignments. The intuition is that hard-assignment EM converges faster, since it makes discrete steps. In contrast, soft-assignment EM can converge very slowly to a local maximum, since close to the maximum, each iteration makes only small changes to the parameters. The flip side of this argument is that soft-assignment EM can traverse paths that are infeasible to the hard-assignment EM. For example, if two clusters need to shift their means in a coordinated fashion, soft-assignment EM can progressively change their means. On the other hand, hard-assignment EM needs to make a "jump," since it cannot simultaneously reassign multiple instances and change the class means.

Box 19.B — Case Study: EM in Practice. *The EM algorithm is guaranteed to monotonically improve the training log-likelihood at each iteration. However, there are no guarantees as to the speed of convergence or the quality of the local maxima attained. To gain a better perspective of how the algorithm behaves in practice, we consider here the application of the method to the* ICU-Alarm *network discussed in earlier learning chapters.*

We start by considering the progress of the training data likelihood during the algorithm's iterations. In this example, $1,000$ *samples were generated from the* ICU-Alarm *network. For each instance, we then independently and randomly hid 50 percent of the variables. As can be seen in figure 19.B.1a, much of the improvement over the performance of the random starting point is in the*

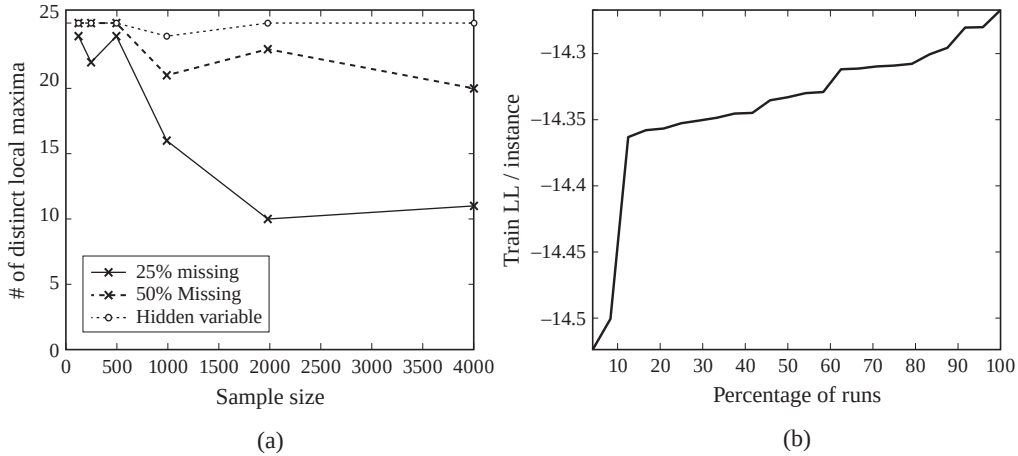


Figure 19.B.2 — Local maxima in likelihood surface. (a) Number of unique local maxima (in 25 runs) for different sample sizes and missing value configurations. (b) Distribution of training likelihood of local maxima attained for 25 random starting points with 1,000 samples and one hidden variable.

first few iterations. However, examining the convergence of different parameters in (b), we see that some parameters change significantly after the fifth iteration, even though changes to the likelihood are relatively small. In practice, any nontrivial model will display a wide range of sensitivity to the network parameters. Given more training data, the sensitivity will, typically, overall decrease. Owing to these changes in parameters, the training likelihood continues to improve after the initial iterations, but very slowly. This behavior of fast initial improvement, followed by slow convergence, is typical of EM.

We next consider the behavior of the learned model on unseen test data. As we can see in (c), early in the process, test-data improvement correlates with training-data performance. However, after the 10*th iterations, training performance continues to improve, but test performance decreases. This phenomenon is an instance of* overfitting *to the training data. With more data or fewer unobserved values, this phenomenon will be less pronounced. With less data or hidden variables, on the other hand, explicit techniques for coping with the problem may be needed (see box 19.C).*

overfitting

A second key issue any type of optimization of the likelihood in the case of missing data is that of local maxima. To study this phenomenon, we consider the number of local maxima for 25 *random starting points under different settings. As the sample size (x-axis) grows, the number of local maxima diminishes. In addition, the number of local maxima when more values are missing (dashed line) is consistently greater than the number of local maxima in the setting where more data is available (solid line). Importantly, in the case where just a single variable is hidden, the number of local maxima is large, and remains large even when the amount of training data is quite large. To see that this is not just an artifact of possible permutations of the values of the hidden variable, and to demonstrate the importance of achieving a superior local maxima, in (b) we show the training set log-likelihood of the* 25 *different local maxima attained. The difference between the*

best and worst local maxima is over 0.2 *bit-per-instance. While this may not seem significant, for a training set of* $1,000$ *instances, this corresponds to a factor of* $2^{0.2*1,000} \approx 10^{60}$ *in the training set likelihood. We also note that the spread of the quality in different local maxima is quite uniform, so that it is not easy to attain a good local maximum with a small number of random trials.*

19.2.3 Comparison: Gradient Ascent versus EM

So far we have discussed two algorithms for parameter learning with incomplete data: gradient ascent and EM. As we will discuss (see box 19.C), there are many issues involved in the actual implementation of these algorithms: the choice of initial parameters, the stopping criteria, and so forth. However, before discussing these general points, it is worth comparing the two algorithms.

There are several points of similarity in the overall strategy of both algorithms. Both algorithms are *local* in nature. At each iteration they maintain a "current" set of parameters, and use these to find the next set. Moreover, both perform some version of greedy optimization based on the current point. Gradient ascent attempts to progress in the steepest direction from the current point. EM performs a greedy step in improving its target function given the local parameters. Finally, both algorithms provide a guarantee to converge to local maxima (or, more precisely, to stationary points where the gradient is 0). On one hand, this is an important guarantee, in the sense that both are at least locally maximal. On the other hand, this is a weak guarantee, since many real-world problems have multimodal likelihood functions, and thus we do not know how far the learned parameters are from the global maximum (or maxima).

In terms of the actual computational steps, the two algorithms are also quite similar. For table-CPDs, the main component of either an EM iteration or a gradient step is computing the expected sufficient statistics of the data given the current parameters. This involves performing inference on each instance. Thus, both algorithms can exploit dynamic programming procedures (for example, clique tree inference) to compute all the expected sufficient statistics in an instance efficiently.

In term of implementation details, the algorithms provide different benefits. On one hand, gradient ascent allows to use "black box" nonlinear optimization techniques, such as conjugate gradient ascent (see appendix A.5.2). This allows the implementation to build on a rich set of existing tools. Moreover, gradient ascent can be easily applied to various CPDs by using the chain rule of derivatives. On the other hand, EM relies on maximization from complete data. Thus, it allows for a fairly straightforward use of learning procedure for complete data in the case of incomplete data. The only change is replacing the part that accumulates sufficient statistics by a procedure that computes expected sufficient statistics. As such, most people find EM easier to implement.

A final aspect for consideration is the convergence rate of the algorithm. Although we cannot predict in advance how many iterations are needed to learn parameters, analysis can show the general behavior of the algorithm in terms of how fast it approaches the convergence point.

Suppose we denote by $\ell_t = \ell(\theta^t : \mathcal{D})$ the likelihood of the solution found in the t'th iteration (of either EM or gradient ascent). The algorithm converges toward $\ell^* = \lim_{t\to\infty} \ell_t$. The error at the t'th iteration is

$$\epsilon_t = \ell^* - \ell_t.$$

convergence rate

Although we do not go through the proof, one can show that EM has *linear convergence rate*. This means that for each domain there exists a t_0 and $\alpha < 1$ such that for all $t \geq t_0$

$$\epsilon_{t+1} \leq \alpha \epsilon_t.$$

On the face of it, this is good news, since it shows that the error decreases at each iteration. Such a convergence rate means that $\ell_{t+1} - \ell_t = \epsilon_t - \epsilon_{t+1} \geq \epsilon_t(1 - \alpha)$. In other words, if we know α, we can bound the error

$$\epsilon_t \leq \frac{\ell_{t+1} - \ell_t}{1 - \alpha}.$$

While this result provides a bound on the error (and also suggests a way of estimating it), it is not always a useful one. In particular, if α is relatively close to 1, then even when the difference is likelihood between successive iterations is small, the error can be much larger. Moreover, the number of iterations to convergence can be very large. In practice we see this behavior quite often. **The first iterations of EM show huge improvement in the likelihood. These are then followed by many iterations that slowly increase the likelihood; see box 19.B. Conjugate gradient often has opposite behavior. The initial iterations (which are far away from the local maxima) often take longer to improve the likelihood. However, once the algorithm is in the vicinity of maximum, where the log-likelihood function is approximately quadratic, this method is much more efficient in zooming on the maximum.** Finally, it is important to keep in mind that these arguments are asymptotic in the number of iterations; the actual number of iterations required for convergence may not be in the asymptotic regime. Thus, the rate of convergence of different algorithms may not be the best indicator as to which of them is likely to work most efficiently in practice.

Box 19.C — Skill: Practical Considerations in Parameter Learning. *There are a few practical considerations in implementing both gradient-based methods and EM for learning parameters with missing data. We now consider a few of these. We present these points mostly in the context of the EM algorithm, but most of our points apply equally to both classes of algorithms.*

In a practical implementation of EM, there are two key issues that one needs to address. The first is the presence of local maxima. As demonstrated in box 19.B, the likelihood of even relatively simple networks can have a large number of local maxima that significantly differ in terms of their quality. There are several adaptations of these local search algorithms that aim to consistently reach beneficial local maxima. These adaptations include a judicious selection of initialization, and methods for modifying the search so as to achieve a better local maximum. The second key issue involves the convergence of the algorithm: determining convergence, and improving the rate of convergence.

Local Maxima *One of the main limitations of both the EM and the gradient ascent procedures is that they are only guaranteed to reach a stationary point, which is usually a local maximum. How do we improve the odds of finding a global — or at least a good local — maximum?*

The first place where we can try to address the issue of local maxima is in the initialization of the algorithm. EM and gradient ascent, as well as most other "local" algorithms, require a starting

point — a set of initial parameters that the algorithm proceeds to improve. Since both algorithms are deterministic, this starting point (implicitly) determines which local maximum is found.

In practice, different initializations can result in radically different convergence points, sometimes with very different likelihood values. Even when the likelihood values are similar, different convergence points may represent semantically different conclusions about the data. *This issue is particularly severe when hidden variables are involved, where we can easily obtain very different clusterings of the data. For example, when clustering text documents by similarity (for example, using a version of the model in box 17.E where the document cluster variable is hidden), we can learn one model where the clusters correspond to document topics, or another where they correspond to the style of the publication in which the document appeared (for example, newspaper, webpage, or blog). Thus, initialization should generally be considered very seriously in these situations, especially when the amount of missing data is large or hidden variables are involved.*

In general, we can initialize the algorithm either in the E-step, by picking an initial set of parameters, or in the M-step, by picking an initial assignment to the unobserved variables. In the first type of approach, the simplest choices for starting points are either a set of parameters fixed in advance or randomly chosen parameters. If we use predetermined initial parameters, we should exercise care in choosing them, since a misguided choice can lead to very poor outcomes. In particular, for some learning problems, the seemingly natural choice of uniform parameters can lead to disastrous results; see exercise 19.11. Another easy choice is applicable for parts of the network where we have only a moderate amount of missing data. Here, we can sometimes estimate parameters using only the observed data, and then use those to initialize the E-step. Of course, this approach is not always feasible, and it is inapplicable when we have a hidden variable. A different natural choice is to use the mean of our prior over parameters. On one hand, if we have good prior information, this might serve as a good starting position. Note that, although this choice does bias the learning algorithm to prefer the prior's view of the data, the learned parameters can be drastically different in the end. On the other hand, if the prior is not too informative, this choice suffers from the same drawbacks we mentioned earlier. Finally, a common choice is to use a randomized starting point, an approach that avoids any intrinsic bias. However, there is also no reason to expect that a random choice will give rise to a good solution. For this reason, often one tries multiple random starting points, and the convergence point of highest likelihood is chosen.

The second class of methods initializes the procedure at the M-step by completing the missing data. Again, there are many choices for completing the data. For example, we can use a uniform or a random imputation method to assign values to missing observations. This procedure is particularly useful when we have different patterns of missing observations in each sample. Then, the counts from the imputed data consist of actual counts combined with imputed ones. The real data thus bias the estimated parameters to be reasonable. Another alternative is to use a simplified learning procedure to learn initial assignment to missing values. This procedure can be, for example, hard-assignment EM. As we discussed, such a procedure usually converges faster and therefore can serve as a good initialization. However, hard-assignment EM also requires a starting point, or a selection among multiple random starting points.

When learning with hidden variables, such procedures can be more problematic. For example, if we consider a naive Bayes clustering model and use random imputation, the result would be that we randomly assign instances to clusters. With a sufficiently large data set, these clusters will be very similar (since they all sample from the same population). In a smaller data set the sampling noise

might distinguish the initial clusters, but nonetheless, this is not a very informed starting point. We discuss some methods for initializing a hidden variable in section 19.5.3.

Other than initialization, we can also consider modifying our search so as to reduce the risk of getting stuck at a poor local optimum. The problem of avoiding local maxima is a standard one, and we describe some of the more common solutions in appendix A.4.2. Many of these solutions are applicable in this setting as well. As we mentioned, the approach of using multiple random restarts is commonly used, often with a beam search *modification to quickly prune poor starting points. In particular, in this beam search variant, K EM runs are carried out in parallel and every few iterations only the most promising ones are retained. A variant of this approach is to generate K EM threads at each step by slightly perturbing the most beneficial $k < K$ threads from the previous iteration. While such adaptations have no formal guarantees, they are extremely useful in practice in terms trading off quality of solution and computational requirements.*

beam search

Annealing methods (appendix A.4.2) have also been used successfully in the context of the EM algorithm. In such methods, we gradually transform from an easy objective with a single local maximum to the desired EM objective, and thereby we potentially avoid many local maxima that are far away from the central basin of attraction. Such an approach can be carried out by directly smoothing the log-likelihood function and gradually reducing the level to which it is smoothed, or implicitly by gradually altering the weights of training instances.

Finally, we note that we can never determine with certainty whether the EM convergence point is truly the global maximum. In some applications this limitation is acceptable — for example, if we care only about fitting the probability distribution over the training examples (say for detecting instances from a particular subpopulation). In this case, if we manage to learn parameters that assign high probability for samples in the target population, then we might be content even if these parameters are not the best ones possible. On the other hand, if we want to use the learned parameters to reveal insight about the domain, then we might care about whether the parameters are truly the optimal ones or not. In addition, if the learning procedure does not perform well, we have to decide whether the problem stems from getting trapped in a poor local maximum, or from the fact that the model is not well suited to the distribution in our particular domain.

Stopping Criteria *Both algorithms we discussed have the property that they will reach a fixed point once they converged on a stationary point of the likelihood surface. In practice, we never really reach the stationary point, although we can get quite close to it. This raises the question of when we stop the procedure.*

The basic idea is that when solutions at successive iterations are similar to each other, additional iterations will not change the solution by much. The question is how to measure similarity of solutions. There are two main approaches. The first is to compare the parameters from successive iterations. The second is to compare the likelihood of these choices of parameters. Somewhat surprisingly, these two criteria are quite different. In some situations small changes in parameters lead to dramatic changes in likelihood, and in others large changes in parameters lead to small changes in the likelihood.

To understand how there can be a discrepancy between changes in parameters and changes in likelihood, consider the properties of the gradient as shown in theorem 19.2. Using a Taylor expansion of the likelihood, this gradient provides us with an estimate how the likelihood will change when we change the parameters. We see that if $P(x, \mathbf{u} \mid \boldsymbol{o}[m], \boldsymbol{\theta})$ is small in most data

instances, then the gradient $\frac{\partial \ell(\boldsymbol{\theta}:\mathcal{D})}{\partial P(x|\boldsymbol{u})}$ will be small. This implies that relatively large changes in $P(x \mid \boldsymbol{u})$ will not change the likelihood by much. This can happen for example if the event $\boldsymbol{u}$ is uncommon in the training data, and the value of $P(x \mid \boldsymbol{u})$ is involved in the likelihood only in a few instances. On the flip side, if the event $x, \boldsymbol{u}$ has a large posterior in all samples, then the gradient $\frac{\partial \ell(\boldsymbol{\theta}:\mathcal{D})}{\partial P(x|\boldsymbol{u})}$ will be of size proportional to M. In such a situation a small change in the parameter can result in a large change in the likelihood.

In general, since we are aiming to maximize the likelihood, large changes in the parameters that have negligible effect on the likelihood are of less interest. Moreover, measuring the magnitude of changes in parameters is highly dependent on our parameterization. For example, if we use the reparameterization of equation (19.3), the difference (say in Euclidean distance) between two sets of parameters can change dramatically. Using the likelihood for tracking convergence is thus less sensitive to these choices and more directly related to the goal of the optimization.

Even once we decide what to measure, we still need to determine when we should stop the process. Some gradient-based methods, such as conjugate gradient ascent, build an estimate of the second-order derivative of the function. Using these derivatives, they estimate the improvement we expect to have. We can then decide to stop when the expected improvement is not more than a fixed amount of log-likelihood units. We can apply similar stoping criteria to EM, where again, if the change in likelihood in the last iteration is smaller than a predetermined threshold we stop the iterations.

Importantly, although the training set log-likelihood is guaranteed to increase monotonically until convergence, there is no guarantee that the generalization performance of the model — the expected log-likelihood relative to the underlying distribution — also increases monotonically. (See section 16.3.1.) Indeed, it is often the case that, as we approach convergence, the generalization performance starts to decrease, due to *overfitting* of the parameters to the specifics of the training data.

overfitting

Thus, an alternative approach is to measure directly when additional improvement to the training set likelihood does not contribute to generalization. To do so we need to separate the available data into a training set and a validation set (see box 16.A). We run learning on the training set, but at the end of each iteration we evaluate the log-likelihood of the validation set *(which is not seen during learning). We stop the procedure when the likelihood of the validation set does not improve. (As usual, the actual performance of the model would then need to be evaluated on a separate test set.) This method allows us to judge when the procedure stops learning the interesting phenomena and begins to overfit the training data. On the flip side, such a procedure is both slower (since we need to evaluate likelihood on an additional data set at the end of iteration) and forces us to train on a smaller subset of data, increasing the risk of overfitting. Moreover, if the validation set is small, then the estimate of the generalization ability by the likelihood on this set is noisy. This noise can influence the stopping time.*

validation set

Finally, we note that in EM much of the improvement is typically observed in the first few iterations, but the final convergence can be quite slow. Thus, in practice, it is often useful to limit the number of EM iterations or use a lenient convergence threshold. This is particularly important when EM is used as part of a higher-level algorithm (for example, structure learning) and where, in the intermediate stages of the overall learning algorithm, approximate parameter estimates are often sufficient. Moreover, early stopping can help reduce overfitting, as we discussed.

Accelerating Convergence *There are also several strategies that can help improve the rate of convergence of EM to its local optimum. We briefly list a few of them.*

The first idea is to use hybrid algorithms *that mix EM and gradient methods. The basic intuition is that EM is good at rapidly moving to the general neighborhood of a local maximum in few iterations but bad at pinpointing the actual maximum. Advanced gradient methods, on the other hand, quickly converge once we are close to a maximum. This observation suggests that we should run EM for few iterations and then switch over to using a method such as conjugate gradient. Such hybrid algorithms are often much more efficient. Another alternative is to use* accelerated EM *methods that take even larger steps in the search than standard EM (see section 19.7).*

accelerated EM

incremental EM

Another class of variations comprises incremental methods. *In these methods we do not perform a full E-step or a full M-step. Again, the high-level intuition is that, since we view our procedure as maximizing the energy functional $F_{\mathcal{D}}[\boldsymbol{\theta}, Q]$, we can consider steps that increase this functional but do not necessarily find the maximum value parameters or Q. For example, recall that $\boldsymbol{\theta}$ consists of several subcomponents, one per CPD. Rather than maximizing all the parameters at once, we can consider a partial update where we maximize the energy functional with respect to one of the subcomponents while freezing the others; see exercise 19.16. Another type of partial update is based on writing Q as a product of independent distributions — each one over the missing values in a particular instance. Again, we can optimize the energy functional with respect to one of these while freezing the other; see exercise 19.17. These partial updates can provide two types of benefit: they can require less computation than a full EM update, and they can propagate changes between the statistics and the parameters much faster, reducing the total number of iterations.*

Box 19.D — Case Study: EM for Robot Mapping. *One interesting application of the EM algorithm is to robotic mapping. Many variants of this applications have been proposed; we focus on one by Thrun et al. (2004) that explicitly tries to use the probabilistic model to capture the structure in the environment.*

The data in this application are a point cloud representation of an indoor environment. The point cloud can be obtained by collecting a sequence of point clouds, measured along a robot's motion trajectory. One can use a robot localization procedure to (approximately) assess the robot's pose (position and heading) along each point in the trajectory, which allows the different measurements to be put on a common frame of reference. Although the localization process is not fully accurate, the estimates are usually reasonable for short trajectories. One can then take the points obtained over the trajectory, and fit the points using polygons, to derive a 3D map of the surfaces in the robot's environment. However, the noise in the laser measurements, combined with the errors in localization, leads adjacent polygons to have slightly different surface normals, giving rise to a very jagged representation of the environment.

The EM algorithm can be used to fit a more compact representation of the environment to the data, reducing the noise and providing a smoother, more realistic output. In particular, in this example, the model consists of a set of 3D planes $p_1, \ldots, p_K$, each characterized by two parameters $\boldsymbol{\alpha}_k, \beta_k$, where $\boldsymbol{\alpha}_k$ is a unit-length vector in $\mathbb{R}^3$ that encodes the plane's surface normal vector, and β_k is a scalar that denotes its distance to the origin of the global coordinate system. Thus, the distance of any point $\boldsymbol{x}$ to the plane is $d(\boldsymbol{x}, p_k) = |\alpha_k \boldsymbol{x} - \beta_k|$.

(a) (b) (c) (d)

Figure 19.D.1 — Sample results from EM-based 3D plane mapping (a) Raw data map obtained from range finder. (b) Planes extracted from map using EM. (c) Fragment of reconstructed surface using raw data. (d) The same fragment reconstructed using planes.

correspondence variable

data association

The probabilistic model also needs to specify, for each point $\boldsymbol{x}_m$ in the point cloud, to which plane $\boldsymbol{x}_m$ belongs. This assignment can be modeled via a set of correspondence variables C_m *such that $C_m = k$ if the measurement point $\boldsymbol{x}_m$ was generated by the kth plane. Each assignment to the correspondence variables, which are unobserved, encodes a possible solution to the* data association *problem. (See box 12.D for more details.) We define $P(\boldsymbol{X}_m \mid C_m = k : \boldsymbol{\theta}_k)$ to be $\propto \mathcal{N}\left(d(\boldsymbol{x}, p_k) \mid 0; \sigma^2\right)$. In addition, we also allow an additional value $C_m = 0$ that encodes points that are not generated by any of the planes; the distribution $P(\boldsymbol{X}_m \mid C_m = 0)$ is taken to be uniform over the (finite) space.*

Given a probabilistic model, the EM algorithm can be applied to find the assignment of points to planes — the correspondence variables, which are taken to be hidden; and the parameters $\boldsymbol{\alpha}_k, \beta_k$ that characterize the planes. Intuitively, the E-step computes the assignment to the correspondence variables by assigning the weight of each point proportionately to its distance to each of them. The M-step then recomputes the parameters of each plane to fit the points assigned to it. See exercise 19.18 and exercise 19.19. The algorithm also contains an additional outer loop that heuristically suggests new surfaces to be added to the model, and removes surfaces that do not have enough support in the data (for example, one possible criterion can depend on the total weight that different data points assign to the surface).

The results of this algorithm are shown in figure 19.D.1. One can see that the resulting map is considerably smoother and more realistic than the results derived directly from the raw data.

19.2.4 Approximate Inference ⋆

The main computational cost in both gradient ascent and EM is in the computation of expected sufficient statistics. This step requires running probabilistic inference on each instance in the training data. These inference steps are needed both for computing the likelihood and for computing the posterior probability over events of the form x, pa_X for each variable and its parents. For some models, such as the naive Bayes clustering model, this inference step is almost trivial. For other models, this inference step can be extremely costly. In practice, we

often want to learn parameters for models where exact inference is impractical. Formally, this happens when the tree-width of the unobserved parts of the model is large. (Note the contrast to learning from complete data, where the cost of learning the model did not depend on the complexity of inference.) In such situations the cost of inference becomes the limiting factor in our ability to learn from data.

Example 19.9

Recall the network discussed in example 6.11 and example 19.9, where we have n students taking m classes, and the grade for each student in each class depends on both the difficulty of the class and his or her intelligence. In the ground network for this example, we have a set of variables $\boldsymbol{I} = \{I(s)\}$ for the n students (denoting the intelligence level of each student s), $\boldsymbol{D} = \{D(c)\}$ for the m courses (denoting the difficulty level of each course c), and $\boldsymbol{G} = \{G(s,c)\}$ for the grades, where each variable $G(s,c)$ has as parents $I(s)$ and $D(c)$. Since this network is derived from a plate model, the CPDs are all shared, so we only have three CPDs that must be learned: $P(I(S))$, $P(D(C))$, $P(G(S,C) \mid I(S), D(C))$.

Suppose we only observe the grades of the students but not their intelligence or the difficulty of courses, and we want to learn this model. First, we note that there is no way to force the model to respect our desired semantics for the (hidden) variables I and D; for example, a model in which we flip the two values for I is equally good. Nevertheless, we can hope that some value for I will correspond to "high intelligence" and the other to "low intelligence," and similarly for D.

To perform EM in this model, we need to infer the expected counts of assignments to triplets of variables of the form $I(s), D(c), G(s,c)$. Since we have parameter sharing, we will aggregate these counts and then estimate the CPD $P(G(S,C) \mid I(S), D(C))$ from the aggregate counts. The problem is that observing a variable $G(s,c)$ couples its two parents. Thus, this network induces a Markov network that has a pairwise potential between any pair of $I(s)$ and $D(c)$ variables that share an observed child. If enough grade variables are observed, this network will be close to a full bipartite graph, and exact inference about the posterior probability becomes intractable. This creates a serious problem in applying either EM or gradient ascent for learning this seemingly simple model from data. ■

An obvious solution to this problem is to use approximate inference procedures. A simple approach is to view inference as a "black box." Rather than invoking exact inference in the learning procedures shown in algorithm 19.1 and algorithm 19.2, we can simply invoke one of the approximate inference procedures we discussed in earlier chapters. This view is elegant because it decouples the choices made in the design of the learning procedure from the choices made in the approximate inference procedures.

However, this decoupling can obscure important effects of the approximation on our learning procedure. For example, suppose we use approximate inference for computing the gradient in a gradient ascent approach. In this case, our estimate of the gradient is generally somewhat wrong, and the errors in successive iterations are generally not consistent with each other. Such inaccuracies can confuse the gradient ascent procedure, a problem that is particularly significant when the procedure is closer to the convergence point and the gradient is close to 0, so that the errors can easily dominate. **A key question is whether learning with approximate inference results in an approximate learning procedure; that is, whether we are guaranteed to find a local maximum of an approximation of the likelihood function. In general, there are very few cases where we can provide any types of guarantees on the interaction between approximate inference and learning. Nevertheless, in practice, the use of approximate**

inference is often unavoidable, and so many applications use some form of approximate inference despite the lack of theoretical guarantees.

structured variational

One class of approximation algorithms for which a unifying perspective is useful is in the combination of EM with the global approximate inference methods of chapter 11. Let us consider first the *structured variational* methods of section 11.5, where the integration is easiest to understand. In these methods, we are attempting to find an approximate distribution Q that is close to an unnormalized distribution $\tilde{P}$ in which we are interested. We saw that algorithms in this class can be viewed as finding a distribution Q in a suitable family of distributions that maximizes the energy functional:

$$F[\tilde{P}, Q] = \boldsymbol{E}_Q\left[\log \tilde{P}\right] + \boldsymbol{H}_Q(\mathcal{X}).$$

Thus, in these approximate inference procedures, we search for a distribution Q that maximizes

$$\max_{Q \in \mathcal{Q}} F[\tilde{P}, Q].$$

We saw that we can view EM as an attempt to maximize the same energy functional, with the difference that we are also optimizing over the parameterization $\boldsymbol{\theta}$ of $\tilde{P}$. We can combine both goals into a single objective by requiring that the distribution Q used in the EM functional come from a particular family $\mathcal{Q}$. Thus, we obtain the following *variational EM* problem:

variational EM

$$\max_{\boldsymbol{\theta}} \max_{Q \in \mathcal{Q}} F_{\mathcal{D}}[\boldsymbol{\theta}, Q], \tag{19.7}$$

where $\mathcal{Q}$ is a family of approximate distributions we are considering for representing the distribution over the unobserved variables.

To apply the variational EM framework, we need to choose the family of distributions $\mathcal{Q}$ that will be used to approximate the distribution $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$. Importantly, because this posterior distribution is a product of the posteriors for the different training instance, our approximation Q can take the same form without incurring any error. Thus, we need only to decide how to represent the posterior $P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$ for each instance m. We therefore define a class $\mathcal{Q}$ that we will use to approximate $P(\boldsymbol{h}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})$. Importantly, since the evidence $\boldsymbol{o}[m]$ is different for each data instance m, the posterior distribution for each instance is also different, and hence we need to use a different distribution $Q[m] \in \mathcal{Q}$ to approximate the posterior for each data instance. In principle, using the techniques of section 11.5, we can use any class $\mathcal{Q}$ that allows tractable inference. In practice, a common solution is to use the mean field approximation, where we assume that Q is a product of marginal distributions (one per each unobserved value).

Example 19.10

Consider using the mean field approximation (see section 11.5.1) for the learning problem of example 19.9. Recall that in the mean field approximation we approximate the target posterior distribution by a product of marginals. More precisely, we approximate $P(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m) \mid \boldsymbol{o}, \boldsymbol{\theta})$ *by a distribution*

$$Q(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m)) = Q(I(s_1)) \cdots Q(I(s_n)) Q(D(c_1)) \cdots Q(D(c_m)).$$

Importantly, although the prior over the variables $I(s_1), \ldots, I(s_n)$ *is identical, their posterior is generally different. Thus, the marginal of each of the variable has different parameters in* Q *(and similarly for the* $D(c)$ *variables).*

In our approximate E-step, given a set of parameters $\boldsymbol{\theta}$ for the model, we need to compute approximate expected sufficient statistics. We do so in two steps. First, we use iterations of the mean field update equation equation (11.54) to find the best choice of marginals in Q to approximate $P(I(s_1), \ldots, I(s_n), D(c_1), \ldots, D(c_m) \mid \boldsymbol{o}, \boldsymbol{\theta})$. We then use the distribution Q to compute approximate expected sufficient statistics by finding:

$$\begin{aligned} \bar{M}_Q[g_{(i,j)}, I(s_i), D(c_j)] &= Q(I(s_i), D(c_j))\mathbf{1}\{G(s_i, c_j) = g_{(i,j)}\} \\ &= Q(I(s_i))Q(D(c_j))\mathbf{1}\{G(s_i, c_j) = g_{(i,j)}\}. \end{aligned}$$ ■

Given our choice of $\mathcal{Q}$, we can optimize the variational EM objective very similarly to the optimization of the exact EM objective, by iterating over two steps:

variational E-step

- **Variational E-step** For each m, find

$$Q^t[m] = \arg\max_{Q \in \mathcal{Q}} F_{\boldsymbol{o}[m]}[\boldsymbol{\theta}, Q].$$

  This step is identical to our definition of variational inference in chapter 11, and it can be implemented using the algorithms we discussed there, usually involving iterations of local updates until convergence.

  At the end of this step, we have an approximate distribution $Q^t = \prod_m Q^t[m]$ and can collect the expected sufficient statistics. To compute the expected sufficient statistics, we combine the observed values in the data with expected counts from the distribution Q. This process requires answering queries about events in the distribution Q. For some approximations, such as the mean field approximation, we can answer such queries efficiently (that is, by multiplying the marginal probabilities over each variables); see example 19.10. If we use a richer class of approximate distributions, we must perform a more elaborate inference process. Note that, because the approximation Q is simpler than the original distribution P, we have no guarantee that a clique tree for Q will respect the family-preservation property relative to families in P. Thus, in some cases, we may need to perform queries that are outside the clique tree used to perform the E-step (see section 10.3.3.2).

- **M-step** We find a new set of parameters

$$\boldsymbol{\theta}^{t+1} = \arg\max_{\theta} F_{\mathcal{D}}[\boldsymbol{\theta}, Q^t];$$

  this step is identical to the M-step in standard EM.

The preceding algorithm is essentially performing coordinate-wise ascent alternating between optimization of Q and $\boldsymbol{\theta}$. It opens up the way to alternative ways of maximizing the same objective function. For example, we can limit the number of iterations in the variational E-step. Since each such iteration improves the energy functional, we do not need to reach a maximum in the Q dimension before making an improvement to the parameters.

Importantly, regardless of the method used to optimize the variational EM functional of equation (19.7), we can provide some guarantee regarding the properties of our optimum. Recall that we showed that

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \max_{Q} F_{\mathcal{D}}[\boldsymbol{\theta}, Q] \geq \max_{Q \in \mathcal{Q}} F_{\mathcal{D}}[\boldsymbol{\theta}, Q].$$

lower bound

Thus, maximizing the objective of equation (19.7) maximizes a *lower bound* of the likelihood. When we limit the choice of Q to be in a particular family, we cannot necessarily get a tight bound on the likelihood. However, since we are maximizing a lower bound, we know that we do not overestimate the likelihood of parameters we are considering. If the lower bound is relatively good, this property implies that we distinguish high-likelihood regions in the parameter space from very low ones. Of course, if the lower bound is loose, this guarantee is not very meaningful.

We can try to extend these ideas to other approximation methods. For example, generalized belief propagation section 11.3 is an attractive algorithm in this context, since it can be fairly efficient. Moreover, because the cluster graph satisfies the family preservation property, computation of an expected sufficient statistic can be done locally within a single cluster in the graph. The question is whether such an approximation can be understood as maximizing a clear objective. Recall that cluster-graph belief propagation can be viewed as attempting to maximize an approximation of the energy functional where we replace the term $\boldsymbol{I}_Q(\mathcal{X})$ by approximate entropy terms. Using exactly the same arguments as before, we can then show that, if we use generalized belief propagation for computing expected sufficient statistics in the E-step, then we are effectively attempting to maximize the approximate version of the energy functional. In this case, we cannot prove that this approximation is a lower bound to the correct likelihood. Moreover, if we use a standard message passing algorithm to compute the fixed points of the energy functional, we have no guarantees of convergence, and we may get oscillations both within an E-step and over several steps, which can cause significant problems in practice. Of course, we can use other approximations of the energy functional, including ones that are guaranteed to be lower bounds of the likelihood, and algorithms that are guaranteed to be convergent. These approaches, although less commonly used at the moment, share the same benefits of the structured variational approximation.

More broadly, the ability to characterize the approximate algorithm as attempting to optimize a clear objective function is important. For example, an immediate consequence is that, to monitor the progress of the algorithm, we should evaluate the approximate energy functional, since we know that, at least when all goes well, this quantity should increase until the convergence point.

19.3 Bayesian Learning with Incomplete Data ⋆

19.3.1 Overview

In our discussion of parameter learning from complete data, we discussed the limitations of maximum likelihood estimation, many of which can be addressed by the Bayesian approach. In the Bayesian approach, we view the parameters as unobserved variables that influence the probability of all training instances. Learning then amounts to computing the probability of new examples based on the observation, which can be performed by computing the posterior probability over the parameters, and using it for prediction.

More precisely, in Bayesian reasoning, we introduce a prior $P(\boldsymbol{\theta})$ over the parameters, and are interested in computing the posterior $P(\boldsymbol{\theta} \mid \mathcal{D})$ given the data. In the case of complete data, we saw that if the prior has some properties (for example, the priors over the parameters of different CPDs are independent, and the prior is conjugate), then the posterior has a nice form

and is representable in a compact manner. Because the posterior is a product of the prior and the likelihood, it follows from our discussion in section 19.1.3 that these useful properties are lost in the case of incomplete data. In particular, as we can see from figure 19.4 and figure 19.5, the parameter variables are generally correlated in the posterior. Thus, we can no longer represent the posterior as a product of posteriors over each set of parameters. Moreover, the posterior will generally be highly complex and even multimodal. Bayesian inference over this posterior would generally require a complex integration procedure, which generally has no analytic solution.

MAP estimation

One approach, once we realize that incomplete data makes the prospects of exact Bayesian reasoning unlikely, is to focus on the more modest goal of *MAP estimation*, which we first discussed in section 17.4.4. In this approach, rather than integrating over the entire posterior $P(\mathcal{D}, \boldsymbol{\theta})$, we search for a maximum of this distribution:

$$\tilde{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \frac{P(\boldsymbol{\theta})P(\mathcal{D} \mid \boldsymbol{\theta})}{P(\mathcal{D})}.$$

Ideally, the neighborhood of the MAP parameters is the center of mass of the posterior, and therefore, using them might be a reasonable approximation for averaging over parameters in their neighborhood. Using the same transformations as in equation (17.14), the problem reduces to one of computing the optimum:

$$\text{score}_{\text{MAP}}(\boldsymbol{\theta} \ : \ \mathcal{D}) = \ell(\boldsymbol{\theta} : \mathcal{D}) + \log P(\boldsymbol{\theta}).$$

MAP-EM

This function is simply the log-likelihood function with an additional prior term. Because this prior term is usually well behaved, we can generally easily extend both gradient-based methods and the EM algorithm to this case; see, for example, exercise 19.20 and exercise 19.21. Thus, finding MAP parameters is essentially as hard or as easy as finding MLE parameters. As such, it is often applicable in practice. Of course, the same caveats that we discussed in section 17.4.4 — the sensitivity to parameterization, and the insensitivity to the form of the posterior — also apply here.

A second approach is to try to address the task of full Bayesian learning using an approximate method. Recall from section 17.3 that we can cast Bayesian learning as inference in the meta-network that includes all the variables in all the instances as well as the parameters. Computing the probability of future events amounts to performing queries about the posterior probability of the $(M + 1)$st instance given the observations about the first M instances. In the case of complete data, we could derive closed-form solutions to this inference problem. In the case of incomplete data, these solutions do not exist, and so we need to resort to approximate inference procedures.

In theory, we can apply any approximate inference procedure for Bayesian networks to this problem. Thus, all the procedures we discussed in the inference chapter can potentially be used for performing Bayesian inference with incomplete data. Of course, some are more suitable than others.

For example, we can conceivably perform likelihood weighting, as described in section 12.2: we first sample parameters from the prior, and then the unobserved variables. Each such sample will be weighted by the probability of the observed data given the sampled parameter and hidden variables. Such a procedure is relatively easy to implement and does not require running complex inference procedure. However, since the parameter space is a high-dimensional continuous region, the chance of sampling high-posterior parameters is exceedingly small. As a

result, virtually all the samples will be assigned negligible weight. Unless the learning problem is relatively easy and the prior is quite informative, this inference procedure would provide a poor approximation and would require a huge number of samples.

In the next two sections, we consider two approximate inference procedures that can be applied to this problem with some degree of success.

19.3.2 MCMC Sampling

A common strategy for dealing with hard Bayesian learning problems is to perform MCMC simulation (see section 12.3). Recall that, in these methods, we construct a Markov chain whose state is the assignment to all unobserved variables, such that the stationary distribution of the chain is posterior probability over these variables. In our case, the state of the chain consists of $\boldsymbol{\theta}$ and $\mathcal{H}$, and we need to ensure that the stationary distribution of the chain is the desired posterior distribution.

19.3.2.1 Gibbs Sampling

Gibbs sampling

One of the simplest MCMC strategies for complex multivariable chains is *Gibbs sampling*. In Gibbs sampling, we choose one of the variables and sample its value given the value of all the other variables. In our setting, there are two types of variables: those in $\mathcal{H}$ and those in $\boldsymbol{\theta}$; we deal with each separately.

Suppose $X[m]$ is one of the variables in $\mathcal{H}$. The current state of the MCMC sampler has a value for all other variables in $\mathcal{H}$ and for $\boldsymbol{\theta}$. Since the parameters are known, selecting a value for $X[m]$ requires a sampling step that is essentially the same as the one we did when we performed Gibbs sampling for inference in the m'th instance. This step can be performed using the same sampling procedure as in section 12.3.

meta-network

Now suppose that $\boldsymbol{\theta}_{X|\boldsymbol{U}}$ are the parameters for a particular CPD. Again, the current state of the sampler assigns value for all of the variables in $\mathcal{H}$. Since the structure of the *meta-network* is such that $\boldsymbol{\theta}_{X|\boldsymbol{U}}$ are independent of the parameters of all other CPDs given $\mathcal{D}$ and $\mathcal{H}$, then we need to sample from $P(\boldsymbol{\theta}_{X|\boldsymbol{U}} \mid \mathcal{D}, \mathcal{H})$ — the posterior distribution over the parameters given the complete data $\mathcal{D}, \mathcal{H}$. In section 17.4 we showed that, if the prior is of a particular form (for example, a product of Dirichlet priors), then the posterior based on complete data also has a compact form. Now, we can use these properties to sample from the posterior. To be concrete, if we consider table-CPDs with Dirichlet priors, then the posterior is a product of Dirichlet distributions, one for each assignment of values for $\boldsymbol{U}$. Thus, if we know how to sample from a Dirichlet distribution, then we can sample from this posterior. It turns out that sampling from a Dirichlet distribution can be done using a reasonably efficient procedure; see box 19.E.

Thus, we can apply Gibbs sampling to the meta-network. If we simulate a sufficiently long run, the samples we generate will be from the joint posterior probability of the parameters and the hidden variables. We can then use these samples to make predictions about new samples and to estimate the marginal posterior of parameters or hidden variables. The BUGS system (see box 12.C) provides a simple, general-purpose tool for Gibbs-sampling-based Bayesian learning.

Box 19.E — Skill: Sampling from a Dirichlet distribution. *Suppose we have a parameter vector random variable $\boldsymbol{\theta} = \langle \theta_1, \ldots, \theta_k \rangle$ that is distributed according to a Dirichlet distribution $\boldsymbol{\theta} \sim Dirichlet(\alpha_1, \ldots, \alpha_K)$. How do we sample a parameter vector from this distribution?*

An effective procedure for sampling such a vector relies on an alternative definition of the Dirichlet distribution. We need to start with some definitions.

Definition 19.6 Gamma distribution

A continuous random variable X has a Gamma distribution $\mathrm{Gamma}(\alpha, \beta)$ *if it has the density*

$$p(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}.$$

■

We can see that the $x^{\alpha-1}$ term is reminiscent of components of the Dirichlet distribution. Thus, it might not be too surprising that there is a connection between the two distributions.

Theorem 19.7

Let $X_1, \ldots, X_k$ be independent continuous random variables such that $X_i \sim \mathrm{Gamma}(\alpha_i, 1)$. Define the random vector

$$\boldsymbol{\theta} = \left\langle \frac{X_1}{X_1 + \cdots + X_k}, \ldots, \frac{X_k}{X_1 + \cdots + X_k} \right\rangle.$$

Then, $\boldsymbol{\theta} \sim Dirichlet(\alpha_1, \ldots, \alpha_k)$.

Thus, we can think of a Dirichlet distribution as a two-step process. First, we sample k independent values, each from a separate Gamma distribution. Then we normalize these values to get a distribution. The normalization creates the dependency between the components of the vector $\boldsymbol{\theta}$.

This theorem suggests a natural way to sample from a Dirichlet distribution. If we can sample from Gamma distributions, we can sample values $X_1, \ldots, X_k$ from the appropriate Gamma distribution and then normalize these values.

The only remaining question is how to sample from a Gamma distribution. We start with a special case. If we consider a variable $X \sim \mathrm{Gamma}(1, 1)$, then the density function is

$$p(X = x) = e^{-x}.$$

In this case, we can solve the cumulative distribution using simple integration and get that

$$P(X < x) = 1 - e^{-x}.$$

From this, it is not hard to show the following result:

Lemma 19.2

If $U \sim \mathrm{Unif}([0:1])$, then $-\ln U \sim \mathrm{Gamma}(1, 1)$.

In particular, if we want to sample parameter vectors from $Dirichlet(1, \ldots, 1)$, which is the uniform distribution over parameter vectors, we need to sample k values from the uniform distribution, take their negative logarithm, and normalize. Since a sample from a uniform distribution can be readily obtained from a pseudo-random number generator, we get a simple procedure for sampling from the uniform distribution over multinomial distributions. Note that this procedure is not the one we intuitively would consider for this problem. When $\alpha \neq 1$ the sampling problem is harder, and requires more sophisticated methods, often based on the rejection sampling *approach described in section 14.5.1; these methods are outside the scope of this book.*

19.3.2.2 Collapsed MCMC

Recall that, in many situations, we can make MCMC sampling more efficient by using collapsed particles that represent a partial state of the system. If we can perform exact inference over the remaining state, then we can use MCMC sampling over the smaller state space and thereby get more efficient sampling.

We can apply this idea in the context of Bayesian inference in two different ways. In one approach, we have *parameter collapsed particles*, where each particle is an assignment to the parameters $\boldsymbol{\theta}$, associated with a distribution over $\mathcal{H}$; in the other, we have *data completion distribution particles*, where each particle is an assignment to the unobserved variables $\mathcal{H}$, associated with a distribution over $\boldsymbol{\theta}$. We now discuss each of these approaches in turn.

Parameter Collapsed Particles Suppose we choose the collapsed particles to contain assignments to the parameters $\boldsymbol{\theta}$, accompanied by distributions over the hidden variables. Thus, we need to be able to deal with queries about $P(\mathcal{H} \mid \boldsymbol{\theta}, \mathcal{D})$ and $P(\mathcal{D}, \boldsymbol{\theta})$. First note that given $\boldsymbol{\theta}$, the different training instances are conditionally independent. Thus, we can perform inference in each instance separately. The question now is whether we can perform this instance-level inference efficiently. This depends on the structure of the network we are learning.

Consider, for example, the task of learning the naive Bayes clustering model of section 19.2.2.4. In this case, each instance has a single hidden variable, denoting the cluster of the instance. Given the value of the parameters, inference over the hidden variable involves summing over all the values of the hidden variable, and computing the probability of the observation variables given each value. These operations are linear in the size of the network, and thus can be done efficiently. This means that evaluating the likelihood of a proposed particle is quite fast. On the other hand, if we are learning parameters for the network of example 19.9, then the network structure is such that we cannot perform efficient exact inference. In this case the cost of evaluating the likelihood of a proposed particle is nontrivial and requires additional approximations. Thus, the ease of operations with this type of collapsed particle depends on the network structure.

In addition to evaluating the previous queries, we need to be able to perform the sampling steps. In particular, for Gibbs sampling, we need to be able to sample from the distribution:

$$P(\boldsymbol{\theta}_{X_i|\mathrm{Pa}_{X_i}} \mid \{\boldsymbol{\theta}_{X_j|\mathrm{Pa}_{X_j}}\}_{j \neq i}, \mathcal{D}).$$

Unfortunately, sampling from this conditional distribution is unwieldy. Even though we assume the value of all other parameters, since we do not have complete data, we are not guaranteed to have a simple form for this conditional distribution (see example 19.11). As an alternative to Gibbs sampling, we can use Metropolis-Hastings. Here, in each proposal step, we suggest new parameter values and evaluate the likelihood of these new parameters relative to the old ones. This step requires that we evaluate $P(\mathcal{D}, \boldsymbol{\theta})$, which is as costly as computing the likelihood. This fact makes it critical that we construct a good proposal distribution, since a poor one can lead to many (expensive) rejected proposals.

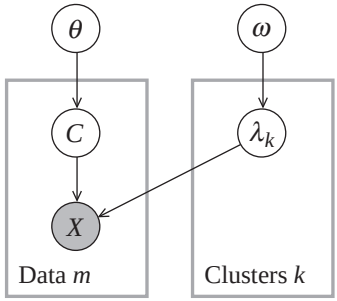


Figure 19.9 Plate model for Bayesian clustering

Example 19.11

Bayesian clustering

Let us return to the setting of Bayesian clustering *described in section 19.2.2.4. In this model, which is illustrated in figure 19.9, we have a set of data points* $\mathcal{D} = \{\boldsymbol{x}[1], \ldots, \boldsymbol{x}[M]\}$*, which are taken from one of a set of* K *clusters. We assume that each cluster is characterized by a distribution* $Q(\boldsymbol{X} \mid \boldsymbol{\lambda})$*, which has the same form for each cluster, but different parameters. As we discussed, the form of the class-conditional distribution depends on the data; typical models include naive Bayes for discrete data, or Gaussian distributions for continuous data. This decision is orthogonal to our discussion here. Thus, we have a set of* K *parameter vectors* $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K$*, each sampled from a distribution* $P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega})$*. We use a hidden variable* $C[m]$ *to represent the cluster from which the* m*'th data point was sampled. Thus, the class-conditional distribution* $P(\boldsymbol{X} \mid C = c^k, \boldsymbol{\lambda}_{1,\ldots,K}) = Q(\boldsymbol{X} \mid \boldsymbol{\lambda}_k)$*. We assume that the cluster variable* C *is sampled from a multinomial with parameters* $\boldsymbol{\theta}$*, sampled from a Dirichlet distribution* $\boldsymbol{\theta} \sim Dirichlet(\alpha_0/K, \ldots, \alpha_0/K)$*. The symmetry of the model relative to clusters reflects the fact that cluster identifiers are meaningless placeholders; it is only the partition of instances to clusters that is significant.*

To consider the use of parameter collapsed particles, let us begin by writing down the data likelihood given the parameters.

$$P(\mathcal{D} \mid \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K, \boldsymbol{\theta}) = \prod_{m=1}^{M} \left(\sum_{k=1}^{K} P(C[m] = c^k \mid \boldsymbol{\theta}) P(\boldsymbol{x}[m] \mid C[m] = c^k, \boldsymbol{\lambda}_k) \right). \quad (19.8)$$

In this case, because of the simple structure of the graphical model, this expression is easy to evaluate for any fixed set of parameters.

Now, let us consider the task of sampling $\boldsymbol{\lambda}_k$ *given* $\boldsymbol{\theta}$ *and* $\boldsymbol{\lambda}_{-k}$*, where we use a subscript of* $-k$ *to denote the set consisting of all of the values* $k' \in \{1, \ldots, K\} - \{k\}$*. The distribution with which we wish to sample* $\boldsymbol{\lambda}_k$ *is*

$$P(\boldsymbol{\lambda}_k \mid \boldsymbol{\lambda}_{-k}, \mathcal{D}, \boldsymbol{\theta}, \boldsymbol{\omega}) \propto P(\mathcal{D} \mid \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_K, \boldsymbol{\theta}) P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega}).$$

Examining equation (19.8), we see that, given the data and the parameters other than $\boldsymbol{\lambda}_k$*, all of the terms* $P(\boldsymbol{x}[m] \mid C[m] = c^{k'}, \boldsymbol{\lambda}_{k'})$ *for* $k' \neq k$ *can now be treated as a constant, and aggregated into a single number; similarly, the terms* $P(C[m] = c^k \mid \boldsymbol{\theta})$ *are also constant. Hence, each of the terms inside the outermost product can be written as a linear function* $a_m P(\boldsymbol{x}[m] \mid C[m] =$

$c^k) + b_m$. *Unfortunately, the entire expression is a product of these linear functions, making the sampling distribution for* $\boldsymbol{\lambda}_k$ *proportional to a degree* M *polynomial in its likelihood function (multiplied by* $P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega})$*). This distribution is rarely one from which we can easily sample.*

random-walk chain

The Metropolis-Hastings approach is more feasible in this case. Here, as discussed in section 14.5.3, we can use a random-walk chain *as a proposal distribution and use the data likelihood to compute the acceptance probabilities. In this case, the computation is fairly straightforward, since it involves the ratio of two expressions of the form of equation (19.8), which are the same except for the values of* $\boldsymbol{\lambda}_k$*. Unfortunately, although most of the terms in the numerator and denominator are identical, they appear within the scope of a summation over* k *and therefore do not cancel. Thus, to compute the acceptance probability, we need to compute the full data likelihood for both the current and proposed parameter choices; in this particular network, this computation can be performed fairly efficiently.* ■

Data Completion Collapsed Particles An alternative choice is to use collapsed particles that assign a value to $\mathcal{H}$. In this case, each particle represents a complete data set. Recall that if the prior distribution satisfies certain properties, then we can use closed-form formulas to compute parameter posteriors from $P(\boldsymbol{\lambda} \mid \mathcal{D}, \mathcal{H})$ and to evaluate the marginal likelihood $P(\mathcal{D}, \mathcal{H})$. This implies that if we are using a well-behaved prior, we can evaluate the likelihood of particles in time that does not depend on the network structure.

For concreteness, consider the case where we are learning a Bayesian network with table-CPDs where we have an independent Dirichlet prior over each distribution $P(X_i \mid \mathrm{pa}_{X_i})$. In this case, if we have a particle that represents a complete data instance, we can summarize it by the sufficient statistics $M[x_i, \mathrm{pa}_{x_i}]$, and using these we can compute both the posterior over parameters and the marginal likelihood using closed-form formulas; see section 17.4 and section 18.3.4.

Example 19.12

Let us return to the Bayesian clustering task, but now consider the setting where each particle $\boldsymbol{c}$ *is an assignment to the hidden variables* $C[1], \ldots, C[M]$*. Given an assignment to these variables, we are now in the regime of complete data, in which the different parameters are independent in the posterior. In particular, let* $I_k(\boldsymbol{c})$ *be the set of indexes* $\{m : c[m] = k\}$*. We can now compute the distribution associated with our particle* $\boldsymbol{c}$ *as a Dirichlet posterior*

$$P(\boldsymbol{\theta} \mid \boldsymbol{c}) = \mathit{Dirichlet}(\alpha_0/K + |I_1(\boldsymbol{c})|, \ldots, \alpha_0/K + |I_K(\boldsymbol{c})|).$$

We also have that:

$$P(\boldsymbol{\lambda}_k \mid \boldsymbol{c}, \mathcal{D}, \boldsymbol{\omega}) = P(\boldsymbol{\lambda}_k \mid \mathcal{D}_{I_k(\boldsymbol{c})}, \boldsymbol{\omega}) \propto P(\boldsymbol{\lambda}_k \mid \boldsymbol{\omega}) \prod_{m \in I_k(\boldsymbol{c})} P(\boldsymbol{x}[m] \mid \boldsymbol{\lambda}_k),$$

that is, the posterior over $\boldsymbol{\lambda}_k$ *starting from the prior defined by* $\boldsymbol{\omega}$ *and conditioning on the data instances in* $I_k(\boldsymbol{c})$*. If we now further assume that* $P(\boldsymbol{\lambda} \mid \boldsymbol{\omega})$ *is a conjugate prior to* $Q(\boldsymbol{X} \mid \boldsymbol{\lambda})$*, this posterior can be computed in closed form.*

To apply Gibbs sampling, we also need to specify a distribution for sampling a new value for $C[m']$ *given* $\boldsymbol{c}_{-m'}$*, where again, we use the notation* $-m'$ *to indicate all values* $\{1, \ldots, M\} - \{m'\}$*. Similarly, let* $I_k(\boldsymbol{c}_{-m'})$ *denote the set of indexes* $\{m \neq m' : c[m] = k\}$*. Due to the*

independencies represented by the model structure, we have:

$$P(C[m'] = k \mid \boldsymbol{c}_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto \\ P(C[m'] = k \mid \boldsymbol{c}_{-m'}) P(\boldsymbol{x}[m'] \mid C[m'] = k, \boldsymbol{x}[I_k(\boldsymbol{c}_{-m'})], \boldsymbol{\omega}). \tag{19.9}$$

The second term on the right-hand side is simply a Bayesian prediction over $\boldsymbol{X}$ from the parameter posterior $P(\boldsymbol{\lambda}_k \mid \mathcal{D}_{I_k(\boldsymbol{c}_{-m'})}, \boldsymbol{\omega})$, as defined. Because of the symmetry of the parameters for the different clusters, the term does not depend on m' or on k, but only on the data set on which we condition. We can rewrite this term as $Q(\boldsymbol{X} \mid \mathcal{D}_{I_k(\boldsymbol{c}_{-m'})}, \boldsymbol{\omega})$. The first term on the right-hand side is the prior on cluster assignment for the instance m', as determined by the Dirichlet prior and the assignments of the other instances. Some algebra allows us to simplify this expression, resulting in:

$$P(C[m'] = k \mid \boldsymbol{c}_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto (|I_k(\boldsymbol{c}_{-m'})| + \alpha_0/K) Q(\boldsymbol{X} \mid \mathcal{D}_{I_k(\boldsymbol{c}_{-m'})}, \boldsymbol{\omega}). \tag{19.10}$$

Assuming we have a conjugate prior, this expression can be easily computed. Overall, for most conjugate priors, the cost of computing the sampling distribution for $C[m]$ in this model is $O(MK)$. ■

It turns out that efficient sampling is also possible in more general models; see exercise 19.22. Alternatively we can use a Metropolis-Hastings approach where the proposal distribution can propose to modify several hidden values at once; see exercise 19.23.

Comparison Both types of collapsed particles can be useful for learning in practice, but they have quite different characteristics.

As we discussed, when we use parameter collapsed particles, the cost of evaluating a particle (for example, in a Metropolis-Hastings iteration) is determined by cost of inference in the network. In the worst case, this cost can be exponential, but in many examples it can be efficient. In contrast, the cost of evaluating data collapsed particles depends on properties of the prior. If the prior is properly chosen, the cost is linear in the size of the network.

Another aspect is the space in which we perform MCMC. In the case of parameter collapsed particles, the MCMC procedure is performing integration over a high-dimensional continuous space. The simple Metropolis-Hastings procedures we discussed in this book are usually quite poor for addressing this type of task. However, there is an extensive literature of more efficient MCMC procedures for this task (these are beyond the scope of this book). In the case of data collapsed particles, we perform the integration over parameters in closed form and use MCMC to explore the discrete (but exponential) space of assignments to the unobserved variables. In this problem, relatively simple MCMC methods, such as Gibbs sampling, can be fairly efficient.

To summarize, there is no clear choice between these two options. Both types of collapsed particles can speed up the convergence of the sampling procedure and the accuracy of the estimates of the parameters.

19.3.3 Variational Bayesian Learning

Another class of approximate inference procedures that we can apply to perform Bayesian inference in the case of incomplete data are variational approximations. Here, we can use the methods we developed in chapter 11 to the inference problem posed by Bayesian learning paradigm, resulting in an approach called *variational Bayes*. Recall that, in a variational approximation, we

variational Bayes

aim to find a distribution Q, from a predetermined family of distributions $\mathcal{Q}$, that is close to the real posterior distribution. In our case, we attempt to approximate $P(\mathcal{H}, \boldsymbol{\theta} \mid \mathcal{D})$; thus, the unnormalized measure $\tilde{P}$ in equation (11.3) is $P(\mathcal{H}, \boldsymbol{\theta}, \mathcal{D})$, and our approximating distribution Q is over the parameters and the hidden variables.

Plugging in the variational principle for our problem, and using the fact that $P(\mathcal{H}, \boldsymbol{\theta}, \mathcal{D}) = P(\boldsymbol{\theta})P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})$, we have that the energy functional takes the form:

$$F[P, Q] = \boldsymbol{E}_Q[\log P(\boldsymbol{\theta})] + \boldsymbol{E}_Q[\log P(\mathcal{H}, \mathcal{D} \mid \boldsymbol{\theta})] + \boldsymbol{H}_Q(\boldsymbol{\theta}, \mathcal{H}).$$

The development of such an approximation requires that we decide on the class of approximate distributions we want to consider. While there are many choices here, a natural one is to decouple the posterior over the parameters from the posterior over the missing data. That is, assume that

$$Q(\boldsymbol{\theta}, \mathcal{H}) = Q(\boldsymbol{\theta})Q(\mathcal{H}). \tag{19.11}$$

This is clearly a nontrivial assumption, since our previous discussion shows that these two posteriors are coupled by the data. Nonetheless, we can hope that an approximation that decouples the two distributions will be more tractable.

Recall that, in our discussion of structured variational methods, we saw that the interactions between the structure of the approximation Q and the true distribution P can lead to further structural simplifications in Q (see section 11.5.2.4). Using these tools, we can find the following simplification.

Theorem 19.8 *Let $P(\boldsymbol{\theta})$ be a parameter prior satisfying global parameter independence, $P(\boldsymbol{\theta}) = \prod_i P(\boldsymbol{\theta}_{X_i|\boldsymbol{U}_i})$. Let $\mathcal{D}$ be a partially observable IID data set. If we consider a variational approximation with distributions satisfying $Q(\boldsymbol{\theta}, \mathcal{H}) = Q(\boldsymbol{\theta})Q(\mathcal{H})$, then Q can be decomposed as*

$$Q(\boldsymbol{\theta}, \mathcal{H}) = \prod_i Q(\boldsymbol{\theta}_{X_i|\boldsymbol{U}_i}) \prod_m Q(\boldsymbol{h}[m]).$$

The proof is by direct application of proposition 11.7 and is left as an exercise (exercise 19.24).

This theorem shows that, once we decouple the posteriors over the parameters and missing data, we also lose the coupling between components of the two distributions (that is, different parameters or different instances). Thus, we can further decompose each of the two posteriors into a product of independent terms. This result matches our intuition, since the coupling between the parameters and missing data was the source of dependence between components of the two distributions. That is, the posteriors of two parameters were dependent due to incomplete data, and the posterior of missing data in two instances were dependent due to uncertainty about the parameters.

This theorem does not necessarily justify the (strong) assumption of equation (19.11), but it does suggest that it provides significant computational gains. In this case, we see that we can assume that the approximate posterior also satisfies global parameter independence, and similarly the approximate distribution over $\mathcal{H}$ consists of independent posteriors, one per instance. This simplification already makes the representation of Q much more tractable. Other simplifications, following the same logic, are also possible.

The variational Bayes approach often gives rise to very natural update rules.

Example 19.13

Consider again the Bayesian clustering model of section 19.2.2.4. In this case, we aim to represent the posterior over the parameters $\boldsymbol{\theta}_H, \boldsymbol{\theta}_{X_1|H}, \ldots, \boldsymbol{\theta}_{X_n|H}$ and over the hidden variables $H[1], \ldots, H[M]$. The decomposition of theorem 19.8 allows us write Q as a product distribution, with a term for each of these variables. Thus, we have that

$$Q = Q(\boldsymbol{\theta}_H) \left[\prod_i Q(\boldsymbol{\theta}_{X_i|H})\right] \left[\prod_m Q(H[m])\right].$$

mean field

This factorization is essentially a mean field *approximation. Using the results of section 11.5.1, we see that the fixed-point equations for this approximation are of the form*

$$\begin{aligned}
Q(\boldsymbol{\theta}_H) &\propto \exp\left\{\ln P(\boldsymbol{\theta}_H) + \sum_m \boldsymbol{E}_{Q(H[m])}[\ln P(H[m] \mid \boldsymbol{\theta}_H)]\right\} \\
Q(\boldsymbol{\theta}_{X_i|H}) &\propto \exp\left\{\ln P(\boldsymbol{\theta}_{X_i|H}) + \sum_m \boldsymbol{E}_{Q(H[m])}\left[\ln P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H})\right]\right\} \\
Q(H[m]) &\propto \exp\Big\{\boldsymbol{E}_{Q(\boldsymbol{\theta}_H)}[\ln P(H[m] \mid \boldsymbol{\theta}_H)] \\
&\qquad + \sum_i \boldsymbol{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\left[\ln P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H})\right]\Big\}.
\end{aligned}$$

The application of the mean-field theory allows us to identify the structure of the update equation. To provide a constructive solution, we also need to determine how to evaluate the expectations in these update equations. We now examine these expectations in the case where all the variables are binary and the priors over parameters are simple Dirichlet distributions (Beta distributions, in fact).

We start with the first fixed-point equation. A value for $\boldsymbol{\theta}_H$ is a pair $\langle \theta_{h^0}, \theta_{h^1} \rangle$. Using the definition of the Dirichlet prior, we have that

$$\ln P(\boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle) = \ln c + (\alpha_{h^0} - 1) \ln \theta_{h^0} + (\alpha_{h^1} - 1) \ln \theta_{h^1},$$

where α_{h^0} and α_{h^1} are the hyperparameters of the prior $P(\boldsymbol{\theta}_H)$, and c is the normalizing constant of the prior (which we can ignore). Similarly, we can see that

$$\boldsymbol{E}_{Q(H[m])}[\ln P(H[m] \mid \boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle)] = Q(H[m] = h^0) \ln \theta_{h^0} + Q(H[m] = h^1) \ln \theta_{h^1}.$$

Combining these results, we get that

$$\begin{aligned}
Q(\boldsymbol{\theta}_H = \langle \theta_{h^0}, \theta_{h^1} \rangle) &\propto \exp\left\{\left(\alpha_{h^0} + \sum_m Q(H[m] = h^0) - 1\right) \ln \theta_{h^0} + \right. \\
&\qquad \left. \left(\alpha_{h^1} + \sum_m Q(H[m] = h^1) - 1\right) \ln \theta_{h^1}\right\} \\
&= \theta_{h^0}^{\alpha_{h^0} + \sum_m Q(H[m]=h^0) - 1} \theta_{h^1}^{\alpha_{h^1} + \sum_m Q(H[m]=h^1) - 1}.
\end{aligned}$$

In other words, $Q(\boldsymbol{\theta}_H)$ is a Beta distribution with hyperparameters

$$\begin{aligned}\alpha'_{h^0} &= \alpha_{h^0} + \sum_m Q(H[m] = h^0) \\ \alpha'_{h^1} &= \alpha_{h^1} + \sum_m Q(H[m] = h^1).\end{aligned}$$

Note that this is exactly the Bayesian update for θ_H with the expected sufficient statistics given $Q(\mathcal{H})$.

A similar derivation shows that $Q(\boldsymbol{\theta}_{X_i|H})$ is also a pair of independent Beta distributions (one for each value of H) that are updated with the expected sufficient statistics given $Q(\mathcal{H})$.

M-step

*These updates are reminiscent of the EM-update (*M-step*), since we use expected sufficient statistics to update the posterior. In the EM M-step, we update the MLE using the expected sufficient statistics. If we carry the analogy further, the last fixed-point equation, which updates $Q(H[m])$, corresponds*

E-step

to the E-step, since it updates the expectations over the missing values. Recall that, in the E-step *of EM, we use the current parameters to compute*

$$Q(H[m]) = P(H[m] \mid x_1[m], \ldots x_n[m]) \propto P(H[m] \mid \boldsymbol{\theta}_H) \prod_i P(x_i[m] \mid H[m], \boldsymbol{\theta}_{X_i|H}).$$

If we were doing a Bayesian approach, we would not simply take our current values for the parameters $\boldsymbol{\theta}_H, \boldsymbol{\theta}_{X_i|H}$; rather, we would average over their posteriors. Examining this last fixed-point equation, we see that we indeed average over the (approximate) posteriors $Q(\boldsymbol{\theta}_H)$ and $Q(\boldsymbol{\theta}_{X_i|H})$. However, unlike standard Bayesian averaging, where we compute the average value of the parameter itself, here we average its logarithm; that is, we evaluate terms of the form

$$\mathbf{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\big[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\big] = \int_0^1 Q(\theta_{x_i|H[m]}) \ln \theta_{x_i|H[m]} d\theta_{x_i|H[m]}.$$

Using methods that are beyond the scope of this book, one can show that this integral has a closed-form solution:

$$\mathbf{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\big[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\big] = \varphi(\alpha'_{x_i|h}) - \varphi(\sum_{x'_i} \alpha'_{x'_i|h}),$$

where $\boldsymbol{\alpha}'$ are the hyperparameters of the posterior approximation in $Q(\boldsymbol{\theta}_{X_i|H})$ and $\varphi(z) = (\ln \Gamma(z))' = \frac{\Gamma'(z)}{\Gamma(z)}$ is the digamma function*, which is equal to $\ln(z)$ plus a polynomial function of $\frac{1}{z}$. And so, for $z \gg 1$, $\varphi(z) \approx \ln(z)$. Using this approximation, we see that*

digamma function

$$\mathbf{E}_{Q(\boldsymbol{\theta}_{X_i|H})}\big[\ln P(x_i \mid H[m], \boldsymbol{\theta}_{X_i|H})\big] \approx \ln \frac{\alpha'_{x_i|h}}{\sum_{x'_i} \alpha'_{x'_i|h}},$$

that is, the logarithm of the expected conditional probability according to the posterior $Q(\boldsymbol{\theta}_{X_i|H})$. This shows that if the posterior hyperparameters are large the variational update is almost identical to EM's E-step.

To wrap up, we applied the structured variational approximation to the Bayesian learning problem. Using the tools we developed in previous chapters, we defined tractable fixed-point equations.

As with the mean field approximation we discussed in section 11.5.1, we can find a fixed-point solution for Q by iteratively applying these equations.

The resulting algorithm is very similar to applications of EM. Applications of the update equations for the parameters are almost identical to standard EM of section 19.2.2.4 in the sense that we use expected sufficient statistics. However, instead of finding the MLE parameters given these expected sufficient statistics, we compute the posterior assuming these were observed. The update for $Q(H[m])$ is reminiscent to the computation of $P(H[m])$ when we know the parameters. However, instead of using parameter values we use expectations of their logarithm and then take the exponent. ■

This example shows that we can find a variational approximation to the Bayesian posterior using an EM-like algorithm in which we iterate between updates to the parameter posteriors and updates to the missing data posterior. These ideas generalize to other network structures in a fairly straightforward way. The update for the posterior over parameter is similar to Bayesian update with expected sufficient statistics, and the update of the posterior over hidden variable is similar to a computation with the expected parameters (with the differences discussed earlier). In more complex examples we might need to make further assumptions about the distribution Q in order to get a tractable approximation. For example, if there are multiple missing values per instance, then we might not be able to afford to represent their distribution by the joint distribution and would instead need to introduce structure into Q. The basic ideas are similar to ones we explored before, and so we do not elaborate them. See exercise 15.6 for one example.

Of course, this method has some clear drawbacks. Because we are representing the parameter posterior by a factored distribution, we cannot expect to represent a multimodal posterior. Unfortunately, we know that the posterior is often multimodal. For example, in the clustering problem, we know that change in names of values of H would not change the prediction. Thus, the posterior in this example should be symmetric under such renaming. This implies that a unimodal distribution can only be a partial approximation to the true posterior. In multimodal cases, the effect of the variational approximation cannot be predicted. It may select one of the peaks and try to approximate it using Q, or it may choose a "broad" distribution that averages over some or all of the peaks.

19.4 Structure Learning

We now move to discuss the more complex task of learning the network structure as well as the parameters, again in the presence of incomplete data. Recall that in the case of complete data, we started by defining a score for evaluating different network structures and then examined search procedures that can maximize this score. As we will see, **both components of structure learning — the scoring function and the search procedure — are considerably more complicated in the case of incomplete data. Moreover, in the presence of hidden variables, even our search space becomes significantly more complex, since we now have to select the value space for the hidden variables, and even the number of hidden variables that the model contains.**

19.4.1 Scoring Structures

In section 18.3, we defined three scores: the likelihood score, the BIC score, and the Bayesian score. As we discussed, the likelihood score does not penalize more complex models, and it is therefore not useful when we want to compare between models of different complexity. Both the BIC and Bayesian score have built-in penalization for complex models and thus trade off the model complexity with its fit to the data. Therefore, they are far less likely to overfit.

We now consider how to extend these scores to the case when some of the data are missing. On the face of it, the score we want to evaluate is the same Bayesian score we considered in the case of complete data:

$$\text{score}_{\mathcal{B}}(\mathcal{G} \ : \ \mathcal{D}) = \log P(\mathcal{D} \mid \mathcal{G}) + \log P(\mathcal{G})$$

where $P(\mathcal{D} \mid \mathcal{G})$ is the marginal likelihood of the data:

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

In the complete data case, the likelihood term inside the integral had a multiplicative factorization, and thus we could simplify it. In the case of incomplete data, the likelihood involves summing out over the unobserved variables, and thus it does not decompose.

As we discussed, we can view the computation of the marginal likelihood as an inference problem. For most learning problems with incomplete data, this inference problem is a difficult one. We now consider different strategies for dealing with this issue.

19.4.1.1 Laplace Approximation

One approach for approximating an integral in a high-dimensional space is to provide a simpler approximation to it, which we can then integrate in closed form. One such method is the *Laplace approximation*, described in box 19.F.

Laplace approximation

Box 19.F — Concept: Laplace Approximation. *The Laplace approximation can be applied to any function of the form $f(\boldsymbol{w}) = e^{g(\boldsymbol{w})}$ for some vector $\boldsymbol{w}$. Our task is to compute the integral*

$$F = \int f(\boldsymbol{w}) d\boldsymbol{w}.$$

Using Taylor's expansion, we can expand an approximation of g around a point $\boldsymbol{w}_0$

$$g(\boldsymbol{w}) \approx g(\boldsymbol{w}_0) + \left[\frac{\partial g(\boldsymbol{w})}{\partial x_i}\right]\Big|_{\boldsymbol{w}=\boldsymbol{w}_0} (\boldsymbol{w} - \boldsymbol{w}_0) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^T \left[\frac{\partial \partial g(\boldsymbol{w})}{\partial x_i \partial x_j}\right]\Big|_{\boldsymbol{w}=\boldsymbol{w}_0} (\boldsymbol{w} - \boldsymbol{w}_0),$$

where $\left[\frac{\partial g(\boldsymbol{w})}{\partial x_i}\right]\Big|_{\boldsymbol{w}=\boldsymbol{w}_0}$ denotes the vector of first derivatives and $\left[\frac{\partial \partial g(\boldsymbol{w})}{\partial x_i \partial x_j}\right]\Big|_{\boldsymbol{w}=\boldsymbol{w}_0}$ denotes the Hessian *— the matrix of second derivatives.*

Hessian

If $\boldsymbol{w}_0$ is the maximum of $g(\boldsymbol{w})$, then the second term disappears. We now set

$$\boldsymbol{C} = -\left[\frac{\partial^2 g(\boldsymbol{w})}{\partial x_i \partial x_j}\right]\Big|_{\boldsymbol{w}=\boldsymbol{w}_0}$$

to be the negative of the matrix of second derivatives of $g(\boldsymbol{w})$ at $\boldsymbol{w}_0$. Since $\boldsymbol{w}_0$ is a maximum, this matrix is positive semi-definitive. Thus, we get the approximation

$$g(\boldsymbol{w}) \approx g(\boldsymbol{w}_0) - \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^T \boldsymbol{C}(\boldsymbol{w} - \boldsymbol{w}_0).$$

Plugging this approximation into the definition of $f(x)$, we can write

$$\int f(\boldsymbol{w}) d\boldsymbol{w} \approx f(\boldsymbol{w}_0) \int e^{-\frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}_0)^T \boldsymbol{C}(\boldsymbol{w}-\boldsymbol{w}_0)} d\boldsymbol{w}.$$

The integral is identical to the integral of an unnormalized Gaussian distribution with covariance matrix $\Sigma = C^{-1}$. We can therefore solve this integral analytically and obtain:

$$\int f(\boldsymbol{w}) d\boldsymbol{w} \approx f(\boldsymbol{w}_0) |\boldsymbol{C}|^{-\frac{1}{2}} (2\pi)^{\frac{1}{2}\dim(\boldsymbol{C})},$$

where $\dim(C)$ is the dimension of the matrix $\boldsymbol{C}$.

At a high level, the Laplace approximation uses the value at the maximum and the curvature (the matrix of second derivatives) to approximate the integral of the function. This approximation works well when the function f is dominated by a single peak that has roughly a Gaussian shape.

How do we use the Laplace approximation in our setting? Taking g to be the log-likelihood function combined with the prior $\log P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{G}) + \log P(\boldsymbol{\theta}|\mathcal{G})$, we get that $\log P(\mathcal{D}, \mathcal{G})$ can be approximated by the *Laplace score*:

Laplace score

$$\text{score}_{Laplace}(\mathcal{G} \ : \ \mathcal{D}) = \log P(\mathcal{G}) + \log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_{\mathcal{G}}, \mathcal{G}) + \frac{\dim(\boldsymbol{C})}{2} \log 2\pi - \frac{1}{2} \log |\boldsymbol{C}|,$$

where $\tilde{\boldsymbol{\theta}}_{\mathcal{G}}$ are the MAP parameters and $\boldsymbol{C}$ is the negative of the Hessian matrix of the log-likelihood function. More precisely, the entries of $\boldsymbol{C}$ are of the form

$$- \left. \frac{\partial^2 \log P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{G})}{\partial \theta_{x_i|\boldsymbol{u}_i} \partial \theta_{x_j|\boldsymbol{u}_j}} \right|_{\tilde{\boldsymbol{\theta}}_{\mathcal{G}}} = - \sum_m \left. \frac{\partial^2 \log P(\boldsymbol{o}[m] \mid \boldsymbol{\theta}, \mathcal{G})}{\partial \theta_{x_i|\boldsymbol{u}_i} \partial \theta_{x_j|\boldsymbol{u}_j}} \right|_{\tilde{\boldsymbol{\theta}}_{\mathcal{G}}},$$

where $\theta_{x_i|\boldsymbol{u}_i}$ and $\theta_{x_j|\boldsymbol{u}_j}$ are two parameters (not necessarily from the same CPD) in the parameterization of the network.

The Laplace score takes into account not only the number of free parameters but also the curvature of the posterior distribution in each direction. Although the form of this expression arises directly by approximating the posterior marginal likelihood, it is also consistent with our intuitions about the desired behavior. Recall that the parameter posterior is a concave function, and hence has a negative definitive Hessian. Thus, the negative Hessian C is positive definite and therefore has a positive determinant. A large determinant implies that the curvature at the MAP point is sharp; that is, the peak is relatively narrow and most of its mass is at the maximum. In this case, the model is probably overfitting to the training data, and we incur a large penalty. Conversely, if the curvature is small, the peak is wider, and the mass of the posterior is distributed over a larger set of parameters. In this case, overfitting is less likely, and, indeed, the Laplace score imposes a smaller penalty on the model.

To compute the Laplace score, we first need to use one of the methods we discussed earlier to find the MAP parameters of the distribution, and then compute the Hessian matrix. The computation of the Hessian is somewhat involved. To compute the entry for the derivative relative to $\theta_{x_i|\boldsymbol{u}_i}$ and $\theta_{x_j|\boldsymbol{u}_j}$, we need to compute the joint distribution over $x_i, x_j, \boldsymbol{u}_i, \boldsymbol{u}_j$ given the observation; see exercise 19.9. Because these variables are not necessarily together in a clique (or cluster), the cost of doing such computations can be much higher than computing the likelihood. Thus, this approximation, while tractable, is still expensive in practice.

19.4.1.2 Asymptotic Approximations

One way of avoiding the high cost of the Laplace approximation is to approximate the term $|\boldsymbol{C}|^{-\frac{1}{2}}$. Recall that the likelihood is the sum of the likelihood of each instance. Thus, the Hessian matrix is the sum of many Hessian matrixes, one per instance. We can consider asymptotic approximations that work well when the number of instances grows ($M \to \infty$). For this analysis, we assume that all data instances have the same observation pattern; that is, the set of variables $\boldsymbol{O}[m] = \boldsymbol{O}$ for all m.

Consider the matrix $\boldsymbol{C}$. As we just argued, this matrix has the form

$$\boldsymbol{C} = \sum_{m=1}^{M} \boldsymbol{C}_m,$$

where $\boldsymbol{C}_m$ is the negative of the hessian of $\log P(\boldsymbol{o}[m] \mid \boldsymbol{\theta}, \mathcal{G})$. We can view each $\boldsymbol{C}_m$ as a sample from a distribution that is induced by the (random) choice of assignment $\boldsymbol{o}$ to $\boldsymbol{O}$; each assignment $\boldsymbol{o}$ induces a different matrix $\boldsymbol{C}_{\boldsymbol{o}}$. We can now rewrite:

$$\boldsymbol{C} = M \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{C}_m.$$

As M grows, the term $\frac{1}{M}\sum_{m=1}^{M} \boldsymbol{C}_m$ approaches the expectation $\boldsymbol{E}_{P^*}[\boldsymbol{C}_{\boldsymbol{o}}]$.

Taking the determinant of both sides, and recalling that $\det(\alpha A) = \alpha^{\dim(A)} \det(A)$, we get

$$\det(\boldsymbol{C}) = M^{\dim(\boldsymbol{C})} \det\left(\frac{1}{M}\sum_{m=1}^{M} \boldsymbol{C}_m\right) \approx M^{\dim(\boldsymbol{C})} \det\left(\boldsymbol{E}_{P^*}[\boldsymbol{C}_{\boldsymbol{o}}]\right).$$

Taking logarithms of both sides, we get that

$$\log \det(\boldsymbol{C}) \approx \dim(\boldsymbol{C}) \log M + \log \det\left(\boldsymbol{E}_{P^*}[\boldsymbol{C}_{\boldsymbol{o}}]\right).$$

Notice that the last term does not grow with M. Thus, when we consider the asymptotic behavior of the score, we can ignore it. This rough argument is the outline of the proof for the following result.

Theorem 19.9 *As $M \to \infty$, we have that:*

$$\text{score}_{Laplace}(\mathcal{G} : \mathcal{D}) = \text{score}_{BIC}(\mathcal{G} : \mathcal{D}) + O(1)$$

BIC score *where* $\text{score}_{BIC}(\mathcal{G} : \mathcal{D})$ *is the* BIC score

$$\text{score}_{BIC}(\mathcal{G} : \mathcal{D}) = \log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_{\mathcal{G}}, \mathcal{G}) - \frac{\log M}{2}\text{Dim}[\mathcal{G}] + \log P(\mathcal{G}) + \log P(\tilde{\boldsymbol{\theta}}_{\mathcal{G}} \mid \mathcal{G}).$$

This result shows that the BIC score is an asymptotic approximation to the Laplace score, a conclusion that is interesting for several important reasons. First, it shows that the intuition we had for the case of complete data, where the score trades off the likelihood of the data with a structure penalty, still holds. Second, as in the complete data case, the asymptotic behavior of this penalty is logarithmic in the number of samples; this relationship implies the rate at which more instances can lead us to introduce new parameters.

independent parameters

An important subtlety in this analysis is hidden in the use of the notation $\mathrm{Dim}[\mathcal{G}]$. In the case of complete data, this notation stood for the number of *independent parameters* in the network, a quantity that we could easily compute. Here, it turns out that for some models, the actual number of degrees of freedom is smaller than the space of parameters. This implies that the matrix $\boldsymbol{C}$ is not of full rank, and so its determinant is 0. In such cases, we need to perform a variant of the Laplace approximation in the appropriate subspace, which leads to a determinant of a smaller matrix. The question of how to determine the right number of degrees of freedom (and thus the magnitude of $\mathrm{Dim}[\mathcal{G}]$) is still an open problem.

19.4.1.3 Cheeseman-Stutz Approximation

We can use the Laplace/BIC approximations to derive an even tighter approximation to the Bayesian score. The intuition is that, in the case of complete data, the full Bayesian score was more precise than the BIC score since it took into account the extent to which each parameter was used and how its range of values influenced the likelihood. These considerations are explicit in the integral form of the likelihood and implicit in the closed-form solution of the integral. When we use the BIC score on incomplete data, we lose these fine-grained distinctions in evaluating the score.

Recall that the closed-form solution of the Bayesian score is a function of the sufficient statistics of the data. An ad hoc approach for constructing a similar (approximate) score when we have incomplete data is to apply the closed-form solution of the Bayesian score on some approximation of the statistics of the data. A natural choice would be the expected sufficient statistics given the MAP parameters. These expected sufficient statistics represent the completion of the data given our most likely estimate of the parameters.

More formally, for a network $\mathcal{G}$ and a set of parameters $\boldsymbol{\theta}$, we define $\mathcal{D}^*_{\mathcal{G},\boldsymbol{\theta}}$ to be a fictitious "complete" data set whose actual counts are the same as the *fractional* expected counts relative to this network; that is, for every event $\boldsymbol{x}$:

$$M_{\mathcal{D}^*_{\mathcal{G},\boldsymbol{\theta}}}[\boldsymbol{x}] = \bar{M}_{P(\mathcal{H}|\mathcal{D},\boldsymbol{\theta},\mathcal{G})}[\boldsymbol{x}]. \tag{19.12}$$

Because the expected counts are based on a coherent distribution, there can be such a data set (although it might have instances with fractional weights). To evaluate a particular network $\mathcal{G}$, we define the data set $\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_{\mathcal{G}}}$ induced by our network $\mathcal{G}$ and its MAP parameters $\tilde{\boldsymbol{\theta}}_{\mathcal{G}}$, and approximate the Bayesian score $P(\mathcal{D} \mid \mathcal{G})$ by $P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_{\mathcal{G}}} \mid \mathcal{G})$, using the standard integration over the parameters.

While straightforward in principle, a closer look suggests that this approximation cannot be a very good one. The first term,

$$P(\mathcal{D} \mid \mathcal{G}) = \int \sum_{\mathcal{H}} p(\mathcal{D}, \mathcal{H} \mid \boldsymbol{\theta}, \mathcal{G}) P(\boldsymbol{\theta} \mid \mathcal{G}) d\boldsymbol{\theta} = \sum_{\mathcal{H}} \int p(\mathcal{D}, \mathcal{H} \mid \boldsymbol{\theta}, \mathcal{G}) P(\boldsymbol{\theta} \mid \mathcal{G}) d\boldsymbol{\theta},$$

involves a summation of exponentially many integrals over the parameter space — one for each assignment to the hidden variables $\mathcal{H}$. On the other hand, the approximating term

$$P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) = \int p(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \boldsymbol{\theta}, \mathcal{G}) P(\boldsymbol{\theta} \mid \mathcal{G}) d\boldsymbol{\theta}$$

is only a single such integral. In both terms, the integrals are over a "complete" data set, so that one of these sums is on a scale that is exponentially larger than the other.

One ad hoc solution is to simply correct for this discrepancy by estimating the difference:

$$\log P(\mathcal{D} \mid \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}).$$

We use the asymptotic Laplace approximation to write each of these terms, to get:

$$\begin{aligned}
\log P(\mathcal{D} \mid \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) &\approx \left[\log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}) - \frac{1}{2}\mathrm{Dim}[\mathcal{G}] \log M\right] \\
&\quad - \left[\log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}) - \frac{1}{2}\mathrm{Dim}[\mathcal{G}] \log M\right] \\
&= \log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}).
\end{aligned}$$

The first of these terms is the log-likelihood achieved by the MAP parameters on the observed data. The second is the log-likelihood on the fictional data set, a term that can be computed in closed form based on the statistics of the fictional data set. We see that the first term is, again, a summation of an exponential number of terms representing different assignments to $\mathcal{H}$. We note that the Laplace approximation is valid only at the large sample limit, but more careful arguments can show that this construction is actually fairly accurate for a large class of situations.

Putting these arguments together, we can write:

$$\begin{aligned}
\log P(\mathcal{D} \mid \mathcal{G}) &= \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) + \log P(\mathcal{D} \mid \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) \\
&\approx \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) + \log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}).
\end{aligned}$$

Cheeseman-Stutz score

This approximation is the basis for the *Cheeseman-Stutz score*:

$$\mathrm{score}_{CS}(\mathcal{G} : \mathcal{D}) = \log P(\mathcal{G}) + \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \mathcal{G}) + \log P(\mathcal{D} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G}) - \log P(\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_\mathcal{G}} \mid \tilde{\boldsymbol{\theta}}_\mathcal{G}, \mathcal{G})$$

The appealing property of the Cheeseman-Stutz score is that, unlike the BIC score, it uses the closed-form solution of the complete data marginal likelihood in the context of incomplete data. Experiments in practice (see box 19.G) show that this score is much more accurate than the BIC score and much cheaper to evaluate than the Laplace score.

19.4.1.4 Candidate Method

candidate method

Another strategy for approximating the score is the *candidate method*; it uses a particular choice of parameters (the *candidate*) to evaluate the marginal likelihood. Consider any set of parameters $\boldsymbol{\theta}$. Using the chain law of probability, we can write $P(\mathcal{D}, \boldsymbol{\theta} \mid \mathcal{G})$ in two different ways:

$$\begin{aligned}
P(\mathcal{D}, \boldsymbol{\theta} \mid \mathcal{G}) &= P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{G}) P(\boldsymbol{\theta} \mid \mathcal{G}) \\
P(\mathcal{D}, \boldsymbol{\theta} \mid \mathcal{G}) &= P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G}) P(\mathcal{D} \mid \mathcal{G}).
\end{aligned}$$

Equating the two right-hand terms, we can write

$$P(\mathcal{D} \mid \mathcal{G}) = \frac{P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{G}) P(\boldsymbol{\theta} \mid \mathcal{G})}{P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G})}. \tag{19.13}$$

The first term in the numerator is the likelihood of the observed data given $\boldsymbol{\theta}$, which we should be able to evaluate using inference (exact or approximate). The second term in the numerator is the prior over the parameters, which is usually given. The denominator is the posterior over the parameters, the term most difficult to approximate.

The candidate method reduces the problem of computing the marginal likelihood to the problem of generating a reasonable approximation to the parameter posterior $P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G})$. It lets us estimate the likelihood when using methods such as MCMC sampling to approximate the posterior distribution. Of course, the quality of our approximation depends heavily on the design of the MCMC sampler. If we use a simple sampler, then the precision of our estimate of $P(\boldsymbol{\theta} \mid \mathcal{D}, \mathcal{G})$ will be determined by the number of sampled particles (since each particle either has these parameters or not). If, instead, we use collapsed particles, then each particle will have a distribution over the parameters, providing a better and smoother estimate for the posterior.

The quality of our estimate also depends on the particular choice of candidate $\boldsymbol{\theta}$. We can obtain a more robust estimate by averaging the estimates from several choices of candidate parameters (say several likely parameter assignments based on our simulations). However, each of these requires inference for computing the numerator in equation (19.13), increasing the cost.

An important property of the candidate method is that equation (19.13), on which the method is based, is not an approximation. Thus, if we could compute the denominator exactly, we would have an exact estimate for the marginal likelihood. This gives us the option of using more computational resources in our MCMC approximation to the denominator, to obtain increasingly accurate estimates. By contrast, the other methods all rely on an asymptotic approximation to the score and therefore do not offer a similar trade-off of accuracy to computational cost.

19.4.1.5 Variational Marginal Likelihood

A different approach to estimating the marginal likelihood is using the variational approximations we discussed in section 19.3.3. Recall from corollary 19.1 that, for any distribution Q,

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = F_{\mathcal{D}}[\boldsymbol{\theta}, Q] + \boldsymbol{D}(Q(\mathcal{H}) \| P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})).$$

It follows that the energy functional is a lower bound of the marginal likelihood, and the difference between them is the relative entropy between the approximate posterior distribution and the true one. Thus, if we find a good approximation Q of the posterior $P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$, then the relative entropy term is small, so that that energy functional is a good approximation of the marginal likelihood. As we discussed, the energy functional itself has the form:

$$F_{\mathcal{D}}[\boldsymbol{\theta}, Q] = \boldsymbol{E}_{\mathcal{H} \sim Q}[\ell(\boldsymbol{\theta} : \mathcal{D}, \mathcal{H})] + \boldsymbol{H}_Q(\mathcal{H}).$$

Both of these terms can be computed using inference relative to the distribution Q. Because this distribution was chosen to allow tractable inference, this provides a feasible approach for approximating the marginal likelihood.

mixture distribution

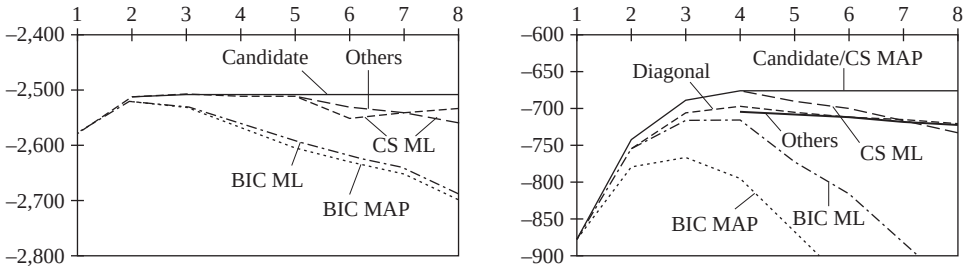
Box 19.G — Case Study: Evaluating Structure Scores. *To study the different approximations to the Bayesian score in a restricted setting, Chickering and Heckerman (1997) consider learning a naive Bayes* mixture distribution*, as in section 19.2.2.4, where the cardinality K of the hidden variable (the number of mixture components) is unknown. Adding more values to the class variables increases the representational power of the model, but also introduces new parameters and thus increases the ability of the model to overfit the data. Since the class of distributions that are representable with a cardinality of K is contained within those that are representable with a cardinality of $K' > K$, the likelihood score increases monotonically with the cardinality of the class variable. The question is whether the different structure scores can pinpoint a good cardinality for the hidden variable. To do so, they perform MAP parameter learning on structures of different cardinality and then evaluate the different scores. Since the structure learning problem is one-dimensional (in the sense that the only parameter to learn is the cardinality of the class variable), there is no need to consider a specific search strategy in the evaluation.*

It is instructive to evaluate performance on both real data, and on synthetic data where the true number of clusters is known. However, even in synthetic data cases, where the true cardinality of the hidden variable is known, using this true cardinality as the "gold standard" for evaluating methods may not be appropriate, as with few data instances, the "optimal" model may be one with fewer parameters. Thus, Chickering and Heckerman instead compare all methods to the candidate method, using MCMC to evaluate the denominator; with enough computation, one can use this method to obtain high-quality approximations to the correct marginal likelihood.

The data in the synthetic experiments were generated from a variety of models, which varied along several axes: the true cardinality of the hidden variable (d); the number of observed variables (n); and the number of instances (M). The first round of experiments revealed few differences between the different scores. An analysis showed that this was because synthetic data sets with random parameter choices are too easy. Because of the relatively large number of observed variables, such random models always had distinguished clusters. That is, using the true parameters, the posterior probability $P(c \mid x_1, \ldots, x_n)$ is close to 1 for the true cluster value and 0 for all others. Thus, the instances belonging to different clusters are easily separable, making the learning problem too easy.

To overcome this problem, they considered sampling networks where the values of the parameters for $P(X_i \mid c)$ are correlated for different values of c. If the correlation is absolute, then the clusters overlap. For intermediate correlation the clusters were overlapping but not identical. By tuning the degree of correlation in sampling the distribution, they managed to generate networks with different degree of separation between the clusters. On data sets where the generating distribution did not have any separation between clusters, all the scores preferred to set the cardinality of the cluster variable to 1, as expected. When they examined data sets where the generating distribution had partial overlap between clusters they saw differentiating behavior between scoring methods. They also performed this same analysis on several real-world data sets. Figure 19.G.1 demonstrates the results for two data sets and summarizes the results for many of the synthetic data sets, evaluating the ability of the different methods to come close to the "optimal" cardinality, as determined by the candidate method.

Overall, the results suggest that BIC tends to underfit badly, almost always selecting models with an overly low cardinality for the hidden variable; moreover, its score estimate for models of higher (and more appropriate) cardinality tended to decrease very sharply, making it very unlikely that



Generating Model			Error in cluster cardinality				
n	d	M	Laplace	CS MAP	CS ML	BIC MAP	BIC ML
32	4	400	0	0	0	0	0
64	4	400	0	0	0	0	0
128	4	400	0.2	0.2	0.2	1	1
64	4	400	0	0	0	0	0
64	6	400	0.4	0.4	0.4	0.4	0.4
64	8	400	0.2	0.6	1	1	1
64	4	50	0.2	0	0.4	0.6	0.6
64	4	100	0	0	0.2	0.8	0.6
64	4	200	0	0	0	0	0

Figure 19.G.1 — Evaluation of structure scores for a naive Bayes clustering model In the graphs in the top row, the x axis denotes different cluster cardinalities, and the y axis the marginal likelihood estimated by the method. The graph on the left represents synthetic data with $d = 4$, $n = 128$, and $M = 400$. The graph on the right represents a real-world data set, with $d = 4$, $n = 35$ and $M = 47$. The table at bottom shows errors in model selection for the number of values of a hidden variable, as made by different approximations to the marginal likelihood. The errors are computed as differences between the cardinality selected by the method and the "optimal" cardinality selected by the "gold standard" candidate method. The errors are averaged over five data sets. The blocks of lines correspond to experiments where one of the three parameters defining the synthetic network varied while the others were held constant. (Adapted from Chickering and Heckerman (1997), with permission.)

they would be chosen. The other asymptotic approximations were all reasonably good, although all of them tended to underestimate the marginal likelihood as the number of clusters grows. A likely reason is that many of the clusters tend to become empty in this setting, giving rise to a "ridge-shaped" likelihood surface, where many parameters have no bearing on the likelihood. In this case, the "peak"-shaped estimate of the likelihood used by the asymptotic approximations tends to underestimate the true value of the integral. Among the different asymptotic approximations, the Cheeseman-Stutz approximation using the MAP configuration of the parameters had a slight edge over the other methods in its accuracy, and was more robust when dealing with parameters that are close to 0 or 1. It was also among the most efficient of the methods (other than the highly inaccurate BIC approach).

19.4.2 Structure Search

19.4.2.1 A Naive Approach

Given a definition of the score, we can now consider the structure learning task. In the most general terms, we want to explore the set of graph structures involving the variables of interest, score each one of these, and select the highest-scoring one. For some learning problems, such as the one discussed in box 19.G, the number of structures we consider is relatively small, and thus we can simply systematically score each structure and find the best one.

This strategy, however, is infeasible for most learning problems. Usually the number of structures we want to consider is very large — exponential or even superexponential in the number of variables — and we do not want to score all them. In section 18.4, we discussed various optimization procedures that can be used to identify a high-scoring structures. As we showed, for certain types of constraints on the network structure — tree-structured networks or a given node ordering (and bounded indegree) — we can actually find the optimal structure efficiently. In the more general case, we apply a hill-climbing procedure, using search operators that consist of local network modifications, such as edge addition, deletion, or reversal.

Unfortunately, the extension of these methods to the case of learning with missing data quickly hits a wall, since all of these methods relied on the decomposition of the score into a sum of individual family scores. This requirement is obvious in the case of learning tree-structured networks and in the case of learning with a fixed ordering: in both cases, the algorithm relied explicitly on the decomposition of the score as a sum of family scores.

The difficulty is a little more subtle in the case of the hill-climbing search. There, in each iteration, we consider applying $O(n^2)$ possible search operators (approximately 1–2 operators for each possible edge in the network). This number is generally quite large, so that the evaluation of the different possible steps in the search is a significant computational bottleneck. Although the same issue arises in the complete data case, there we could significantly reduce the computational burden due to two ideas. First, since the score is based on sufficient statistics, we could cache sufficient statistics and reuse them. Second, since the score is decomposable, the change in score of many of the operators is oblivious to modifications in another part of the network. Thus, as we showed in section 18.4.3.3, once we compute the delta-score of an operator o relative to a candidate solution $\mathcal{G}$:

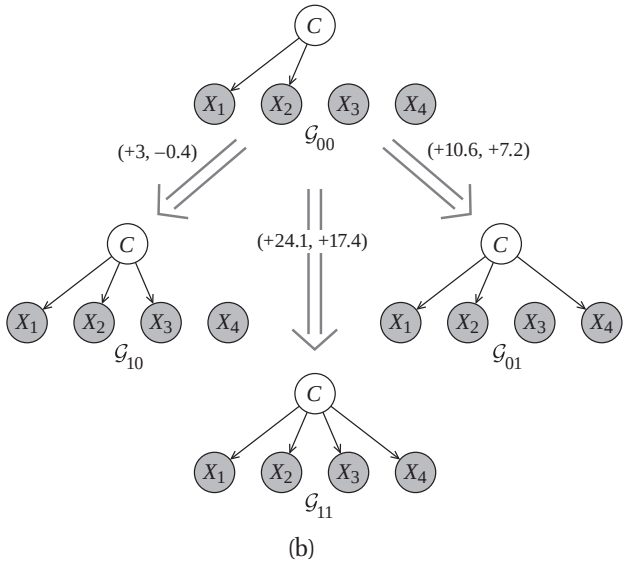
$$\delta(\mathcal{G} : o) = \text{score}(o(\mathcal{G}) : \mathcal{D}) - \text{score}(\mathcal{G} : \mathcal{D}),$$

the same quantity is also the delta-score $\delta(\mathcal{G}' : o)$ for any other $\mathcal{G}'$ that is similar to $\mathcal{G}$ in the local topology that is relevant for o. For example, if o adds $X \to Y$, then the delta-score remains unchanged for any graph $\mathcal{G}'$ for which the family of Y is the same as in $\mathcal{G}$. The decomposition property implied that the search procedure in the case of complete data could maintain a priority queue of the effect of different search operators from previous iterations of the algorithm and avoid repeated computations.

When learning from incomplete data, the situation is quite different. As we discussed, local changes in the structure can result in global changes in the likelihood function. Thus, after a local structure change, the parameters of all the CPDs might change. As a consequence, the score is not decomposable; that is, the delta-score of one local modification (for example, adding an arc) can change after we modify a remote part of the network.

X_1	X_2	X_3	X_4	Count
0	0	0	1	2
0	0	1	0	1
0	1	0	0	2
0	1	0	1	1
0	1	1	0	8
0	1	1	1	1
1	0	0	0	3
1	0	0	1	42
1	0	1	0	10
1	0	1	1	11
1	1	0	1	15
1	1	1	0	3
1	1	1	1	1

(a)



(b)

Figure 19.10 Nondecomposability of structure scores in the case of missing data. (a) A training set over variables $X_1, \ldots, X_4$. (b) Four possible networks over $X_1, \ldots, X_4$ and a hidden variable C. Arrows from the top network to the other three are labeled with the change in log-likelihood (LL) and Cheeseman-Stutz (CS) score, respectively. The baseline score (for $\mathcal{G}_{00}$) is: -336.5 for the LL score, and -360 for the CS score. We can see that the contribution of adding the arc $C \rightarrow X_3$ is radically different when X_4 is added as a child of C. This example shows that both the log-likelihood and the Cheeseman-Stutz score are not decomposable.

Example 19.14

Consider a task of learning a network structure for clustering, where we are also trying to determine whether different features are relevant. More precisely, assume we have a hidden variable C, and four possibly related variables $X_1, \ldots, X_4$. Assume that we have already decided that both X_1 and X_2 are children of C, and are trying to decide which (if any) of the edges $C \rightarrow X_3$ and $C \rightarrow X_4$ to include in the model, thereby giving rise to the four possible models in figure 19.10a. Our training set is as shown in figure 19.10b, and the resulting delta-scores relative to the baseline network $\mathcal{G}_{00}$ are shown in (c). As we can see, adding the edge $C \rightarrow X_3$ to the original structure $\mathcal{G}_{00}$ leads only to a small improvement in the likelihood, and a slight decrease in the Cheeseman-Stutz score. However, adding the edge $C \rightarrow X_3$ to the structure $\mathcal{G}_{01}$, where we also have $C \rightarrow X_4$, leads to a substantial improvement.

This example demonstrates a situation where the score is not decomposable. The intuition here is simple. In the structure $\mathcal{G}_{00}$, the hidden variable C is "tuned" to capture the dependency between X_1 and X_2. In this network structure, there is a weak dependency between these two variables and X_3. In $\mathcal{G}_{10}$, the hidden variable has more or less the same role, and therefore there is little explanatory benefit for X_3 in adding the edge to the hidden variable. However, when we add X_3 and X_4 together, the hidden variable shifts to capture the strong dependency between X_3 and X_4 while still capturing some of the dependencies between X_1 and X_2. Thus, the score improves

dramatically, and in a nonadditive manner. ■

As a consequence of these problems, a search procedure that uses one of the scores we discussed has to evaluate the score of each candidate structure it considers, and it cannot rely on cached computations. In all of the scores we considered, this evaluation involves nontrivial computations (for example, running EM or an MCMC procedure) that are much more expensive than the cost of scoring a structure in the case of complete data. The actual cost of computation in these steps depends on the network structure (that is, the cost of inference in the network) and the number of iterations to convergence. Even in simple networks (for example, ones with a single hidden variable) this computation is an order of magnitude longer than evaluation of the score in complete data.

The main problem is that, in this type of search, most of the computation results are discarded. To understand why, recall that to select a move o from our current graph $\mathcal{G}$, we first evaluate all candidate successors $o(\mathcal{G})$. To evaluate each candidate structure $o(\mathcal{G})$, we compute the MLE or MAP parameters for $o(\mathcal{G})$, score it, and then compare it to the score of other candidates we consider at this point. Since we select to apply only one of the proposed search operators o at each iteration, the parameters learned for other structures $o'(\mathcal{G})$ are not needed. In practice, search using the modify-score-discard strategy is rarely feasible; it is useful only for learning in small domains, or when we have many constraints on the network structure, and so do not have many choices at each decision point.

19.4.2.2 Heuristic Solutions

There are several heuristics for avoiding this significant computational cost. We list a few of them here. We note that nondecomposability is a major issue in the context of Markov network learning, and so we return to these ideas in much greater length in section 20.7.

One approach is to construct "quick and dirty" estimates of the change in score. We can employ such estimates in a variety of ways. In one approach, we can simply use them as our estimates of the delta-score in any of the search algorithms used earlier. Alternatively, we can use them as a pruning mechanism, focusing our attention on the moves whose estimated change in score is highest and evaluating them more carefully. This approach uses the estimates to prioritize our computational resources and invest computation on careful evaluation of the real change in score for fewer modifications.

There are a variety of different approaches we can use to estimate the change in score. One approach is to use computations of delta-scores acquired in previous steps. More precisely, suppose we are at a structure $\mathcal{G}_0$, and evaluate a search operator whose delta-score is $\delta(\mathcal{G}_0 : o)$. We can then assume for the next rounds of search that the delta-score for this operator has not changed, even though the network structure has changed. This approach allows us to cache the results computation for at least some number of subsequent iterations (as though we are learning from complete data). In effect, this approach approximates the score as decomposable, at least for the duration of a few iterations. Clearly, this approach is only an approximation, but one that may be quite reasonable: even if the delta-scores themselves change, it is not unreasonable to assume that a step that was good relative to one structure is often probably good also relative to a closely related structure. Of course, this assumption can also break down: as the score is not decomposable, applying a set of "beneficial" search operators together can lead to structures with worse score.

The implementation of such a scheme requires us to make various decisions. How long do we maintain the estimate $\delta(\mathcal{G} : o)$? Which other search operators invalidate this estimate after they are applied? There is no clear right answer here, and the actual details of implementations of this heuristic approach differ on these counts.

Another approach is to compute the score of the modified network, but assume that only the parameters of the changed CPD can be optimized. That is, we freeze all of the parameters except those of the single CPD $P(X \mid \boldsymbol{U})$ whose family composition has changed, and optimize the parameters of $P(X \mid \boldsymbol{U})$ using gradient ascent or EM. When we optimize only the parameters of a single CPD, EM or gradient ascent should be faster for two reasons. First, because we have only a few parameters to learn, the convergence is faster. Second, because we modify only the parameters of a single CPD, we can cache intermediate computations; see exercise 19.15.

The set of parameterizations where only the CPD $P(X \mid \boldsymbol{U})$ is allowed to change is a subset of the set of all possible parameterizations for our network. Hence, any likelihood that we can achieve in this case would also be achievable if we ran a full optimization. As a consequence, the estimate of the likelihood in the modified network is a lower bound of the actual likelihood we can achieve if we can optimize all the parameters. If we are using a score such as the BIC score, this estimate is a proven lower bound on the score. If we are using a score such as the Cheeseman-Stutz score, this argument is not valid, but appears generally to hold in practice. That is, the score of the network with frozen parameters will be usually either smaller or very close to that of the one were we can optimize all parameters.

More generally, if a heuristic estimate is a proven lower bound or upper bound, we can improve the search procedure in a way that is guaranteed not to lose the optimal candidates. In the case of a lower bound, an estimated value that is higher than moves that we have already evaluated allows us to prune those other moves as guaranteed to be suboptimal. Conversely, if we have a move with a guaranteed upper bound that is lower than previously evaluated candidates, we can safely eliminate it. In practice, however, such bounds are hard to come by.

19.4.3 Structural EM

We now consider a different approach to constructing a heuristic that identifies helpful moves during the search. This approach shares some of the ideas that we discussed. However, by putting them together in a particular way, it provides significant computational savings, as well as certain guarantees.

19.4.3.1 Approximating the Delta-score

One efficient approach for approximating the score of network is to construct some complete data set $\mathcal{D}^*$, and then apply a score based on the complete data set. This was precisely the intuition that motivated the Cheeseman-Stutz approximation. However, the Cheeseman-Stutz approximation is computationally expensive. The data set we use — $\mathcal{D}^*_{\mathcal{G},\tilde{\boldsymbol{\theta}}_{\mathcal{G}}}$ — is constructed by finding the MAP parameters for our current candidate $\mathcal{G}$. We also needed to introduce a correction term that would improve the approximation to the marginal likelihood; this correction term required that we run inference over $\mathcal{G}$. Because these steps must be executed for each candidate network, this approach quickly becomes infeasible for large search spaces.

However, what if we do not want to obtain an accurate approximation to the marginal

likelihood? Rather, we want only a heuristic that would help identify useful moves in the space.

completed data

In this case, one simple heuristic is to construct a single *completed data set* $\mathcal{D}^*$ and use it to evaluate multiple different search steps. That is, to evaluate a search operator o, we define

$$\hat{\delta}_{\mathcal{D}^*}(\mathcal{G} : o) = \text{score}(o(\mathcal{G}) : \mathcal{D}^*) - \text{score}(\mathcal{G} : \mathcal{D}^*),$$

where we can use any complete-data scoring function for the two terms on the right-hand side. The key observation is that this expression is simply a delta-score relative to a complete data set, and it can therefore be evaluated very efficiently. We will return to this point.

Clearly, the results of applying this heuristic depend on our choice of completed data set $\mathcal{D}^*$, an observation that immediately raises some important questions: How do we define our completed data set $\mathcal{D}^*$? Can we provide any guarantees on the accuracy of this heuristic? One compelling answer to these questions is obtained from the following result:

Theorem 19.10

Let $\mathcal{G}_0$ be a graph structure and $\tilde{\boldsymbol{\theta}}_0$ be the MAP parameters for $\mathcal{G}_0$ given a data set $\mathcal{D}$. Then for any graph structure $\mathcal{G}$:

$$\text{score}_{BIC}(\mathcal{G} : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}) - \text{score}_{BIC}(\mathcal{G}_0 : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}) \leq \text{score}_{BIC}(\mathcal{G} : \mathcal{D}) - \text{score}_{BIC}(\mathcal{G}_0 : \mathcal{D}).$$

This theorem states that the true improvement in the BIC score of network $\mathcal{G}$, relative to the network $\mathcal{G}_0$ that we used to construct our completed data $\mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}$, is at least as large as the estimated improvement of the score using the completed data $\mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}$.

The proof of this theorem is essentially the same as the proof of theorem 19.5; see exercise 19.25. Although the analogous result for the Bayesian score is not true (due to the nonlinearity of the Γ function used in the score), it is approximately true, especially when we have a reasonably large sample size; thus, we often apply the same ideas in the context of the Bayesian score, albeit without the same level of theoretical guarantees.

This result suggests the following scheme. Consider a graph structure $\mathcal{G}_0$. We compute its MAP parameters $\tilde{\boldsymbol{\theta}}_0$, and construct a complete (fractional) data set $\mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}$. We can now use the BIC score relative to this completed data set to evaluate the delta-score for any modification o to $\mathcal{G}$. We can thus define

$$\hat{\delta}_{\mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}}(\mathcal{G} : o) = \text{score}_{BIC}(o(\mathcal{G}) : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}) - \text{score}_{BIC}(\mathcal{G} : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}).$$

The theorem guarantees that our heuristic estimate for the delta-score is a lower bound on the true change in the BIC score. The fact that this estimate is a lower bound is significant, since it guarantees that any change that we make that improves the estimated score will also improve the true score.

19.4.3.2 The Structural EM Algorithm

Importantly, the preceding guarantee holds not just for the application of a single operator, but also for any series of changes that modify $\mathcal{G}_0$. Thus, we can use our completed data set $\mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}$ to estimate and apply an arbitrarily long sequence of operators to $\mathcal{G}_0$; as long as we have that

$$\text{score}_{BIC}(\mathcal{G} : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0}) > \text{score}_{BIC}(\mathcal{G}_0 : \mathcal{D}^*_{\mathcal{G}_0,\tilde{\boldsymbol{\theta}}_0})$$

for our new graph $\mathcal{G}$, we are guaranteed that the true score of $\mathcal{G}$ is also better.

Algorithm 19.3 The structural EM algorithm for structure learning

Procedure Structural-EM (
  $\mathcal{G}^0$, // Initial bayesian network structure over $X_1, \ldots, X_n$
  $\boldsymbol{\theta}^0$, // Initial set of parameters for $\mathcal{G}^0$
  $\mathcal{D}$ // Partially observed data set
)
1 **for** each $t = 0, 1 \ldots,$ until convergence
2 // Optional parameter learning step
3 $\boldsymbol{\theta}^{t'} \leftarrow$ Expectation-Maximization$(\mathcal{G}^t, \boldsymbol{\theta}^t, \mathcal{D})$
4 // Run EM to generate expected sufficient statistics for $\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}}$
5 $\mathcal{G}^{t+1} \leftarrow$ Structure-Learn$(\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}})$
6 $\boldsymbol{\theta}^{t+1} \leftarrow$ Estimate-Parameters$(\mathcal{D}^*_{\mathcal{G}^t, \boldsymbol{\theta}^{t'}}, \mathcal{G}^{t+1})$
7 **return** $\mathcal{G}^t, \boldsymbol{\theta}^t$

However, we must take care in interpreting this guarantee. Assume that we have already modified $\mathcal{G}_0$ in several ways, to obtain a new graph $\mathcal{G}$. Now, we are considering a new operator o, and are interested in determining whether that operator is an improvement; that is, we wish to estimate the delta-score: $\text{score}_{BIC}(o(\mathcal{G}) : \mathcal{D}) - \text{score}_{BIC}(\mathcal{G} : \mathcal{D})$. The theorem tells us that if $o(\mathcal{G})$ satisfies $\text{score}_{BIC}(o(\mathcal{G}) : \mathcal{D}^*_{\mathcal{G}_0, \tilde{\boldsymbol{\theta}}_0}) > \text{score}_{BIC}(\mathcal{G}_0 : \mathcal{D}^*_{\mathcal{G}_0, \tilde{\boldsymbol{\theta}}_0})$, then it is necessarily better than our original graph $\mathcal{G}_0$. However, it does not follow that if $\hat{\delta}_{\mathcal{D}^*_{\mathcal{G}_0, \tilde{\boldsymbol{\theta}}_0}}(\mathcal{G} : o) > 0$, then $o(\mathcal{G})$ is necessarily better than $\mathcal{G}$. In other words, we can verify that each of the graphs we construct improves over the graph used to construct the completed data set, but not that each operator improves over the previous graph in the sequence. Note that we are guaranteed that our estimate is a true lower bound for any operator applied directly to $\mathcal{G}_0$. Intuitively, we believe that our estimates are likely to be reasonable for graphs that are "similar" to $\mathcal{G}_0$. (This intuition was also the basis for some of the heuristics described in section 19.4.2.2.) However, as we move farther away, our estimates are likely to degrade. Thus, at some point during our search, we probably want to select a new graph and construct a more relevant complete data set.

structural EM

This observation suggests an EM-like algorithm, called *structural EM*, shown in algorithm 19.3. In structural EM, we iterate over a pair of steps. In the E-step, we use our current model to generate (perhaps implicitly) a completed data set, based on which we compute expected sufficient statistics. In the M-step, we use these expected sufficient statistics to improve our model. The biggest difference is that now our M-step can improve not only the parameters, but also the structure. (Note that the structure-learning step also reestimates the parameters.) The structure learning procedure in the M-step can be any of the procedures we discussed in section 18.4, whether a general-purpose heuristic search or an exact search procedure for a specialized subset of networks for which we have an exact solution (for example, a maximum weighted spanning tree procedure for learning trees). If we use the BIC score, theorem 19.10 guarantees that, if this search procedure finds a structure that is better than the one we used in the previous iteration, then the structural EM procedure will monotonically improve the score.

Since the scores are upper-bounded, the algorithm must converge. Unlike the case of EM, we cannot, however, prove that the structure it finds is a local maximum.

19.4.3.3 Structural EM for Selective Clustering

We now illustrate the structural EM algorithm on a particular class of networks. Consider the task of learning structures that generalize our example of example 19.14; these networks are similar to the naive Bayes clustering of section 19.2.2.4, except that some observed variables may be independent of the cluster variable. Thus, in our structure, the class variable C is a root, and each observed attribute X_i is either a child of C or a root by itself. This limited set of structures contains 2^n choices.

Before discussing how to learn these structures using the ideas we just explored, let us consider why this problem is an interesting one. One might claim that, instead of structure learning, we can simply run parameter learning within the full structure (where each X_i is a child of C); after all, if X_i is independent of C, then we can capture this independence within the parameters of the CPD $P(X_i \mid C)$. However, as we discussed, statistical noise in the sampling process guarantees that we will never have true independence in the empirical distribution. Learning a more restricted model with fewer edges is likely to result in more robust clustering. Moreover, this approach allows us to detect irrelevant attributes during the clustering, providing insight into the domain.

If we have a complete data set, learning in this class of models is trivial. Since this class of structures is such that we cannot have cycles, if the score is decomposable, the choice of family for X_i does not impact the choice of parents for X_j. Thus, we can simply select the optimal family for each X_i separately: either C is its only parent, or it has no parents. We can thus select the optimal structure using $2n$ local score evaluations.

The structural EM algorithm applies very well in this setting. We initialize each iteration with our current structure $\mathcal{G}_t$. We then perform the following steps:

- Run parameter estimation (such as EM or gradient ascent) to learn parameters $\tilde{\boldsymbol{\theta}}_t$ for $\mathcal{G}_t$.
- Construct a new structure $\mathcal{G}_{t+1}$ so that $\mathcal{G}_{t+1}$ contains the edge $C \rightarrow X_i$ if

$$\text{FamScore}(X_i \mid \{C\} \ : \ \mathcal{D}^*_{\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t}) > \text{FamScore}(X_i \mid \emptyset \ : \ \mathcal{D}^*_{\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t}).$$

We continue this procedure until convergence, that is, until an iteration that makes no changes to the structure.

According to theorem 19.10, if we use the BIC score in this procedure, then any improvement to our expected score based on $\mathcal{D}^*_{\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t}$ is guaranteed to give rise to an improvement in the true BIC score; that is, $\text{score}_{BIC}(\mathcal{G}_{t+1} \ : \ \mathcal{D}) \geq \text{score}_{BIC}(\mathcal{G}_t \ : \ \mathcal{D})$. Thus, each iteration (until convergence) improves the score of the model.

One issue in implementing this procedure is how to evaluate the family scores in each iteration: $\text{FamScore}(X_i \mid \emptyset \ : \ \mathcal{D}^*_{\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t})$ and $\text{FamScore}(X_i \mid \{C\} \ : \ \mathcal{D}^*_{\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t})$. The first term depends on sufficient statistics for X_i in the data set; as X_i is fully observed, these can be collected once and reused in each iteration. The second term requires sufficient statistics of X_i

and C in $\mathcal{D}^*_{\mathcal{G}_t,\tilde{\boldsymbol{\theta}}_t}$; here, we need to compute:

$$\begin{aligned}\bar{M}_{\mathcal{D}^*_{\mathcal{G}_t,\tilde{\boldsymbol{\theta}}_t}}[x_i, c] &= \sum_m P(C[m] = c, X_i[m] = x_i \mid \boldsymbol{o}[m], \mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t) \\ &= \sum_{m, X_i[m]=x_i} P(C[m] = c \mid \boldsymbol{o}[m], \mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t).\end{aligned}$$

We can collect all of these statistics with a singe pass over the data, where we compute the posterior over C in each instance. Note that these are the statistics we need for parameter learning in the full naive Bayes network, where each X_i is connected to C. In some of the iterations of the algorithm, we will compute these statistics even though X_i and C are independent in $\mathcal{G}_t$. Somewhat surprisingly, even when the joint counts of X_i and C are obtained from a model where these two variables are independent, the expected counts can show a dependency between them; see exercise 19.26.

Note that this algorithm can take very large steps in the space. Specifically, the choice of edges in each iteration is made from scratch, independently of the choice in the previous structure; thus, $\mathcal{G}_{t+1}$ can be quite different from $\mathcal{G}_t$. Of course, this observation is true only up to a point, since the use of the distribution based on $(\mathcal{G}_t, \tilde{\boldsymbol{\theta}}_t)$ does bias the reconstruction to favor some aspects of the previous iteration. This point goes back to the inherent nondecomposability of the score in this case, which we saw in example 19.14. To understand the limitation, consider the convergence point of EM for a particular graph structure where C has a particular set of children $\boldsymbol{X}$. At this point, the learned model is optimized so that C captures (as much as possible) the dependencies between its children in $\boldsymbol{X}$, to allow the variables in $\boldsymbol{X}$ to be conditionally independent given C. Thus, different choices of $\boldsymbol{X}$ will give rise to very different models. When we change the set of children, we change the information that C represents, and thus change the score in a global way. As a consequence, the choice of $\mathcal{G}_t$ that we used to construct the completed data does affect our ability to add certain edges into the graph.

This issue brings up the important question of how we can initialize this search procedure. A simple initialization point is to use the full network, which is essentially the naive Bayes clustering network, and let the search procedure prune edges. An alternative is to start with a random subset of edges. Such a randomized starting point can allow us to discover "local maxima" that are not accessible from the full network. One might also tempted to use the empty network as a starting point, and then consider adding edges. It is not hard to show, however, that the empty network is a bad starting point: structural EM will never add a new edge if we initialize the algorithm with the empty network; see exercise 19.27.

19.4.3.4 An Effective Implementation of Structural EM

Our description of the structural EM procedure is at a somewhat abstract level, and it lends itself to different types of implementations. The big unresolved issue in this description is how to represent and manage the completed data set created in the E-step. Recall that the number of completions of each instance is exponential in the number of missing values in that instance. If we have a single hidden variable, as in the selective naive Bayes classifier of section 19.4.3.3, then storing all completions (and their relative weights) might be a feasible implementation. However, if we have several unobserved variables in each instance, then this solution rapidly becomes impractical.

We can, however, exploit the fact that procedures that learn from complete data sets do not need to access all the instances; they require only sufficient statistics computed from the data set. Thus, we do not need to maintain all the instances of the completed data set; we need only to compute the relevant sufficient statistics in the completed data set. These sufficient statistics are, by definition, the expected sufficient statistics based on the current model $(\mathcal{G}_t, \boldsymbol{\theta}_t)$ and the observed data. This is precisely the same idea that we utilized in the E-step of standard EM for parameter estimation. However, there is one big difference. In parameter estimation, we know in advance the sufficient statistics we need. When we perform structure learning, this is no longer true. When we change the structure, we need a new set of sufficient statistics for the parts of the model we have changed. For example, if in the original network X is a root, then, for parameter estimation, we need only sufficient statistics of X alone. Now, if we consider adding Y as a parent of X, we need the joint statistics of X and Y together. If we do add the edge $Y \rightarrow X$, and now consider Z as an additional parent of X, we now need the joint statistics of X, Y, and Z.

This suggests that the number of sufficient statistics we may need can be quite large. One strategy is to compute in advance the set of sufficient statistics we might need. For specialized classes of structures, we may know this set exactly. For example, in the clustering scenario that we examined in section 19.4.3.3, we know the precise sufficient statistics that are needed for the M-step. Similarly, if we restrict ourselves to trees, we know that we are interested only in pairwise statistics and can collect all of them in advance. If we are willing to assume that our network has a bounded indegree of at most k, then we can also decide to precompute all sufficient statistics involving $k + 1$ or fewer variables; this approach, however, can be expensive for k greater than two or three. An alternative strategy is to compute sufficient statistics "on demand" as the search progresses through the space of different structures. This approach allows us to compute only the sufficient statistics that the search procedure requires. However, it requires that we revisit the data and perform new inference queries on the instances; moreover, this inference generally involves variables that are not together in a family and therefore may require out-of-clique inference, such as the one described in section 10.3.3.2.

Importantly, however, once we compute sufficient statistics, all of the decomposability properties for complete data that we discussed in section 18.4.3.3 hold for the resulting delta-scores. Thus, we can apply our caching-based optimizations in this setting, greatly increasing the computational efficiency of the algorithm. This property is key to allowing the structural EM algorithm to scale up to large domains with many variables.

19.5 Learning Models with Hidden Variables

In the previous section we examined searching for structures when the data are incomplete. In that discussion, we confined ourselves to structures involving a given set of variables. Although this set can include hidden variables, we implicitly assumed that we knew of the existence of these variables, and could simply treat them as an extreme case of missing data. Of course, it is important to remember that hidden variables introduce important subtleties, such as our inability to identify the model. Nevertheless, as we discussed, in section 16.4.2, hidden variables are useful for a variety of reasons.

In some cases, prior knowledge may tell us that a hidden variable belongs in the model, and

perhaps even where we should place it relative to the other variables. In other cases (such as the naive Bayes clustering), the placement of the hidden variable is dictated by the goals of our learning (clustering of the instances into coherent groups). In still other cases, however, we may want to infer automatically that it would be beneficial to introduce a hidden variable into the model. This opportunity raises a whole range of new questions: When should we consider introducing a hidden variable? Where in the network should we connect it? How many values should we allow it to have?

In this section, we first present some results that provide intuition regarding the role that a hidden variable can play in the model. We then describe a few heuristics for dealing with some of the computational questions described before.

19.5.1 Information Content of Hidden Variables

One can view the role of a hidden variable as a mechanism for capturing information about the interaction between other variables in the network. In our example of the network of figure 16.1, we saw that the hidden variable "conveyed" information from the parents X_1, X_2, X_3 to the children Y_1, Y_2, Y_3. Similarly, in the naive Bayes clustering network of figure 3.2, the hidden variable captures information between its children. These examples suggest that, in learning a model for the hidden variable, we want to maximize the *information* that the hidden variable captures about its children. We now show that learning indeed maximizes a notion of information between the hidden variable and its children. We analyze a specific example, the naive Bayes network of figure 3.2, but the ideas can be generalized to other network structures.

information content

Suppose we observe M samples of $X_1, \ldots, X_n$ and use maximum likelihood to learn the parameters of the network. Any choice of parameter set $\boldsymbol{\theta}$ defines a distribution $\hat{Q}_{\boldsymbol{\theta}}$ over $X_1, \ldots, X_n, H$ so that

$$\hat{Q}_{\boldsymbol{\theta}}(h, x_1, \ldots, x_n) = \hat{P}_{\mathcal{D}}(x_1, \ldots, x_n) P(h \mid x_1, \ldots, x_n, \boldsymbol{\theta}), \tag{19.14}$$

where $\hat{P}_{\mathcal{D}}$ is the empirical distribution of the observed variables in the data. This is essentially the augmentation of the empirical distribution by our stochastic "reconstruction" of the hidden variable.

Consider for a moment a complete data set $\langle \mathcal{D}, \mathcal{H} \rangle$, where H is also observed. Proposition 18.1 shows that

$$\max_{\boldsymbol{\theta}} \frac{1}{M} \ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle) = \sum_i \boldsymbol{I}_{\hat{P}_{\langle \mathcal{D}, \mathcal{H} \rangle}}(X_i; H) - \boldsymbol{H}_{\hat{P}_{\langle \mathcal{D}, \mathcal{H} \rangle}}(H) - \sum_i \boldsymbol{H}_{\hat{P}_{\langle \mathcal{D}, \mathcal{H} \rangle}}(X_i). \tag{19.15}$$

We now show that a similar relationship holds in the case of incomplete data; in fact, this relationship holds not only at the maximum likelihood point but also in other points of the parameter space:

Proposition 19.2 *Let $\mathcal{D}$ be a data set where $X_1, \ldots, X_n$ are observed and $\boldsymbol{\theta}^0$ be a choice of parameters for the network of figure 3.2. Define $\boldsymbol{\theta}^1$ to be the result of an EM-iteration if we start with $\boldsymbol{\theta}^0$ (that is, the result of an M-step if we use sufficient statistics from $\hat{Q}_{\boldsymbol{\theta}^0}$). Then*

$$\frac{1}{M} \ell(\boldsymbol{\theta}^0 : \mathcal{D}) \leq \sum_i \boldsymbol{I}_{\hat{Q}_{\boldsymbol{\theta}^0}}(X_i; H) - \sum_i \boldsymbol{H}_{\hat{P}_{\mathcal{D}}}(X_i) \leq \frac{1}{M} \ell(\boldsymbol{\theta}^1 : \mathcal{D}). \tag{19.16}$$

Roughly speaking, this result states that the information-theoretic term is approximately equal to the likelihood. When $\boldsymbol{\theta}^0$ is a local maxima of the likelihood, we have that $\boldsymbol{\theta}^1 = \boldsymbol{\theta}^0$, and so we have equality in the left-hand and right-hand sides of equation (19.16). For other parameter choices, the information-theoretic term can be larger than the likelihood, but not by "too much," since it is bounded above by the next iteration of EM. Both the likelihood and the information-theoretic term have the same maxima.

Because the entropy terms $\boldsymbol{H}_{\hat{P}_{\mathcal{D}}}(X_i)$ do not depend on $\boldsymbol{\theta}$, this result implies that maximizing the likelihood is equivalent to finding a hidden variable H that maximizes the information about each of the observed variables. Note that the information here is defined in terms of the distribution $\hat{Q}_{\boldsymbol{\theta}^0}$, as in equation (19.14). This information measures what H conveys about each of the observed variables in the *posterior distribution* given the observations. This is quite intuitive: For example, assume we learn a model, and after observing $x_1, \ldots, x_n$, our posterior over H has $H = h^1$ with high probability. In this case, we are fairly sure about the cluster assignment of the cluster, so if the clustering is informative, we can conclude quite a bit of information about the value of each of the attributes.

Finally, it is useful to compare this result to the complete data case of equation (19.15); there, we had an additional $-\boldsymbol{H}(H)$ term, which accounts for the observations of H. In the case of incomplete data, we do not observe H and thus do not need to account for it. Intuitively, since we sum over all the possible values of H, we are not penalized for more complex (higher entropy) hidden variables. This difference also shows that adding more values to the hidden variable will always improve the likelihood. As we add more values, the hidden variable can only become more informative about the observed variables. Since our likelihood function does not include a penalty term for the entropy of H, this score does not penalize for the increased number of values of H.

We now turn to the proof of this proposition.

Proof Define $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^0)$, then, by corollary 19.1 we have that

$$\ell(\boldsymbol{\theta}^0 : \mathcal{D}) = \boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^0 : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_Q(\mathcal{H}).$$

Moreover, if $\boldsymbol{\theta}^1 = \arg\max_{\boldsymbol{\theta}} \boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H}\rangle)]$, then

$$\boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^0 : \langle \mathcal{D}, \mathcal{H}\rangle)\right] \leq \boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^1 : \langle \mathcal{D}, \mathcal{H}\rangle)\right].$$

Finally, we can use corollary 19.1 again and get that

$$\boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^1 : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_Q(\mathcal{H}) \leq \ell(\boldsymbol{\theta}^1 : \mathcal{D}).$$

Combining these three inequalities, we conclude that

$$\ell(\boldsymbol{\theta}^0 : \mathcal{D}) \leq \boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^1 : \langle \mathcal{D}, \mathcal{H}\rangle)\right] + \boldsymbol{H}_Q(\mathcal{H}) \leq \ell(\boldsymbol{\theta}^1 : \mathcal{D}).$$

Since $\boldsymbol{\theta}^1$ maximize the expected log-likelihood, we can apply equation (19.15) for the completed data set $\langle \mathcal{D}, \mathcal{H}\rangle$, and conclude that

$$\boldsymbol{E}_Q\left[\ell(\boldsymbol{\theta}^1 : \langle \mathcal{D}, \mathcal{H}\rangle)\right] = M\left[\sum_i \boldsymbol{E}_Q\left[\boldsymbol{I}_{\hat{P}_{\langle \mathcal{D}, \mathcal{H}\rangle}}(X_i; H)\right] - \boldsymbol{E}_Q\left[\boldsymbol{H}_{\hat{P}_{\langle \mathcal{D}, \mathcal{H}\rangle}}(H)\right] - \sum_i \boldsymbol{H}_{\hat{P}_{\mathcal{D}}}(X_i)\right].$$

Using basic rewriting, we have that

$$M\mathbf{E}_Q\left[\mathbf{H}_{\hat{P}_{\langle \mathcal{D},\mathcal{H}\rangle}}(H)\right] = \mathbf{H}_Q(\mathcal{H})$$

and that

$$\mathbf{E}_Q\left[\mathbf{I}_{\hat{P}_{\langle \mathcal{D},\mathcal{H}\rangle}}(X_i;H)\right] = \mathbf{I}_{\hat{Q}_{\theta^0}}(X_i;H),$$

which proves the result. ■

19.5.2 Determining the Cardinality

One of the key questions that we need to address for a hidden variable is that of its cardinality.

19.5.2.1 Model Selection for Cardinality

model selection

The simplest approach is to use *model selection*, where we consider a number of different cardinalities for H, and then select the best one. For our evaluation criterion, we can use a Bayesian technique, utilizing one of the approximate scores presented in section 19.4.1; box 19.G provides a comparative study of the different scores in precisely this setting. As another alternative, we can measure test generalization performance on a holdout set or using cross-validation. Both of these methods are quite expensive, even for a single hidden variable, since they both require that we learn a full model for each of the different cardinalities that we are considering; for multiple hidden variables, it is generally intractable.

A cheaper approach is to consider a more focused problem of using H to represent a clustering problem, where we cluster instances based only on the features $\boldsymbol{X}$ in H's (putative) Markov blanket. Here, the assumption is that if we give H enough expressive power to capture the distinctions between different classes of instances, we have captured much of the information in $\boldsymbol{X}$. We can now use any clustering algorithm to construct different clusterings and to evaluate their explanatory power. Commonly used variants are EM with a naive Bayes model, the simpler k-means algorithm, or any other of many existing clustering algorithms. We can now evaluate different cardinalities for H at much lower cost, using a score that measures only the quality of the local clustering. An even simpler approach is to introduce H with a low cardinality (say binary-valued), and then use subsequent learning stages to tell us whether there is still information in the vicinity of H. If there is, we can either increase the cardinality of H, or add another hidden variable.

19.5.2.2 Dirichlet Processes

Bayesian model averaging

A very different alternative is a *Bayesian model averaging* approach, where we do not select a cardinality, but rather average over different possible cardinalities. Here, we use a prior over the set of possible cardinalities of the hidden variable, and use the data to define a posterior. **The Bayesian model averaging approach allows us to circumvent the difficult question of selecting the cardinality of the hidden variable. On the other side, because it fails to make a definitive decision on the set of clusters and on the assignment of instances to clusters, the results of the algorithm may be harder to interpret. Moreover, techniques**

that use Bayesian model averaging are generally computationally even more expensive than approaches that use model selection.

Dirichlet process

One particularly elegant solution is provided by the *Dirichlet process* approach. We provide an intuitive, bottom-up derivation for this approach, which also has extensive mathematical foundations; see section 19.7 for some references.

To understand the basic idea, consider what happens if we apply the approach of example 19.12 but allow the number of possible clusters K to grow very large, much larger than the number of data instances. In this case, the bound K does not limit the expressive power of model, since we can (in principle) put each instance in its own cluster.

Our natural concern about this solution is the possibility of overfitting: after all, we certainly do not want to put each point in its own cluster. However, recall that we are using a Bayesian approach and not maximum likelihood. To understand the difference, consider the posterior distribution over the $(M+1)$'st instance given a cluster assignment for the previous instances $1, \ldots, M$. This formula is also proportional to equation (19.10), with $M+1$ playing the role of m'. (The normalizing constant is different, because here we are also interested in modeling the distribution over $\boldsymbol{X}[M+1]$, whereas there we took $\boldsymbol{x}[m']$ as given.) The first term in the equation, $|I_k(\boldsymbol{c})| + \alpha_0/K$, captures the relative prior probability that the $(M+1)$'st instance selects to join cluster k. Note that the more instances are in the k'th cluster, the higher the probability that the new instance will select to join. Thus, the Dirichlet prior naturally causes instances to prefer to cluster together and thereby helps avoid overfitting.

A second concern is the computational burden of maintaining a very large number of clusters. Recall that if we use the collapsed Gibbs sampling approach of example 19.12, the cost per sampling step grows linearly with K. Moreover, most of this computation seems like a waste: with such a large K, many of the clusters are likely to remain empty, so why should we waste our time considering them?

The solution is to abstract the notion of a cluster assignment. Because clusters are completely symmetrical, we do not care about the specific assignment to the variables $C[m]$, but only about the *partition* of the instances into groups. Moreover, we can collapse all of the empty partitions, treating them all as equivalent. We therefore define a particle σ in our collapsed Gibbs process to encode a *partition* of the the data instances $\{1, \ldots, M\}$: an unordered set of non-empty subsets $\{I_1, \ldots, I_l\}$. Each $I \in \sigma$ is associated with a distribution over the parameters $\Theta_\sigma = \{\boldsymbol{\theta}_I\}_{I \in \sigma}$ and over the multinomial $\boldsymbol{\theta}$. As usual, we define $\sigma_{-m'}$ to denote the partition induced when we remove the instance m'.

partition

To define the sampling process, let $C[m']$ be the variable to be sampled. Let L be the number of (non-empty) clusters in the partition $\sigma_{-m'}$. Introducing $C[m']$ (while keeping the other instances fixed) induces $L+1$ possible partitions: joining one of the L existing clusters, or opening a new one. We can compute the conditional probabilities of each of these outcomes. Let $I \in \sigma$.

$$P(I \leftarrow I \cup \{m'\} \mid \sigma_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto \left(|I| + \frac{\alpha_0}{K}\right) Q(\boldsymbol{x}[m'] \mid \mathcal{D}_I, \boldsymbol{\omega}) \tag{19.17}$$

$$P(\sigma \leftarrow \sigma \cup \{\{m'\}\} \mid \sigma_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto (K - L)\frac{\alpha_0}{K} Q(\boldsymbol{x}[m'] \mid \boldsymbol{\omega}), \tag{19.18}$$

where the first line denotes the event where m' joins an existing cluster I, and the second the event where it forms a new singleton cluster (containing only m') that is added to σ. To compute these transition probabilities, we needed to sum over all possible concrete cluster assignments

that are consistent with σ, but this computation is greatly facilitated by the symmetry of our prior (see exercise 19.29). Using abstract partitions as our particles provides significant computational savings: we need only to compute $L+1$ values for computing the transition distribution, rather than K, reducing the complexity of each Gibbs iteration to $O(NL)$, independent of the number of classes K.

As long as K is larger than the amount of data, it appears to play no real role in the model. Therefore, a more elegant approach is to remove it, allowing the number of clusters to grow to infinity. At the limit, the sampling equation for σ is now even simpler:

$$P(I \leftarrow I \cup \{m'\} \mid \sigma_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto |I| \cdot Q(\boldsymbol{x}[m'] \mid \mathcal{D}_I, \boldsymbol{\omega}) \tag{19.19}$$

$$P(\sigma \leftarrow \sigma \cup \{\{m'\}\} \mid \sigma_{-m'}, \mathcal{D}, \boldsymbol{\omega}) \propto \alpha_0 \cdot Q(\boldsymbol{x}[m'] \mid \boldsymbol{\omega}). \tag{19.20}$$

This scheme removes the bound on the number of clusters and induces a prior that allows any possible partition of the samples. Given the data, we obtain a posterior over the space of possible partitions. This posterior gives positive probability to partitions with different numbers of clusters, thereby averaging over models with different complexity. In general, the number of clusters tends to grow logarithmically with the size of the data. This type of model is called a *nonparametric Bayesian model*; see also box 17.B.

nonparametric Bayesian estimation

Of course, with K at infinity, our Dirichlet prior over $\boldsymbol{\theta}$ is not a legal prior. Fortunately, it turns out that one can define a generalization of the Dirichlet prior that induces these conditional probabilities. One simple derivation comes from a sampling process known as the *Chinese restaurant process*. This process generates a random partition as follows: The guests (instances) enter a restaurant one by one, and each guest chooses between joining one of the non-empty tables (clusters) and opening a new table. The probability that a customer chooses to join a table l at which n_l customers are already sitting is $\propto n_l$; the probability that he opens a new tables is $\propto \alpha_0$. The instances assigned to the same table all use the parameters of that table. It is not hard to show (exercise 19.30) that this prior induces precisely the update equations in equation (19.19) and equation (19.20).

Chinese restaurant process

A second derivation is called the *stick-breaking prior*; it is parameterized by α_0, and defines an infinite sequence of random variables $\beta_i \sim \mathit{Beta}(1, \alpha_0)$. We can now define an infinite-dimensional vector defined as:

stick-breaking prior

$$\lambda_k = \beta_k \prod_{l=1}^{k-1} (1 - \beta_l).$$

This prior is called a stick-breaking prior because it can be viewed as defining a process of breaking a stick into pieces: We first break a piece of fraction β_1, then the second piece is a fraction β_2 of the remainder, and so on. It is not difficult to see that $\sum_k \lambda_k = 1$. It is also possible to show that, under the appropriate definitions, the limit of the distributions $\mathit{Dirichlet}(\alpha_0/K, \ldots, \alpha_0/K)$ as $K \longrightarrow \infty$ induces the stick-breaking prior.

19.5.3 Introducing Hidden Variables

Finally, we consider the question of determining when and where to introduce a hidden variable. The analysis of section 19.5.1 tells us that a hidden variable in a naive Bayes clustering network is optimized to capture information about the variables to which it is connected. Intuitively,

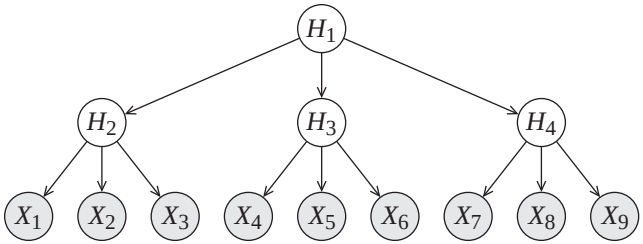


Figure 19.11 An example of a network with a hierarchy of hidden variables

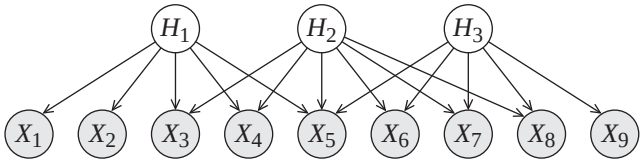


Figure 19.12 An example of a network with overlapping hidden variables

this requirement imposes a significant bias on the parameterization of the hidden variable. This bias helps constrain our search and allows us to learn a hidden variable that plays a meaningful role in the model. Conversely, if we place a hidden variable where the search is not similarly constrained, we run the risk of learning hidden variables that are meaningless, and that only capture the noise in the training data. As a rough rule of thumb, we want the model with the hidden variables to have a lot fewer independent parameters than the number of degrees of freedom of the empirical distribution.

hierarchical organization

Thus, when selecting network topologies involving hidden variables, we must exercise care. One useful example of such a class of topologies are organized *hierarchically* (for example, figure 19.11), where the hidden variables form a treelike hierarchy. Since each hidden variable is a parent of several other variables (either observed or hidden), it serves to mediate the dependencies among its children and between these children and other variables in the network (through its parent). This constraint implies that the hidden variable can improve the likelihood by capturing such dependencies when they exist. The general topology leaves much freedom in determining what is the best hierarchy structure. Intuitively, distance in the hierarchy should roughly correspond to the degree of dependencies between the variables, so that strongly dependent variables would be closer in the hierarchy. This rule is not exact, of course, since the nature of the dependencies influences whether the hidden variable can capture them.

overlapping organization

Another useful class of networks are those with *overlapping* hidden variables; see figure 19.12. In this network, each hidden variable is the parent of several observed variables. The justification is that each hidden variable captures aspects of the instance that several of the observed variables depend on. Such a topology encodes multiple "flavors" of dependencies between the different variables, breaking up the dependency between them as some combination of independent axes. This approach often provides useful information about the structure of the domain. However, once we have an observed variable depending on multiple hidden ones, we might

need to introduce many parameters, or restrict attention to some compact parameterization of the CPDs. Moreover, while the tree structure of hierarchical networks ensures efficient inference, overlapping hidden variables can result in a highly intractable structure.

In both of these approaches, as in others, we need to determine the placements of the hidden variables. As we discussed, once we introduce a hidden variable somewhere within our structure and localize it correctly in the model by connecting it to its correct neighbors, we can estimate parameters for it using EM. Even if we locate it approximately correctly, we can use the structural EM algorithm to adapt both the structure and the parameters.

However, we cannot simply place a hidden variable arbitrarily in the model and expect our learning procedures to learn a reasonable model. Since these methods are based on iterative improvements, running structural EM with a bad initialization usually leads either to a trivial structure (where the hidden variable has few neighbors or disconnected from the rest of the variables) or to a structure that is very similar to the initial network structure. One extreme example of a bad initialization is to introduce a hidden variable that is disconnected from the rest of the variables; here, we can show that the variable will never be connected to the rest of the model (see exercise 19.27).

This discussion raises the important question of how to induce the existence of a hidden variable, and how to assign it a putative position within the network. One approach is based on finding "signatures" that the hidden variable might leave. As we discussed, a hidden variable captures dependencies between the variables to which it is connected. Indeed, if we assume that a hidden variable truly exists in the underlying distribution, we expect its neighbors in the graph to be dependent. For example, in figure 16.1, marginalizing the hidden variable induces correlations among its children and between its parents and its children (see also exercise 3.11). Thus, a useful heuristic is to look for subsets of variables that seem to be highly interdependent.

There are several approaches that one can use to find such subsets. Most obviously, we can learn a structure over the observed variables and then search for subsets that are connected by many edges. An obvious problem with this approach is that most learning methods are biased against learning networks with large indegrees, especially given limited data. Thus, these methods may return sparser structures even when dependencies may exist, preventing us from using the learned structure to infer the existence of a hidden variable. Nevertheless, these methods can be used successfully given a reasonably large number of samples. Another approach is to avoid the structure learning phase and directly consider the dependencies in the empirical distribution. For example, a quick-and-dirty method is to compute a measure of dependency, such as mutual information, between all pairs of variables. This approach avoids the need to examine marginals over larger sets of variables, and hence it is applicable in the case of limited data. However, we note that children of an observed variable will also be highly correlated, so that this approach does not distinguish between dependencies that can be explained by the observed variables and ones that require introducing hidden variables. Nevertheless, we can use this approach as a heuristic for introducing a hidden variable, and potentially employ a subsequent pruning phase to eliminate the variable if it is not helpful, given the observed variables.

19.6 Summary

In this chapter, we considered the problem of learning in the presence of incomplete data. We saw that learning from such data introduces several significant challenges.

One set of challenges involves the statistical interpretation of the learning problem in this setting. As we saw, we need to be aware of the process that generated the missing data and the effect of nonrandom observation mechanisms on the interpretation of the data. Moreover, we also need to be mindful of the possibility of unidentifiability in the models we learn, and as a consequence, to take care when interpreting the results.

A second challenge involves computational considerations. **Most of the key properties that helped make learning feasible in the fully observable case vanish in the partially observed setting. In particular, the likelihood function no longer decomposes, and is even multimodal. As a consequence, the learning task requires global optimization over a high-dimensional space, with an objective that is highly susceptible to local optima.**

We presented two classes of approaches for performing parameter estimation in this setting: a generic gradient-based process, and the EM algorithm, which is specifically designed for maximizing likelihood functions. Both of these methods perform hill climbing over the parameter space, and are therefore guaranteed only to find a local optimum (or rather, a stationary point) of the likelihood function. Moreover, each iteration in these algorithms requires that we solve an inference problem for each (partially observed) instance in our data set, a requirement that introduces a major computational burden.

In some cases, we want not only a single parameter estimate, but also some evaluation of our confidence in those estimates, as would be obtained from Bayesian learning. Clearly, given the challenges we mentioned, closed-form solutions to the integration are generally impossible. However, several useful approximations have been developed and used in practice; most commonly used are the methods based on MCMC methods, and on variational approximations.

We discussed the problem of structure learning in the partially-observed setting. **Most commonly used are the score-based approaches, where we define the problem as one of finding a high-scoring structure. We presented several approximations to the Bayesian score; most of these are based on an asymptotic approximation, and hence should be treated with care given only a small number of samples.** We then discussed the challenges of searching over the space of networks when the score is not decomposable, a setting that (in principle) forces us to apply a highly expensive evaluation procedure to every candidate that we are considering in the search. The structural EM algorithm provides one approach to reduce this cost. It uses an approximation to the score that is based on some completion of the data, allowing us to use the same efficient algorithms that we applied in the complete data case.

Finally, we briefly discussed some of the important questions that arise when we consider hidden variables: Where in the model should we introduce a hidden variable? What should we select as the cardinality of such a variables? And how do we initialize a variable so as to guide the learning algorithm toward "good" regions of the space? While we briefly described some ideas here, the methods are generally heuristic, and there are no guarantees.

Overall, owing to the challenges of this learning setting, the methods we discussed in this chapter are more heuristic and provide weaker guarantees than methods that we encountered in previous learning chapters. For this reason, the application of these methods is more of an art than a science, and there are often variations and alternatives that can be more effective for

particular learning scenarios. This is an active area of study, and even for the simple clustering problem there is still much active research. Thus, we did not attempt to give a complete coverage and rather focused on the core methods and ideas.

☞ **However, while these complications mean that learning from incomplete data is often challenging or even impossible, there are still many real-life applications where the methods we discussed here are highly effective. Indeed, the methods that we described here are some of the most commonly used of any in the book. They simply require that we take care in their application, and generally that we employ a fair amount of hand-tuned engineering.**

19.7 Relevant Literature

The problem of statistical estimation from missing data has received a thorough treatment in the field of statistics. The distinction between the data generating mechanism and the observation mechanism was introduced by Rubin (1976) and Little (1976). Follow-on work defined the notion of MAR and MCAR Little and Rubin (1987). Similarly, the question of identifiability is also a central in statistical inference Casella and Berger (1990); Tanner (1993). Treatment of the subject for Bayesian networks appears in Settimi and Smith (1998a); Garcia (2004).

An early discussion that touches on the gradient of likelihood appears in Buntine (1994). Binder et al. (1997); Thiesson (1995) applied gradient methods for learning with missing values in Bayesian networks. They derived the gradient form and suggested how to compute it efficiently using clique tree calibration. Gradient methods are often more flexible in using models that do not have a closed-form MLE estimate even from complete data (see also chapter 20) or when using alternative objectives. For example, Greiner and Zhou (2002) suggest training using gradient ascent for optimizing conditional likelihood.

The framework of expectation maximization was introduced by Dempster et al. (1977), who generalized ideas that were developed independently in several related fields (for example, the Baum-Welch algorithm in hidden Markov models (Rabiner and Juang 1986)). The use of expectation maximization for maximizing the posterior was introduced by Green (1990). There is a wide literature of extensions of expectation maximization, analysis of convergence rates, and speedup methods; see McLachlan and Krishnan (1997) for a survey. Our presentation of the theoretical foundations of expectation maximization follows the discussion by Neal and Hinton (1998).

The use of expectation maximization in specific graphical models first appeared in various forms (Cheeseman, Self et al. 1988b; Cheeseman, Kelly et al. 1988; Ghahramani and Jordan 1993). Its adaptation for parameter estimation in general graphical models is due to Lauritzen (1995). Several approaches for accelerating EM convergence in graphical models were examined by Bauer et al. (1997) and Ortiz and Kaelbling (1999). The idea of incremental updates within expectation maximization was formulated by Neal and Hinton (1998). The application of expectation maximization for learning the parameters of noisy-or CPDs (or more generally CPDs with causal independence) was suggested by Meek and Heckerman (1997). The relationship between expectation maximization and hard-assignment EM was discussed by Kearns et al. (1997).

There are numerous applications of expectation maximization to a wide variety of problems. The collaborative filtering application of box 19.A is based on Breese et al. (1998). The application to robot mapping of box 19.D is due to Thrun et al. (2004).

There is a rich literature combining expectation maximization with different types of approximate inference procedures. Variational EM was introduced by Ghahramani (1994) and further elaborated by Ghahramani and Jordan (1997). The combination of expectation maximization with various types of belief propagation algorithms has been used in many current applications (see, for example, Frey and Kannan (2000); Heskes et al. (2003); Segal et al. (2001)). Similarly, other combinations have been examined in the literature, such as Monte Carlo EM (Caffo et al. 2005).

Applying Bayesian approaches with incomplete data requires approximate inference. A common solution is to use MCMC sampling, such as Gibbs sampling, using data-completion particles (Gilks et al. 1994). Our discussion of sampling the Dirichlet distribution is based on (Ripley 1987). More advanced sampling is based on the method of Fishman (1976) for sampling from Gamma distributions. Bayesian Variational methods were introduced by MacKay (1997); Jaakkola and Jordan (1997); Bishop et al. (1997) and further elaborated by Attias (1999); Ghahramani and Beal (2000). Minka and Lafferty (2002) suggest a Bayesian method based on expectation propagation.

The development of Laplace-approximation structure scores is based mostly on the presentation in Chickering and Heckerman (1997); this work is also the basis for the analysis of box 19.G. The BIC score was originally suggested by Schwarz (1978). Geiger et al. (1996, 2001) developed the foundations for the BIC score for Bayesian networks with hidden variables. This line of work was extended by several works (D. Rusakov 2005; Settimi and Smith 2000, 1998b). The Cheeseman-Stutz approximation was initially introduced for clustering models by Cheeseman and Stutz (1995) and later adapted for graphical models by Chickering and Heckerman (1997). Variational scores were suggested by Attias (1999) and further elaborated by Beal and Ghahramani (2006).

Search based on structural expectation maximization was introduced by Friedman (1997, 1998) and further discussed in Meila and Jordan (2000); Thiesson et al. (1998). The selective clustering example of section 19.4.3.3 is based on Barash and Friedman (2002). Myers et al. (1999) suggested a method based on stochastic search. An alternative approach uses *reversible jump MCMC* methods that perform Monte Carlo search through both parameter space and structure space (Green 1995). More recent proposals use Dirichlet processes to integrate over potential structures (Rasmussen 1999; Wood et al. 2006).

reversible jump MCMC

Introduction of hidden variables is a classic problem. Pearl (1988) suggested a method based on algebraic constraints in the distribution. The idea of using algebraic signatures of hidden variables has been proposed in several works (Spirtes et al. 1993; Geiger and Meek 1998; Robins and Wasserman 1997; Tian and Pearl 2002; Kearns and Mansour 1998). Using the structural signature was suggested by Martin and VanLehn (1995) and developed more formally by Elidan et al. (2000). Additional methods include hierarchical methods (Zhang 2004; Elidan and Friedman 2005), the introduction of variables to capture temporal correlations (Boyen et al. 1999), and introduction of variables in networks of continuous variables (Elidan et al. 2007).

19.8 Exercises

Exercise 19.1

Consider the estimation problem in example 19.4.

a. Provide upper and lower bounds on the maximum likelihood estimate of θ.
b. Prove that your bounds are tight; that is, there are values of $\psi_{O_X|x^1}$ and $\psi_{O_X|x^0}$ for which these estimates are equal to the maximum likelihood.

Exercise 19.2⋆

Suppose we have a given model $P(\boldsymbol{X} \mid \boldsymbol{\theta})$ on a set of variable $\boldsymbol{X} = \{X_1, \ldots, X_n\}$, and some incomplete data. Suppose we introduce additional variables $\boldsymbol{Y} = \{Y_1, \ldots, Y_n\}$ so that Y_i has the value 1 if X_i is observed and 0 otherwise. We can extend the data, in the obvious way, to include complete observations of the variables $\boldsymbol{Y}$. Show how to augment the model to build a model $P(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{\theta}, \boldsymbol{\theta}') = P(\boldsymbol{X} \mid \boldsymbol{\theta})P(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta}')$ so that it satisfies the *missing at random* assumption.

missing at random

Exercise 19.3

Consider the problem of applying EM to parameter estimation for a variable X whose local probabilistic model is a tree-CPD. We assume that the network structure $\mathcal{G}$ includes the structure of the tree-CPDs in it, so that we have a structure $\mathcal{T}$ for X. We are given a data set $\mathcal{D}$ with some missing values, and we want to run EM to estimate the parameters of $\mathcal{T}$. Explain how we can adapt the EM algorithm in order to accomplish this task. Describe what expected sufficient statistics are computed in the E-step, and how parameters are updated in the M-step.

Exercise 19.4

Consider the problem of applying EM to parameter estimation for a variable X whose local probabilistic model is a noisy-or. Assume that X has parents $Y_1, \ldots, Y_k$, so that our task for X is to estimate the noise parameters $\lambda_0, \ldots, \lambda_k$. Explain how we can use the EM algorithm to accomplish this task. (Hint: Utilize the structural decomposition of the noisy-or node.)

Exercise 19.5

Prove theorem 19.2. (Hint: use lemma 19.1.)

Exercise 19.6

Suppose we are using a gradient method to learn parameters for a network with table-CPDs. Let X be one of the variables in the network with parents $\boldsymbol{U}$. One of the constraints we need to maintain is that

$$\sum_x \theta_{x|\boldsymbol{u}} = 1$$

for every assignment $\boldsymbol{u}$ for $\boldsymbol{U}$. Given the gradient $\frac{\partial}{\partial \theta_{x|\boldsymbol{u}}} \ell(\boldsymbol{\theta} : \mathcal{D})$, show how to project it to null space of this constraint. That is, show how to find a gradient direction that maximizes the likelihood while preserving this constraint.

Exercise 19.7

Suppose we consider reparameterizing table-CPDs using the representation of equation (19.3). Use the chain law of partial derivatives to find the form of $\frac{\partial}{\partial \lambda_{x|\boldsymbol{u}}} \ell(\boldsymbol{\theta} : \mathcal{D})$.

Exercise 19.8⋆

Suppose we have a Bayesian network with table-CPDs. Apply the method of Lagrange multipliers to characterize the maximum likelihood solution under the constraint that each conditional probability sums to one. How does your characterization relate to EM?

Exercise 19.9⋆

We now examine how to compute the Hessian of the likelihood function. Recall that the Hessian of the log-likelihood is the matrix of second derivatives. Assume that our model is a Bayesian network with table-CPDs.

a. Prove that the second derivative of the likelihood of an observation $\boldsymbol{o}$ is of the form:

$$\frac{\partial^2 \log P(\boldsymbol{o})}{\partial \theta_{x_i|\boldsymbol{u}_i} \partial \theta_{x_j|\boldsymbol{u}_j}} = \frac{1}{\theta_{x_i|\boldsymbol{u}_i} \theta_{x_j|\boldsymbol{u}_j}} \left[P(x_i, \boldsymbol{u}_i, x_j, \boldsymbol{u}_j \mid \boldsymbol{o}) - P(x_i, \boldsymbol{u}_i \mid \boldsymbol{o}) P(x_j, \boldsymbol{u}_j \mid \boldsymbol{o}) \right].$$

b. What is the cost of computing the full Hessian matrix of $\log P(\boldsymbol{o})$ if we use clique tree propagation?
c. What is the computational cost if we are only interested in entries of the form

$$\frac{\partial^2}{\partial\theta_{x_i|\boldsymbol{u}_i}\partial\theta_{x'_i|\boldsymbol{u}'_i}}\log P(\boldsymbol{o});$$

that is, we are interested in the "diagonal band" that involves only second derivatives of entries from the same family?

Exercise 19.10⋆

a. Consider the task of estimating the parameters of a univariate Gaussian distribution $\mathcal{N}\left(\mu;\sigma^2\right)$ from a data set $\mathcal{D}$. Show that if we maximize likelihood subject to the constraint $\sigma^2 \geq \epsilon$ for some $\epsilon > 0$, then the likelihood $L(\mu, \sigma^2 : \mathcal{D})$ is guaranteed to remain bounded.
b. Now, consider estimating the parameters of a multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}; \Sigma)$ from a data set $\mathcal{D}$. Provide constraints on Σ that achieve the same guarantee.

Exercise 19.11⋆

Consider learning the parameters of the network $H \to X, H \to Y$, where H is a hidden variable. Show that the distribution where $P(H)$, $P(X \mid H)$, $P(Y \mid H)$ are uniform is a stationary point of the likelihood (gradient is 0). What does that imply about gradient ascent and EM starting from this point?

Exercise 19.12

Prove theorem 19.5. Hint, note that $\ell(\boldsymbol{\theta}^t : \mathcal{D}) = F_{\mathcal{D}}[\boldsymbol{\theta}^t, P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta}^t)]$, and use corollary 19.1.

Exercise 19.13

Consider the task of learning the parameters of a DBN with table-CPDs from a data set with missing data. In particular, assume that our data set consists of a sequence of observations $\boldsymbol{o}_0^{(0)}, \boldsymbol{o}_1^{(1)}, \ldots, \boldsymbol{o}_t^{(T)}$. (Note that we do not assume that the same variables are observed in every time-slice.)

a. Describe precisely how you would run EM in this setting to estimate the model parameters; your algorithm should specify exactly how we run the E-step, which sufficient statistics we compute and how, and how the sufficient statistics are used within the M-step.
b. Given a single trajectory, as before, which of the network parameters might you be able to estimate?

Exercise 19.14⋆

Show that, until convergence, each iteration of hard-assignment EM increases $\ell(\boldsymbol{\theta} : \langle\mathcal{D}, \mathcal{H}\rangle)$.

Exercise 19.15⋆

Suppose that we have an incomplete data set $\mathcal{D}$, and network structure $\mathcal{G}$ and matching parameters. Moreover, suppose that we are interested in learning the parameters of a single CPD $P(X_i \mid \boldsymbol{U}_i)$. That is, we assume that the parameters we were given for all other families are frozen and do not change during the learning. This scenario can arise for several reasons: we might have good prior knowledge about these parameters; or we might be using an incremental approach, as mentioned in box 19.C (see also exercise 19.16).

We now consider how this scenario can change the computational cost of the EM algorithm.

a. Assume we have a clique tree for the network $\mathcal{G}$ and that the CPD $P(X_i \mid \boldsymbol{U}_i)$ was assigned to clique $\boldsymbol{C}_j$. Analyze which messages change after we update the parameters for $P(X_i \mid \boldsymbol{U}_i)$. Use this analysis to show how, after an initial precomputation step, we can perform iterations of this single-family EM procedure with a computational cost that depends only on the size of $\boldsymbol{C}_j$ and not the size of the rest of the cluster tree.

b. Would this conclusion change if we update the parameters of several families that are all assigned to the same cluster in the cluster tree?

Exercise 19.16⋆

incremental EM

We can build on the idea of the single-family EM procedure, as described in exercise 19.15, to define an *incremental EM* procedure for learning all the parameters in the network. In this approach, at each step we optimize the parameters of a single CPD (or several CPDs) while freezing the others. We then iterate between these local EM runs until all families have converged.

Is this modified version of EM still guaranteed to converge? In other words, does

$$\ell(\boldsymbol{\theta}^{t+1} : \mathcal{D}) \geq \ell(\boldsymbol{\theta}^{t} : \mathcal{D})$$

still hold? If so, prove the result. If not, explain why not.

Exercise 19.17⋆

We now consider how to use the interpretation of the EM as maximizing an energy functional to allow partial or incremental updates over the instances. Consider the EM algorithm of algorithm 19.2. In the Compute-ESS we collect the statistics from all the instances. This requires running inference on all the instances.

We now consider a procedure that performs partial updates where it update the expected sufficient statistics for some, but not all, of the instances. In particular, suppose we replace this procedure by one that runs inference on a single instance and uses the update to replace the old contribution of the instance with a new one; see algorithm 19.4. This procedure, instead of computing all the expected sufficient statistics in each E-step, caches the contribution of each instance to the sufficient statistics, and then updates only a single one in each iteration.

a. Show that the incremental EM algorithm converges to a fixed point of the log-likelihood function. To do so, show that each iteration improves the EM energy functional. Hint: you need to define what is the effect of the partial E-step on the energy functional.
b. How would that analysis generalize if in each iteration the algorithm performs a partial update for k instances (instead of 1)?
c. Assume that the computations in the M-step are relatively negligible compared to the inference in the E-step. Would you expect the incremental EM to be more efficient than standard EM? If so, why?

Exercise 19.18⋆

Consider the model described in box 19.D.

a. Assume we perform the E-step for each step $\boldsymbol{x}_m$ by defining

$$\tilde{P}(\boldsymbol{x}_m \mid C_m = k : \boldsymbol{\theta}_k) = \mathcal{N}\left(d(\boldsymbol{x}, p_k) \mid 0; \sigma^2\right)$$

and $\tilde{P}(\boldsymbol{x}_m \mid C_m = 0 : \boldsymbol{\theta}_k) = C$ for some constant C. Why is this formula not a correct application of EM? (Hint: Consider the normalizing constants.)

We note that although this approach is mathematically not quite right, it seems to be a reasonable approximation that works in practice.

b. Given a solution to the E-step, show how to perform maximum likelihood estimation of the model parameters $\boldsymbol{\alpha}_k, \beta_k$, subject to the constraint that $\boldsymbol{\alpha}_k$ be a unit-vector, that is, that $\boldsymbol{\alpha}_k \cdot \boldsymbol{\alpha}_k = 1$. (Hint: Use Lagrange multipliers.)

Algorithm 19.4 The incremental EM algorithm for network with table-CPDs

Procedure Incremental-E-Step (
  $\boldsymbol{\theta}$, // Parameters for update
  m, // instance to update
)
1 Run inference on $\langle \mathcal{G}, \boldsymbol{\theta} \rangle$ using evidence $\boldsymbol{o}[m]$
2 **for** each $i = 1, \ldots, n$
3 **for** each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
4 // Remove old contribution
5 $\bar{M}[x_i, \boldsymbol{u}_i] \leftarrow \bar{M}[x_i, \boldsymbol{u}_i] - \bar{M}_m[x_i, \boldsymbol{u}_i]$
6 // Compute new contribution
7 $\bar{M}_m[x_i, \boldsymbol{u}_i] \leftarrow P(x_i, \boldsymbol{u}_i \mid \boldsymbol{o}[m])$
8 $\bar{M}[x_i, \boldsymbol{u}_i] \leftarrow \bar{M}[x_i, \boldsymbol{u}_i] + \bar{M}_m[x_i, \boldsymbol{u}_i]$

Procedure Incremental-EM (
  $\mathcal{G}$, // Bayesian network structure over $X_1, \ldots, X_n$
  $\boldsymbol{\theta}^0$, // Initial set of parameters for $\mathcal{G}$
  $\mathcal{D}$ // Partially observed data set
)
1 **for** each $i = 1, \ldots, n$
2 **for** each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
3 $\bar{M}[x_i, \boldsymbol{u}_i] \leftarrow 0$
4 **for** each $m = 1 \ldots M$
5 $\bar{M}_m[x_i, \boldsymbol{u}_i] \leftarrow 0$
6 // Initialize the expected sufficient statistics
7 **for** each $m = 1 \ldots M$
8 Incremental-E-Step$(\mathcal{G}, \boldsymbol{\theta}^0, \mathcal{D}, m)$
9 $m \leftarrow 1$
10 **for** each $t = 0, 1 \ldots,$ until convergence
11 // E-step
12 Incremental-E-Step$(\mathcal{G}, \boldsymbol{\theta}^t, \mathcal{D}, m)$
13 $m \leftarrow (m \bmod M) + 1$
14 // M-step
15 **for** each $i = 1, \ldots, n$
16 **for** each $x_i, \boldsymbol{u}_i \in Val(X_i, \mathrm{Pa}_{X_i}^{\mathcal{G}})$
17 $\theta_{x_i|\boldsymbol{u}_i}^{t+1} \leftarrow \frac{\bar{M}[x_i, \boldsymbol{u}_i]}{\bar{M}[\boldsymbol{u}_i]}$
18 **return** $\boldsymbol{\theta}^t$

Exercise 19.19⋆

Consider the setting of exercise 12.29, but now assume that we cannot (or do not wish to) maintain a distribution over the $\boldsymbol{A}_j$'s. Rather, we want to find the assignment $\boldsymbol{a}_1^*, \ldots, \boldsymbol{a}_m^*$ for which $P(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m)$ is maximized.

In this exercise, we address this problem using the EM algorithm, treating the values $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ as parameters. In the E-step, we compute the expected value of the C_i variables; in the M-step, we maximize the value of the $\boldsymbol{a}_j$'s given the distribution over the C_j's.

a. Describe how one can implement this EM procedure exactly, that is, with no need for approximate inference.
b. Why is approximate inference necessary in exercise 12.29 but not here? Give a precise answer in terms of the properties of the probabilistic model.

Exercise 19.20

Suppose that a prior on a parameter vector is $p(\boldsymbol{\theta}) \sim Dirichlet(\alpha_1, \ldots, \alpha_k)$. Derive $\frac{\partial}{\partial \theta_i} \log p(\boldsymbol{\theta})$.

Exercise 19.21

Consider the generalization of the EM procedure to the task of finding the MAP parameters. Let

$$\tilde{F}_{\mathcal{D}}[\boldsymbol{\theta}, Q] = F_{\mathcal{D}}[\boldsymbol{\theta}, Q] + \log P(\boldsymbol{\theta}).$$

a. Prove the following result:

Corollary 19.2 *For a distribution Q, $\text{score}_{\text{MAP}}(\boldsymbol{\theta} : \mathcal{D}) \geq \tilde{F}_{\mathcal{D}}[\boldsymbol{\theta}, Q]$ with equality if only if $Q(\mathcal{H}) = P(\mathcal{H} \mid \mathcal{D}, \boldsymbol{\theta})$.*

b. Show that a coordinate ascent approach on $\tilde{F}_{\mathcal{D}}[\boldsymbol{\theta}, Q]$ requires only changing the M-step to perform MAP rather than ML estimation, that is, to maximize:

$$\boldsymbol{E}_Q[\ell(\boldsymbol{\theta} : \langle \mathcal{D}, \mathcal{H} \rangle)] + \log P(\boldsymbol{\theta}).$$

c. Using exercise 17.12, provide a specific formula for the M-step in a network with table-CPDs.

Exercise 19.22

In this case, we analyze the use of collapsed Gibbs with data completion particles for the purpose of sampling from a posterior in the case of incomplete data.

a. Consider first the simple case of example 19.12. Assuming that the data instances $\boldsymbol{x}$ are sampled from a discrete naive Bayes model with a Dirichlet prior, derive a closed form for equation (19.9).
b. Now, consider the general case of sampling from $P(\mathcal{H} \mid \mathcal{D})$. Here, the key step would involve sampling from the distribution

$$P(X_i[m] \mid \langle \mathcal{D}, \mathcal{H} \rangle_{-X_i[m]}) \propto P(X_i[m], \langle \mathcal{D}, \mathcal{H} \rangle_{-X_i[m]}),$$

where $\langle \mathcal{D}, \mathcal{H} \rangle_{-X_i[m]}$ is a complete data set from which the observation of $X_i[m]$ is removed.
Assuming we have table-CPD and independent Dirichlet priors over the parameters, derive this conditional probability from the form of the marginal likelihood of the data. Show how to use sufficient statistics of the particle to perform this sampling efficiently.

Exercise 19.23⋆

We now consider a Metropolis-Hastings sampler for the same setting as exercise 19.22. For simplicity, we assume that the same variables are hidden in each instance. Consider the proposal distribution for variable X_i specified in algorithm 19.5. (We are using a multiple-transition chain, as in section 12.3.2.4, where each variable has its own kernel.) In this proposal distribution, we resample a value for X_i in all of the instances, based on the current parameters and the completion for all the other variables.

Derive the form of the acceptance probability for this proposal distribution. Show how to use sufficient statistics of the completed data to evaluate this acceptance probability efficiently.

Algorithm 19.5 Proposal distribution for collapsed Metropolis-Hastings over data completions

```
Procedure Proposal-Distribution (
  G,    // Bayesian network structure over X_1, ..., X_n
  D     // completed data set
  X_i   // A variable to sample
)
1   θ ← Estimate-Parameters(D, G)
2   D' ← D
3   for each m = 1 ... M
4     Sample x'_i[m] from P(X_i[m] | x_{-i}[m], θ)
5   return D'
```

Exercise 19.24

Prove theorem 19.8.

Exercise 19.25⋆

Prove theorem 19.10. Hint: Use the proof of theorem 19.5.

Exercise 19.26

Consider learning structure in the setting discussed in section 19.4.3.3. Describe a data set $\mathcal{D}$ and parameters for a network where X_1 and C are independent, yet the expected sufficient statistics $\bar{M}[X_1, C]$ show dependency between X_1 and C.

Exercise 19.27

Consider using the structural EM algorithm to learn the structure associated with a hidden variable H; all other variables are fully observed. Assume that we start our learning process by performing an E-step in a network where H is not connected to any of $X_1, \ldots, X_n$. Show that, for any initial parameter assignment to $P(H)$, the SEM algorithm will not connect H to the rest of the variables in the network.

Exercise 19.28

Consider the task of learning a model involving a binary-valued hidden variable H using the EM algorithm. Assume that we initialize the EM algorithm using parameters that are symmetric in the two values of H; that is, for any variable X_i that has H has a parent, we have $P(X_i \mid \boldsymbol{U}_i, h^0) = P(X_i \mid \boldsymbol{U}_i, h^1)$. Show that, with this initialization, the model will remain symmetric in the two values of H, over all EM iterations.

Exercise 19.29

Derive the sampling update equations for the partition-based Gibbs sampling of equation (19.17) and equation (19.18) from the corresponding update equations over particles defined as ground assignments (equation (19.10)). Your update rules must sum over all assignments consistent with the partition.

Exercise 19.30

Consider the distribution over partitions induced by the Chinese restaurant process.

a. Find a closed-form formula for the probability induced by this process for any partition σ of the guests. Show that this probability is invariant to the order the guests enter the restaurant.
b. Show that a Gibbs sampling process over the partitions generated by this algorithm satisfies equation (19.19) and equation (19.20).

Algorithm 19.6 Proposal distribution over partitions in the Dirichlet process priof

```
    Procedure DP-Merge-Split-Proposal (
      σ    // A partition
    )
1     Uniformly choose two different instances m, l
2     if m, l are assigned to two different clusters I, I' then
3         // Propose partition that merges the two clusters
4       σ' ← σ − {I, I'} ∪ {I ∪ I'}
5     else
6       Let I be the cluster to which m, l are both assigned
7         // Propose to randomly split I so as to separate them
8       I_1 ← {m}
9       I_2 ← {l}
10      for n ∈ I
11        Add n to I_1 with probability 0.5 and to I_2 with probability 0.5
12      σ' ← σ − {I} ∪ {I_1, I_2}
13    return (σ')
```

Exercise 19.31⋆

Algorithm 19.6 presents a Metropolis-Hastings proposal distribution over partitions in the Dirichlet process prior. Compute the acceptance probability of the proposed move.

20 *Learning Undirected Models*

20.1 Overview

In previous chapters, we developed the theory and algorithms for learning Bayesian networks from data. In this chapter, we consider the task of learning Markov networks. Although many of the same concepts and principles arise, the issues and solutions turn out to be quite different.

Perhaps the most important reason for the differences is a key distinction between Markov networks and Bayesian networks: the use of a global normalization constant (the partition function) rather than local normalization within each CPD. This global factor couples all of the parameters across the network, preventing us from decomposing the problem and estimating local groups of parameters separately. This global parameter coupling has significant computational ramifications. As we will explain, in contrast to the situation for Bayesian networks, even simple (maximum-likelihood) parameter estimation with complete data cannot be solved in closed form (except for chordal Markov networks, which are therefore also Bayesian networks). Rather, we generally have to resort to iterative methods, such as gradient ascent, for optimizing over the parameter space. The good news is that the likelihood objective is concave, and so these methods are guaranteed to converge to the global optimum. The bad news is that each of the steps in the iterative algorithm requires that we run inference on the network, making even simple parameter estimation a fairly expensive, or even intractable, process. Bayesian estimation, which requires integration over the space of parameters, is even harder, since there is no closed-form expression for the parameter posterior. Thus, the integration associated with Bayesian estimation must be performed using approximate inference (such as variational methods or MCMC), a burden that is often infeasible in practice.

As a consequence of these computational issues, much of the work in this area has gone into the formulation of alternative, more tractable, objectives for this estimation problem. Other work has been focused on the use of approximate inference algorithms for this learning problem and on the development of new algorithms suited to this task.

The same issues have significant impact on structure learning. In particular, because a Bayesian parameter posterior is intractable to compute, the use of exact Bayesian scoring for model selection is generally infeasible. In fact, scoring any model (computing the likelihood) requires that we run inference to compute the partition function, greatly increasing the cost of search over model space. Thus, here also, the focus has been on approximations and heuristics that can reduce the computational cost of this task. Here, however, there is some good news, arising from another key distinction between Bayesian and Markov networks: the lack of a

global acyclicity constraint in undirected models. Recall (see theorem 18.5) that the acyclicity constraint couples decisions regarding the family of different variables, thereby making the structure selection problem much harder. The lack of such a global constraint in the undirected case eliminates these interactions, allowing us to choose the local structure locally in different parts of the network. In particular, it turns out that a particular variant of the structure learning task can be formulated as a continuous, convex optimization problem, a class of problems generally viewed as tractable. Thus, elimination of global acyclicity removes the main reason for the $\mathcal{NP}$-hardness of structure learning that we saw in Bayesian networks. However, this does *not* make structure learning of Markov networks efficient; the convex optimization process (as for parameter estimation) still requires multiple executions of inference over the network.

A final important issue that arises in the context of Markov networks is the overwhelmingly common use of these networks for settings, such as image segmentation and others, where we have a particular inference task in mind. In these settings, we often want to train a network *discriminatively* (see section 16.3.2), so as to provide good performance for our particular prediction task. Indeed, much of Markov network learning is currently performed for CRFs.

The remainder of this chapter is structured as follows. We begin with the analysis of the properties of the likelihood function, which, as always, forms the basis for all of our discussion of learning. We then discuss how the likelihood function can be optimized to find the maximum likelihood parameter estimates. The ensuing sections discuss various important extensions to these basic ideas: conditional training, parameter priors for MAP estimation, structure learning, learning with missing data, and approximate learning methods that avoid the computational bottleneck of multiple iterations of network inference. These extensions are usually described as building on top of standard maximum-likelihood parameter estimation. However, it is important to keep in mind that they are largely orthogonal to each other and can be combined. Thus, for example, we can also use the approximate learning methods in the case of structure learning or of learning with missing data. Similarly, all of the methods we described can be used with maximum conditional likelihood training. We return to this issue in section 20.8.

We note that, for convenience and consistency with standard usage, we use natural logarithms throughout this chapter, including in our definitions of entropy or KL-divergence.

20.2 The Likelihood Function

As we saw in earlier chapters, the key component in most learning tasks is the likelihood function. In this section, we discuss the form of the likelihood function for Markov networks, its properties, and their computational implications.

20.2.1 An Example

As we suggested, the existence of a global partition function couples the different parameters in a Markov network, greatly complicating our estimation problem. To understand this issue, consider the very simple network A—B—C, parameterized by two potentials $\phi_1(A, B)$ and $\phi_2(B, C)$. Recall that the log-likelihood of an instance $\langle a, b, c\rangle$ is

$$\ln P(a, b, c) = \ln \phi_1(a, b) + \ln \phi_2(b, c) - \ln Z,$$

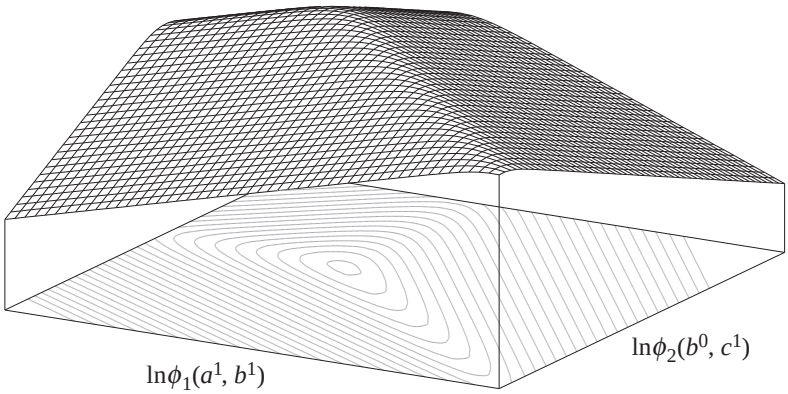


Figure 20.1 Log-likelihood surface for the Markov network A—B—C, as a function of $\ln \phi_1(a^1, b^1)$ (x-axis) and $\ln \phi_2(b^0, c^1)$ (y-axis); all other parameters in both potentials are set to 1. Surface is viewed from the $(+\infty, +\infty)$ point toward the $(-, -)$ quadrant. The data set $\mathcal{D}$ has $M = 100$ instances, for which $M[a^1, b^1] = 40$ and $M[b^0, c^1] = 40$. (The other sufficient statistics are irrelevant, since all of the other log-parameters are 0.)

where Z is the partition function that ensures that the distribution sums up to one. Now, consider the log-likelihood function for a data set $\mathcal{D}$ containing M instances:

$$\begin{aligned}\ell(\boldsymbol{\theta} : \mathcal{D}) &= \sum_m (\ln \phi_1(a[m], b[m]) + \ln \phi_2(b[m], c[m]) - \ln Z(\boldsymbol{\theta})) \\ &= \sum_{a,b} M[a, b] \ln \phi_1(a, b) + \sum_{b,c} M[b, c] \ln \phi_2(b, c) - M \ln Z(\boldsymbol{\theta}).\end{aligned}$$

Thus, we have sufficient statistics that summarize the data: the joint counts of variables that appear in each potential. This is analogous to the situation in learning Bayesian networks, where we needed the joint counts of variables that appear within the same family. This likelihood consists of three terms. The first term involves ϕ_1 alone, and the second term involves ϕ_2 alone. The third term, however, is the log-partition function $\ln Z$, where:

$$Z(\boldsymbol{\theta}) = \sum_{a,b,c} \phi_1(a, b)\phi_2(b, c).$$

Thus, $\ln Z(\boldsymbol{\theta})$ is a function of both ϕ_1 and ϕ_2. As a consequence, it *couples* the two potentials in the likelihood function.

Specifically, consider maximum likelihood estimation, where we aim to find parameters that maximize the log-likelihood function. In the case of Bayesian networks, we could estimate each conditional distribution independently of the other ones. Here, however, when we change one of the potentials, say ϕ_1, the partition function changes, possibly changing the value of ϕ_2 that maximizes $-\ln Z(\boldsymbol{\theta})$. Indeed, as illustrated in figure 20.1, the log-likelihood function in our simple example shows clear dependencies between the two potentials.

In this particular example, we can avoid this problem by noting that the network A—B—C is equivalent to a Bayesian network, say $A \to B \to C$. Therefore, we can learn the parameters

of this BN, and then define $\phi_1(A, B) = P(A)P(B \mid A)$ and $\phi_2(B, C) = P(C \mid B)$. Because the two representations have equivalent expressive power, the same maximum likelihood is achievable in both, and so the resulting parameterization for the Markov network will also be a maximum-likelihood solution. In general, however, there are Markov networks that do not have an equivalent BN structure, for example, the diamond-structured network of figure 4.13 (see section 4.5.2). In such cases, we generally cannot convert a learned BN parameterization into an equivalent MN; indeed, the optimal likelihood achievable in the two representations is generally not the same.

20.2.2 Form of the Likelihood Function

To provide a more general description of the likelihood function, it first helps to provide a more convenient notational basis for the parameterization of these models. For this purpose, we use the framework of *log-linear models*, as defined in section 4.4.1.2. Given a set of features $\mathcal{F} = \{f_i(\boldsymbol{D}_i)\}_{i=1}^k$, where $f_i(\boldsymbol{D}_i)$ is a feature function defined over the variables in $\boldsymbol{D}_i$, we have:

log-linear model

$$P(X_1, \ldots, X_n : \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left\{\sum_{i=1}^{k} \theta_i f_i(\boldsymbol{D}_i)\right\}. \tag{20.1}$$

As usual, we use $f_i(\xi)$ as shorthand for $f_i(\xi\langle\boldsymbol{D}_i\rangle)$. The parameters of this distribution correspond to the weight we put on each feature. When $\theta_i = 0$, the feature is ignored, and it has no effect on the distribution.

As discussed in chapter 4, this representation is very generic and can capture Markov networks with global structure and local structure. A special case of particular interest is when $f_i(\boldsymbol{D}_i)$ is a binary indicator function that returns the value 0 or 1. With such features, we can encode a "standard" Markov network by simply having one feature per potential entry. In more general, however, we can consider arbitrary valued features.

Example 20.1 *As a specific example, consider the simple diamond network of figure 3.10a, where we take all four variables to be binary-valued. The features that correspond to this network are sixteen indicator functions: four for each assignment of variables to each of our four clusters. For example, one such feature would be:*

$$f_{a^0,b^0}(a, b) = \boldsymbol{I}\{a = a^0\}\boldsymbol{I}\{b = b^0\}.$$

With this representation, the weight of each indicator feature is simply the natural logarithm of the corresponding potential entry. For example, $\theta_{a^0,b^0} = \ln \phi_1(a^0, b^0)$. ■

Given a model in this form, the log-likelihood function has a simple form.

Proposition 20.1 *Let $\mathcal{D}$ be a data set of M examples, and let $\mathcal{F} = \{f_i : i = 1, \ldots, k\}$ be a set of features that define a model. Then the log-likelihood is*

$$\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \left(\sum_m f_i(\xi[m])\right) - M \ln Z(\boldsymbol{\theta}). \tag{20.2}$$

sufficient statistics

The *sufficient statistics* of this likelihood function are the sums of the feature values in the instances in $\mathcal{D}$. We can derive a more elegant formulation if we divide the log-likelihood by the number of samples M.

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{d}_i)] - \ln Z(\boldsymbol{\theta}), \tag{20.3}$$

where $\boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{d}_i)]$ is the empirical expectation of f_i, that is, its average in the data set.

20.2.3 Properties of the Likelihood Function

The formulation of proposition 20.1 describes the likelihood function as a sum of two functions. The first function is linear in the parameters; increasing the parameters directly increases this linear term. Clearly, because the log-likelihood function (for a fixed data set) is upper-bounded (the probability of an event is at most 1), the second term $\ln Z(\boldsymbol{\theta})$ balances the first term.

Let us examine this second term in more detail. Recall that the partition function is defined as

$$\ln Z(\boldsymbol{\theta}) = \ln \sum_{\xi} \exp\left\{\sum_i \theta_i f_i(\xi)\right\}.$$

convex partition function

One important property of the partition function is that it is *convex* in the parameters $\boldsymbol{\theta}$. Recall that a function $f(\vec{x})$ is convex if for every $0 \leq \alpha \leq 1$,

$$f(\alpha\vec{x} + (1-\alpha)\vec{y}) \leq \alpha f(\vec{x}) + (1-\alpha) f(\vec{y}).$$

In other words, the function is bowl-like, and every interpolation between the images of two points is larger than the image of their interpolation. One way to prove formally that the function f is convex is to show that the *Hessian* — the matrix of the function's second derivatives — is positive semidefinite. Therefore, we now compute the derivatives of $Z(\boldsymbol{\theta})$.

Hessian

Proposition 20.2 *Let $\mathcal{F}$ be a set of features. Then,*

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i] \\ \frac{\partial^2}{\partial\theta_i\partial\theta_j}\ln Z(\boldsymbol{\theta}) &= \boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j],\end{aligned}$$

where $\boldsymbol{E}_{\boldsymbol{\theta}}[f_i]$ is a shorthand for $\boldsymbol{E}_{P(\mathcal{X}:\boldsymbol{\theta})}[f_i]$.

Proof The first derivatives are computed as:

$$\begin{aligned}\frac{\partial}{\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi}\frac{\partial}{\partial\theta_i}\exp\left\{\sum_j \theta_j f_j(\xi)\right\} \\ &= \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)\exp\left\{\sum_j \theta_j f_j(\xi)\right\} \\ &= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i].\end{aligned}$$

We now consider the second derivative:

$$\begin{aligned}
\frac{\partial^2}{\partial\theta_j\partial\theta_i}\ln Z(\boldsymbol{\theta}) &= \frac{\partial}{\partial\theta_j}\left[\frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\}\right] \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}\left(\frac{\partial}{\partial\theta_j}Z(\boldsymbol{\theta})\right)\sum_{\xi} f_i(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&\quad + \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)f_j(\xi)\exp\left\{\sum_k \theta_k f_k(\xi)\right\} \\
&= -\frac{1}{Z(\boldsymbol{\theta})^2}Z(\boldsymbol{\theta})\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_{\xi} f_i(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&\quad + \frac{1}{Z(\boldsymbol{\theta})}\sum_{\xi} f_i(\xi)f_j(\xi)\tilde{P}(\xi:\boldsymbol{\theta}) \\
&= -\boldsymbol{E}_{\boldsymbol{\theta}}[f_j]\sum_{\xi} f_i(\xi)P(\xi:\boldsymbol{\theta}) \\
&\quad + \sum_{\xi} f_i(\xi)f_j(\xi)P(\xi:\boldsymbol{\theta}) \\
&= \boldsymbol{E}_{\boldsymbol{\theta}}[f_i f_j] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]\boldsymbol{E}_{\boldsymbol{\theta}}[f_j] \\
&= \boldsymbol{Cov}_{\boldsymbol{\theta}}[f_i; f_j].
\end{aligned}$$

Thus, the Hessian of $\ln Z(\boldsymbol{\theta})$ is the covariance matrix of the features, viewed as random variables distributed according to distribution defined by $\boldsymbol{\theta}$. Because a covariance matrix is always positive semidefinite, it follows that the Hessian is positive semidefinite, and hence that $\ln Z(\boldsymbol{\theta})$ is a convex function of $\boldsymbol{\theta}$. ■

Because $\ln Z(\boldsymbol{\theta})$ is convex, its complement $(-\ln Z(\boldsymbol{\theta}))$ is concave. The sum of a linear function and a concave function is concave, implying the following important result:

Corollary 20.1

The log-likelihood function is concave.

This result implies that the log-likelihood is unimodal and therefore has no local optima. It does not, however, imply the uniqueness of the global optimum: Recall that a parameterization of the Markov network can be *redundant*, giving rise to multiple representations of the same distribution. The standard parameterization of a set of table factors for a Markov network — a feature for every entry in the table — is always redundant. In our simple example, for instance, we have:

redundant parameterization

$$f_{a^0,b^0} = 1 - f_{a^0,b^1} - f_{a^1,b^0} - f_{a^1,b^1}.$$

We thus have a continuum of parameterizations that all encode the same distribution, and (necessarily) give rise to the same log-likelihood. Thus, there is a unique globally optimal value for the log-likelihood function, but not necessarily a unique solution. In general, because the function is concave, we are guaranteed that there is a convex region of continuous global optima.

It is possible to eliminate the redundancy by removing some of the features. However, as we discuss in section 20.4, that turns out to be unnecessary, and even harmful, in practice.

We note that we have defined the likelihood function in terms of a standard log-linear parameterization, but the exact same derivation also holds for networks that use shared parameters, as in section 6.5; see exercise 20.1 and exercise 20.2.

20.3 Maximum (Conditional) Likelihood Parameter Estimation

We now move to the question of estimating the parameters of a Markov network with a fixed structure, given a fully observable data set $\mathcal{D}$. We focus in this section on the simplest variant of this task — maximum-likelihood parameter estimation, where we select parameters that maximize the log-likelihood function of equation (20.2). In later sections, we discuss alternative objectives for the parameter estimation task.

20.3.1 Maximum Likelihood Estimation

As for any function, the gradient of the log-likelihood must be zero at its maximum points. For a concave function, the maxima are precisely the points at which the gradient is zero. Using proposition 20.2, we can compute the gradient of the average log-likelihood as follows:

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]. \tag{20.4}$$

This analysis provides us with a precise characterization of the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$:

Theorem 20.1 *Let $\mathcal{F}$ be a set of features. Then, $\boldsymbol{\theta}$ is a maximum-likelihood parameter assignment if and only if $\boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] = \boldsymbol{E}_{\hat{\boldsymbol{\theta}}}[f_i]$ for all i.*

In other words, at the maximal likelihood parameters $\hat{\boldsymbol{\theta}}$, the expected value of each feature relative to $P_{\hat{\boldsymbol{\theta}}}$ matches its empirical expectation in $\mathcal{D}$. In other words, we want the *expected sufficient statistics* in the learned distribution to match the empirical expectations. This type of equality constraint is also called *moment matching*. This theorem easily implies that maximum likelihood estimation is *consistent* in the same sense as definition 18.1: if the model is sufficiently expressive to capture the data-generating distribution, then, at the large sample limit, the optimum of the likelihood objective is the true model; see exercise 20.3.

expected sufficient statistics

moment matching

MLE consistency

By itself, this criterion does not provide a constructive definition of the maximum likelihood parameters. **Unfortunately, although the function is concave, there is no analytical form for its maximum. Thus, we must resort to iterative methods that search for the global optimum. Most commonly used are the gradient ascent methods reviewed in appendix A.5.2, which iteratively take steps in parameter space to improve the objective.** At each iteration, they compute the gradient, and possibly the Hessian, at the current point $\boldsymbol{\theta}$, and use those estimates to approximate the function at the current neighborhood. They then take a step in the right direction (as dictated by the approximation) and repeat the process. Due to the convexity of the problem, this process is guaranteed to converge to a global optimum, regardless of our starting point.

To apply these gradient-based methods, we need to compute the gradient. Fortunately, equation (20.4) provides us with an exact formula for the gradient: the difference between the feature's empirical count in the data and its expected count relative to our current parameterization $\boldsymbol{\theta}$. For example, consider again the fully parameterized network of example 20.1. Here, the features are simply indicator functions; the empirical count for a feature such as $f_{a^0,b^0}(a,b) = \mathbf{1}\{a = a^0\}\mathbf{1}\{b = b^0\}$ is simply the empirical frequency, in the data set $\mathcal{D}$, of the event a^0, b^0. At a particular parameterization $\boldsymbol{\theta}$, the expected count is simply $P_{\boldsymbol{\theta}}(a^0, b^0)$. Very naturally, the gradient for the parameter associated with this feature is the difference between these two numbers.

However, this discussion ignores one important aspect: the computation of the expected counts. In our example, for instance, we must compute the different probabilities of the form $P_{\boldsymbol{\theta}^t}(a,b)$. Clearly, this computation requires that we run inference over the network. As for the case of EM in Bayesian networks, a feature is necessarily part of a factor in the original network, and hence, due to family preservation, all of the variables involved in a feature must occur together in a cluster in a clique tree or cluster graph. Thus, a single inference pass that calibrates an entire cluster graph or tree suffices to compute all of the expected counts. Nevertheless, **a full inference step is required at every iteration of the gradient ascent procedure. Because inference is almost always costly in time and space, the computational cost of parameter estimation in Markov networks is usually high, sometimes prohibitively so.** In section 20.5 we return to this issue, considering the use of approximate methods that reduce the computational burden.

Our discussion does not make a specific choice of algorithm to use for the optimization. In practice, standard gradient ascent is not a particularly good algorithm, both because of its slow convergence rate and because of its sensitivity to the step size. Much faster convergence is obtained with second-order methods, which utilize the Hessian to provide a quadratic approximation to the function. However, from proposition 20.2 we can conclude that the *Hessian* of the log-likelihood function has the form:

log-likelihood Hessian

$$\frac{\partial}{\partial \theta_i \partial \theta_j} \ell(\boldsymbol{\theta} : \mathcal{D}) = -M \mathbf{Cov}_{\boldsymbol{\theta}}[f_i; f_j]. \tag{20.5}$$

To compute the Hessian, we must compute the joint expectation of two features, a task that is often computationally infeasible. Currently, one commonly used solution is the *L-BFGS algorithm*, a gradient-based algorithm that uses line search to avoid computing the Hessian (see appendix A.5.2 for some background).

L-BFGS algorithm

20.3.2 Conditionally Trained Models

As we discussed in section 16.3.2, we often want to use a Markov network to perform a particular inference task, where we have a known set of observed variables, or features, $\boldsymbol{X}$, and a predetermined set of variables, $\boldsymbol{Y}$, that we want to query. In this case, we may prefer to use *discriminative training*, where we train the network as a *conditional random field* (CRF) that encodes a conditional distribution $P(\boldsymbol{Y} \mid \boldsymbol{X})$.

discriminative training

conditional random field

More formally, in this setting, our training set consists of pairs $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^{M}$, specifying assignments to $\boldsymbol{Y}, \boldsymbol{X}$. An appropriate objective function to use in this situation is the *conditional likelihood* or its logarithm, defined in equation (16.3). In our setting, the

conditional likelihood

log-conditional-likelihood has the form:

$$\ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) = \ln P(\boldsymbol{y}[1, \ldots, M] \mid \boldsymbol{x}[1, \ldots, M], \boldsymbol{\theta}) = \sum_{m=1}^{M} \ln P(\boldsymbol{y}[m] \mid \boldsymbol{x}[m], \boldsymbol{\theta}). \tag{20.6}$$

In this objective, we are optimizing the likelihood of each observed assignment $\boldsymbol{y}[m]$ given the corresponding observed assignment $\boldsymbol{x}[m]$. Each of the terms $\ln P(\boldsymbol{y}[1, \ldots, M] \mid \boldsymbol{x}[1, \ldots, M], \boldsymbol{\theta})$ is a log-likelihood of a Markov network model with a different set of factors — the factors in the original network, reduced by the observation $\boldsymbol{x}[1, \ldots, M]$ — and its own partition function. Each term is thereby a concave function, and because the sum of concave functions is concave, we conclude:

Corollary 20.2

The log conditional likelihood of equation (20.6) is a concave function.

As for corollary 20.1, this result implies that the function has a global optimum and no local optima, but not that the global optimum is unique. Here also, redundancy in the parameterization may give rise to a convex region of contiguous global optima.

The approaches for optimizing this objective are similar to those used for optimizing the likelihood objective in the unconditional case. The objective function is a concave function, and so a gradient ascent process is guaranteed to give rise to the unique global optimum. The form of the gradient here can be derived directly from equation (20.4). We first observe that the gradient of a sum is the sum of the gradients of the individual terms. Here, each term is, in fact, a log-likelihood — the log-likelihood of a single data case $\boldsymbol{y}[m]$ in the Markov network obtained by reducing our original model to the context $\boldsymbol{x}[m]$. A reduced Markov network is itself a Markov network, and so we can apply equation (20.4) and conclude that:

$$\frac{\partial}{\partial \theta_i} \ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_{m=1}^{M} \left(f_i(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i \mid \boldsymbol{x}[m]] \right). \tag{20.7}$$

This solution looks deceptively similar to equation (20.4). Indeed, if we aggregate the first component in each of the summands, we obtain precisely the empirical count of f_i in the data set $\mathcal{D}$. There is, however, one key difference. In the unreduced Markov network, the expected feature counts are computed relative to a single model; in the case of the conditional Markov network, these expected counts are computed as the summation of counts in an ensemble of models, defined by the different values of the conditioning variables $\boldsymbol{x}[m]$. This difference has significant computational consequences. Recall that computing these expectations involves running inference over the model. **Whereas in the unconditional case, each gradient step required only a single execution of inference, when training a CRF, we must (in general) execute inference *for every single data case*, conditioning on $\boldsymbol{x}[m]$. On the other hand, the inference is executed on a simpler model, since conditioning on evidence in a Markov network can only reduce the computational cost.** For example, the network of figure 20.2 is very densely connected, whereas the reduced network over $\boldsymbol{Y}$ alone (conditioned on $\boldsymbol{X}$) is a simple chain, allowing linear-time inference.

Discriminative training can be particularly beneficial in cases where the domain of $\boldsymbol{X}$ is very large or even infinite. For example, in our image classification task, the partition function in the

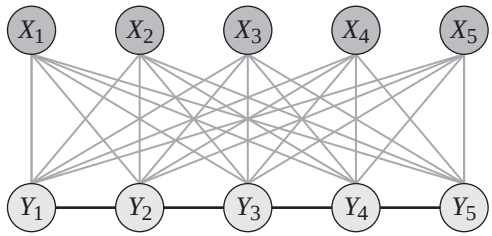


Figure 20.2 A highly connected CRF that allows simple inference when conditioned: The edges that disappear in the reduced Markov network after conditioning on $\boldsymbol{X}$ are marked in gray; the remaining edges form a simple linear chain.

generative setting involves summation (or integration) over the space of all possible images; if we have an $N \times N$ image where each pixel can take 256 values, the resulting space has 256^{N^2} values, giving rise to a highly intractable inference problem (even using approximate inference methods).

Box 20.A — Concept: Generative and Discriminative Models for Sequence Labeling. *One of the main tasks to which probabilistic graphical models have been applied is that of taking a set of interrelated instances and jointly labeling them, a process sometimes called* collective classification. *We have already seen examples of this task in box 4.B and in box 4.E; many other examples exist. Here, we discuss some of the trade-offs between different models that one can apply to this task. We focus on the context of labeling instances organized in a sequence, since it is simpler and allows us to illustrate another important point.*

collective classification

sequence labeling

activity recognition

In the sequence labeling *task, we get as input a sequence of observations $\boldsymbol{X}$ and need to label them with some joint label $\boldsymbol{Y}$. For example, in text analysis (box 4.E), we might have a sequence of words each of which we want to label with some label. In a task of* activity recognition*, we might obtain a sequence of images and want to label each frame with the activity taking place in it (for example, running, jumping, walking). We assume that we want to construct a model for this task and to train it using fully labeled training data, where both $\boldsymbol{Y}$ and $\boldsymbol{X}$ are observed.*

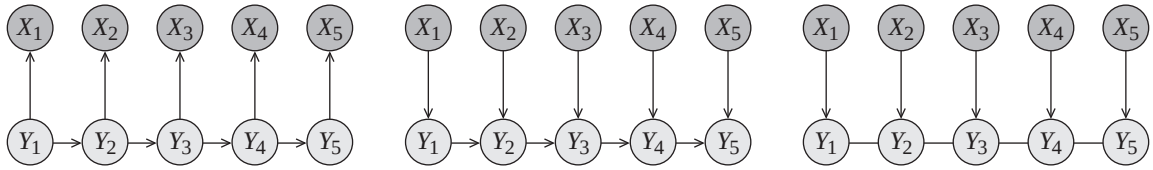
hidden Markov model

maximum entropy Markov model

conditional random field

Figure 20.A.1 illustrates three different types of models that have been proposed and used for sequence labeling, all of which we have seen earlier in this book (see figure 6.2 and figure 4.14). The first model is a hidden Markov model *(or HMM), which is a purely generative model: the model generates both the labels $\boldsymbol{Y}$ and the observations $\boldsymbol{X}$. The second is called a* maximum entropy Markov model *(or MEMM). This model is also directed, but it represents a conditional distribution $P(\boldsymbol{Y} \mid \boldsymbol{X})$; hence, there is no attempt to model a distribution over the $\boldsymbol{X}$'s. The final model is the* conditional random field *(or CRF) of section 4.6.1. This model also encodes a conditional distribution; hence the arrows from $\boldsymbol{X}$ to $\boldsymbol{Y}$. However, here the interactions between the $\boldsymbol{Y}$ are modeled as undirected edges.*

These different models present interesting trade-offs in terms of their expressive power and learnability. First, from a computational perspective, HMMs and MEMMs are much more easily learned. As purely directed models, their parameters can be computed in closed form using either maximum-likelihood or Bayesian estimation (see chapter 17); conversely, the CRF requires that we use an



(a) *HMM* (b) *MEMM* (c) *CRF*

Figure 20.A.1 — Different models for sequence labeling: HMM, MEMM, and CRF

iterative gradient-based approach, which is considerably more expensive (particularly here, when inference must be run separately for every training sequence; see section 20.3.2).

A second important issue relates to our ability to use a rich feature set. As we discussed in example 16.3 and in box 4.E, our success in a classification task often depends strongly on the quality of our features. In an HMM, we must explicitly model the distribution over the features, including the interactions between them. This type of model is very hard, and often impossible, to construct correctly. The MEMM and the CRF are both discriminative models, and therefore they avoid this challenge entirely.

The third and perhaps subtler issue relates to the independence assumptions made by the model. As we discussed in section 4.6.1.2, the MEMM makes the independence assumption that $(Y_i \perp X_j \mid \boldsymbol{X}_{-j})$ *for any* $j > i$*. Thus, an observation from later in the sequence has absolutely no effect on the posterior probability of the current state; or, in other words, the model does not allow for any smoothing. The implications of this can be severe in many settings. For example, consider the task of activity recognition from a video sequence; here, we generally assume that activities are highly persistent: if a person is walking in one frame, she is also extremely likely to be walking in the next frame. Now, imagine that the person starts running, but our first few observations in the sequence are ambiguous and consistent with both running and walking. The model will pick one — the one whose probability given that one frame is highest — which may well be walking. Assuming that activities are persistent, this choice of activity is likely to stay high for a large number of steps; the posterior of the initial activity will never change. In other words, the best we can expect is a prediction where the initial activity is walking, and then (perhaps) transitions to running. The model is incapable of going back and changing its prediction about the first few frames. This problem has been called the* label bias problem.

label bias problem

To summarize, the trade-offs between these different models are subtle and non-definitive. In cases where we have many correlated features, discriminative models are probably better; but, if only limited data are available, the stronger bias of the generative model may dominate and allow learning with fewer samples. Among the discriminative models, MEMMs should probably be avoided in cases where many transitions are close to deterministic. In many cases, CRFs are likely to be a safer choice, but the computational cost may be prohibitive for large data sets.

20.3.3 Learning with Missing Data

We now turn to the problem of parameter estimation in the context of missing data. As we saw in section 19.1, the introduction of missing data introduces both conceptual and technical difficulties. In certain settings, we may need to model explicitly the process by which data are observed. Parameters may not be identifiable from the data. And the likelihood function becomes significantly more complex: there is coupling between the likelihood's dependence on different parameters; worse, the function is no longer concave and generally has multiple local maxima.

The same issues regarding observation processes (ones that are not missing at random) and identifiability arise equally in the context of Markov network learning. The issue regarding the complexity of the likelihood function is analogous, although not quite the same. In the case of Markov networks, of course, we have coupling between the parameters even in the likelihood function for complete data. However, as we discuss, in the complete data case, the log-likelihood function is concave and easily optimized using gradient methods. Once we have missing data, we lose the concavity of the function and can have multiple local maxima. Indeed, the example we used was in the context of a Bayesian network of the form $X \rightarrow Y$, which can also be represented as a Markov network. Of course, the parameterization of the two models is not the same, and so the form of the function may differ. However, one can verify that a function that is multimodal in one parameterization will also be multimodal in the other.

20.3.3.1 Gradient Ascent

As in the case of Bayesian networks, if we assume our data is missing at random, we can perform maximum-likelihood parameter estimation by using some form of gradient ascent process to optimize the likelihood function. Let us therefore begin by analyzing the form of the gradient in the case of missing data. Let $\mathcal{D}$ be a data set where some entries are missing; let $\boldsymbol{o}[m]$ be the observed entries in the mth data instance and $\mathcal{H}[m]$ be the random variables that are the missing entries in that instance, so that for any $\boldsymbol{h}[m] \in Val(\mathcal{H}[m])$, $(\boldsymbol{o}[m], \boldsymbol{h}[m])$ is a complete assignment to $\mathcal{X}$.

As usual, the average log-likelihood function has the form:

$$\begin{aligned} \frac{1}{M} \ln P(\mathcal{D} \mid \boldsymbol{\theta}) &= \frac{1}{M} \sum_{m=1}^{M} \ln \left(\sum_{\boldsymbol{h}[m]} P(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) \right) \\ &= \frac{1}{M} \sum_{m=1}^{M} \ln \left(\sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) \right) - \ln Z. \end{aligned} \tag{20.8}$$

Now, consider a single term within the summation, $\sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta})$. This expression has the same form as a partition function; indeed, it is precisely the partition function for the Markov network that we would obtain by reducing our original Markov network with the observation $\boldsymbol{o}[m]$, to obtain a Markov network representing the conditional distribution $\tilde{P}(\mathcal{H}[m] \mid \boldsymbol{o}[m])$. Therefore, we can apply proposition 20.2 and conclude that:

$$\frac{\partial}{\partial \theta_i} \ln \sum_{\boldsymbol{h}[m]} \tilde{P}(\boldsymbol{o}[m], \boldsymbol{h}[m] \mid \boldsymbol{\theta}) = \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] \mid \boldsymbol{o}[m], \boldsymbol{\theta})}[f_i],$$

that is, the gradient of this term is simply the *conditional* expectation of the feature, given the observations in this instance.

Putting this together with previous computations, we obtain the following:

Proposition 20.3 *For a data set $\mathcal{D}$*

$$\frac{\partial}{\partial \theta_i} \frac{1}{M} \ell(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \left[\sum_{m=1}^{M} \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] | \boldsymbol{o}[m], \boldsymbol{\theta})} [f_i] \right] - \boldsymbol{E}_{\boldsymbol{\theta}}[f_i]. \tag{20.9}$$

In other words, the gradient for feature f_i in the case of missing data is the difference between two expectations — the feature expectation *over the data and the hidden variables* minus the feature expectation over all of the variables.

It is instructive to compare the cost of this computation to that of computing the gradient in equation (20.4). For the latter, to compute the second term in the derivative, we need to run inference once, to compute the expected feature counts relative to our current distribution $P(\mathcal{X} \mid \boldsymbol{\theta})$. The first term is computed by simply aggregating the feature over the data. By comparison, to compute the derivative here, we actually need to run inference separately for every instance m, conditioning on $\boldsymbol{o}[m]$. Although inference in the reduced network may be simpler (since reduced factors are simpler), the cost of this computation is still much higher than learning without missing data. Indeed, not surprisingly, the cost here is comparable to the cost of a single iteration of gradient descent or EM in Bayesian network learning.

20.3.3.2 Expectation Maximization

As for any other probabilistic model, an alternative method for parameter estimation in context of missing data is via the expectation maximization algorithm. In the case of Bayesian network learning, EM seemed to have significant advantages. Can we define a variant of EM for Markov networks? And does it have the same benefits?

The answer to the first question is clearly yes. We can perform an E-step by using our current parameters $\boldsymbol{\theta}^{(t)}$ to compute the expected sufficient statistics, in this case, the expected feature counts. That is, at iteration t of the EM algorithm, we compute, for each feature f_i, the expected sufficient statistic:

$$\bar{M}_{\boldsymbol{\theta}^{(t)}}[f_i] = \frac{1}{M} \left[\sum_{m=1}^{M} \boldsymbol{E}_{\boldsymbol{h}[m] \sim P(\mathcal{H}[m] | \boldsymbol{o}[m], \boldsymbol{\theta})} [f_i] \right].$$

With these expected feature counts, we can perform an M-step by doing maximum likelihood parameter estimation. The proofs of convergence and other properties of the algorithm go through unchanged.

Here, however, there is one critical difference. Recall that, in the case of directed models, given the expected sufficient statistics, we can perform the M-step efficiently, in closed form. By contrast, the M-step for Markov networks requires that we run inference multiple times, once for each iteration of whatever gradient ascent procedure we are using. At step k of this "inner-loop" optimization, we now have a gradient of the form:

$$\bar{M}_{\boldsymbol{\theta}^{(t)}}[f_i] - \boldsymbol{E}_{\boldsymbol{\theta}^{(t,k)}}[f_i].$$

The trade-offs between the two algorithms are now more subtle than in the case of Bayesian networks. For the joint gradient ascent procedure of the previous section, we need to run inference $M + 1$ times in each gradient step: once without evidence, and once for each data case. If we use EM, we run inference M times to compute the expected sufficient statistics in the E-step, and then once for each gradient step, to compute the second term in the gradient. Clearly, there is a computational savings here. However, each of these gradient steps now uses an "out-of-date" set of expected sufficient statistics, making it increasingly less relevant as our optimization proceeds.

In fact, we can view the EM algorithm, in this case, as a form of caching of the first term in the derivative: Rather than compute the expected counts in each iteration, we compute them every few iterations, take a number of gradient steps, and then recompute the expected counts. There is no need to run the "inner-loop" optimization until convergence; indeed, that strategy is often not optimal in practice.

20.3.4 Maximum Entropy and Maximum Likelihood ⋆

We now return to the case of basic maximum likelihood estimation, in order to derive an alternative formulation that provides significant insight. In particular, we now use theorem 20.1 to relate maximum likelihood estimation in log-linear models to another important class or problems examined in statistics: the problem of finding the distribution of maximum entropy subject to a set of constraints.

To motivate this alternative formulation, consider a situation where we are given some summary statistics of an empirical distribution, such as those that may be published in a census report. These statistics may include the marginal distributions of single variables, of certain pairs, and perhaps of other events that the researcher summarizing the data happened to consider of interest. As another example, we might know the average final grade of students in the class and the correlation of their final grade with their homework scores. However, we do not have access to the full data set. While these two numbers constrain the space of possible distributions over the domain, they do not specify it uniquely. Nevertheless, we might want to construct a "typical" distribution that satisfies the constraints and use it to answer other queries.

One compelling intuition is that we should select a distribution that satisfies the given constraints but has no additional "structure" or "information." There are many ways of making this intuition precise. One that has received quite a bit of attention is based on the intuition that entropy is the inverse of information, so that we should search for the distribution of highest entropy. (There are more formal justifications for this intuition, but these are beyond the scope of this book.) More formally, in *maximum entropy* estimation, we solve the following problem:

maximum entropy

Maximum-Entropy:

Find	$Q(\mathcal{X})$
maximizing	$\boldsymbol{H}_Q(\mathcal{X})$
subject to	

$$\boldsymbol{E}_Q[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k. \tag{20.10}$$

expectation constraints

The constraints of equation (20.10) are called *expectation constraints*, since they constrain us to the set of distributions that have a particular set of expectations. We know that this set is

non-empty, since we have one example of a distribution that satisfies these constraints — the empirical distribution.

Somewhat surprisingly, the solution to this problem is a Gibbs distribution over the features $\mathcal{F}$ that matches the given expectations.

Theorem 20.2 *The distribution Q^* is the maximum entropy distribution satisfying equation (20.10) if and only if $Q^* = P_{\hat{\boldsymbol{\theta}}}$, where*

$$P_{\hat{\boldsymbol{\theta}}}(\mathcal{X}) = \frac{1}{Z(\hat{\boldsymbol{\theta}})} \exp\left\{\sum_i \hat{\theta}_i f_i(\mathcal{X})\right\}$$

and $\hat{\boldsymbol{\theta}}$ is the maximum likelihood parameterization relative to $\mathcal{D}$.

Proof For notational simplicity, let $P = P_{\hat{\boldsymbol{\theta}}}$. From theorem 20.1, it follows that $\boldsymbol{E}_P[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})]$ for $i = 1, \ldots, k$, and hence that P satisfies the constraints of equation (20.10). Therefore, to prove that $P = Q^*$, we need only show that $\boldsymbol{H}_P(\mathcal{X}) \geq \boldsymbol{H}_Q(\mathcal{X})$ for all other distributions Q that satisfy these constraints. Consider any such distribution Q.

From proposition 8.1, it follows that:

$$\boldsymbol{H}_P(\mathcal{X}) = -\sum_i \hat{\theta}_i \boldsymbol{E}_P[f_i] + \ln Z(\boldsymbol{\theta}). \tag{20.11}$$

Thus,

$$\begin{aligned}
\boldsymbol{H}_P(\mathcal{X}) - \boldsymbol{H}_Q(\mathcal{X}) &= -\left[\sum_i \hat{\theta}_i \boldsymbol{E}_P[f_i(\mathcal{X})]\right] + \ln Z_P - \boldsymbol{E}_Q[-\ln Q(\mathcal{X})] \\
(i) &= -\left[\sum_i \hat{\theta}_i \boldsymbol{E}_Q[f_i(\mathcal{X})]\right] + \ln Z_P + \boldsymbol{E}_Q[\ln Q(\mathcal{X})] \\
&= \boldsymbol{E}_Q[-\ln P(\mathcal{X})] + \boldsymbol{E}_Q[\ln Q(\mathcal{X})] \\
&= \boldsymbol{D}(Q \| P) \geq 0,
\end{aligned}$$

where (i) follows from the fact that both $P_{\hat{\boldsymbol{\theta}}}$ and Q satisfy the constraints, so that $\boldsymbol{E}_{P_{\hat{\boldsymbol{\theta}}}}[f_i] = \boldsymbol{E}_Q[f_i]$ for all i.

We conclude that $\boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) \geq \boldsymbol{H}_Q(\mathcal{X})$ with equality if and only if $P_{\hat{\boldsymbol{\theta}}} = Q$. Thus, the maximum entropy distribution Q^* is necessarily equal to $P_{\hat{\boldsymbol{\theta}}}$, proving the result. ■

duality

One can also provide an alternative proof of this result based on the concept of *duality* discussed in appendix A.5.4. Using this alternative derivation, one can show that the two problems, maximizing the entropy given expectation constraints and maximizing the likelihood given structural constraints on the distribution, are *convex duals* of each other. (See exercise 20.5.)

Both derivations show that these objective functions provide bounds on each other, and are identical at their convergence point. That is, for the maximum likelihood parameters $\hat{\boldsymbol{\theta}}$,

$$\boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) = -\frac{1}{M}\ell(\hat{\boldsymbol{\theta}} : \mathcal{D}).$$

As a consequence, we see that for any set of parameters $\boldsymbol{\theta}$ and for any distribution Q that satisfy the expectation constraints equation (20.10), we have that

$$\boldsymbol{H}_Q(\mathcal{X}) \leq \boldsymbol{H}_{P_{\hat{\boldsymbol{\theta}}}}(\mathcal{X}) = -\frac{1}{M}\ell(\hat{\boldsymbol{\theta}} : \mathcal{D}) \leq -\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D})$$

with equality if and only if $Q = \mathcal{P}_{\boldsymbol{\theta}}$. We note that, while we provided a proof for this result from first principles, it also follows directly from the theory of convex duality.

Our discussion has shown an entropy dual only for likelihood. A similar connection can be shown between conditional likelihood and conditional entropy; see exercise 20.6.

20.4 Parameter Priors and Regularization

So far, we have focused on maximum likelihood estimation for selecting parameters in a Markov network. However, as we discussed in chapter 17, maximum likelihood estimation (MLE) is prone to overfitting to the training data. Although the effects are not as transparent in this case (due to the lack of direct correspondence between empirical counts and parameters), overfitting of the maximum likelihood estimator is as much of a problem here.

As for Bayesian networks, we can reduce the effect of overfitting by introducing a prior distribution $P(\boldsymbol{\theta})$ over the model parameters. Note that, because we do not have a decomposable closed form for the likelihood function, we do not obtain a decomposable closed form for the posterior in this case. Thus, a fully Bayesian approach, where we integrate out the parameters to compute the next prediction, is not generally feasible in Markov networks. However, we can aim to perform *MAP estimation* — to find the parameters that maximize $P(\boldsymbol{\theta})P(\mathcal{D} \mid \boldsymbol{\theta})$.

MAP estimation

Given that we have no constraints on the conjugacy of the prior and the likelihood, we can consider virtually any reasonable distribution as a possible prior. However, only a few priors have been applied in practice.

20.4.1 Local Priors

Most commonly used is a Gaussian prior on the log-linear parameters $\boldsymbol{\theta}$. The most standard form of this prior is simply a zero-mean diagonal Gaussian, usually with equal variances for each of the weights:

$$P(\boldsymbol{\theta} \mid \sigma^2) = \prod_{i=1}^{k} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{\theta_i^2}{2\sigma^2}\right\},$$

hyperparameter

for some choice of the variance σ^2. This variance is a *hyperparameter*, as were the α_i's in the Dirichlet distribution (section 17.3.2). Converting to log-space (in which the optimization is typically done), this prior gives rise to a term of the form:

$$-\frac{1}{2\sigma^2}\sum_{i=1}^{k}\theta_i^2,$$

This term places a quadratic penalty on the magnitude of the weights, where the penalty is measured in Euclidean, or L_2-norm, generally called an *L_2-regularization* term. This term is

L_2-regularization

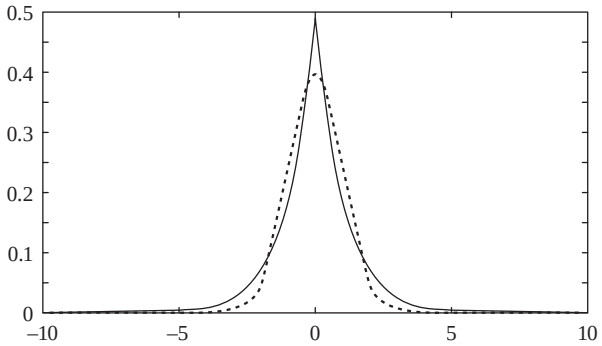


Figure 20.3 Laplacian distribution ($\beta = 1$) and Gaussian distribution ($\sigma^2 = 1$)

concave, and therefore it gives rise to a concave objective, which can be optimized using the same set of methods as standard MLE.

Laplacian distribution

A different prior that has been used in practice uses the zero-mean *Laplacian distribution*, which, for a single parameter, has the form

$$P_{Laplacian}(\theta \mid \beta) = \frac{1}{2\beta} \exp\left\{-\frac{|\theta|}{\beta}\right\}. \tag{20.12}$$

One example of the Laplacian distribution is shown in figure 20.3; it has a nondifferentiable point at $\theta = 0$, arising from the use of the absolute value in the exponent. As for the Gaussian case, one generally assumes that the different parameters θ_i are independent, and often (but not always) that they are identically distributed with the same hyperparameter β. Taking the logarithm, we obtain a term

$$-\frac{1}{\beta}\sum_{i=1}^{k}|\theta_i|$$

L_1-regularization

that also penalizes weights of high magnitude, measured using the L_1-norm. Thus, this approach is generally called *L_1-regularization*.

Both forms of regularization penalize parameters whose magnitude (positive or negative) is large. Why is a bias in favor of parameters of low magnitude a reasonable one? Recall from our discussion in section 17.3 that a prior often serves to pull the distribution toward an "uninformed" one, smoothing out fluctuations in the data. Intuitively, a distribution is "smooth" if the probabilities assigned to different assignments are not radically different. Consider two assignments ξ and ξ'; their relative probability is

$$\frac{P(\xi)}{P(\xi')} = \frac{\tilde{P}(\xi)/Z_{\boldsymbol{\theta}}}{\tilde{P}(\xi')/Z_{\boldsymbol{\theta}}} = \frac{\tilde{P}(\xi)}{\tilde{P}(\xi')}.$$

Moving to log-space and expanding the unnormalized measure $\tilde{P}$, we obtain:

$$\begin{aligned}\ln \frac{P(\xi)}{P(\xi')} &= \sum_{i=1}^{k} \theta_i f_i(\xi) - \sum_{i=1}^{k} \theta_i f_i(\xi') \\ &= \sum_{i=1}^{k} \theta_i (f_i(\xi) - f_i(\xi')).\end{aligned}$$

When all of the θ_i's have small magnitude, this log-ratio is also bounded, resulting in a smooth distribution. Conversely, when the parameters can be large, we can obtain a "spiky" distribution with arbitrarily large differences between the probabilities of different assignments.

In both the L_2 and the L_1 case, we penalize the magnitude of the parameters. In the Gaussian case, the penalty grows quadratically with the parameter magnitude, implying that an increase in magnitude in a large parameter is penalized more than a similar increase in a small parameter. For example, an increase in θ_i from 0 to 0.1 is penalized less than an increase from 3 to 3.1. In the Laplacian case, the penalty is linear in the parameter magnitude, so that the penalty growth is invariant over the entire range of parameter values. This property has important ramifications. In the quadratic case, as the parameters get close to 0, the effect of the penalty diminishes. Hence, the models that optimize the penalized likelihood tend to have many small weights. Although the resulting models are smooth, as desired, they are structurally quite dense. By comparison, in the L_1 case, the penalty is linear all the way until the parameter value is 0. This penalty provides a continued incentive for parameters to shrink until they actually hit 0. As a consequence, **the models learned with an L_1 penalty tend to be much sparser than those learned with an L_2 penalty, with many parameter weights achieving a value of 0. From a structural perspective, this effect gives rise to models with fewer edges and sparser potentials, which are potentially much more tractable.** We return to this issue in section 20.7.

Importantly, both the L_1 and L_2 regularization terms are concave. Because the log-likelihood is also concave, the resulting posterior is concave, and can therefore be optimized efficiently using the gradient-based methods we described for the likelihood case. Moreover, the introduction of these penalty terms serves to reduce or even eliminate multiple (equivalent) optima that arise when the parameterization of the network is redundant. For example, consider the trivial example where we have no data. In this case, the maximum likelihood solution is (as desired) the uniform distribution. However, due to redundancy, there is a continuum of parameterizations that give rise to the uniform distribution. However, when we introduce either of the earlier prior distributions, the penalty term drives the parameters toward zero, giving rise to the unique optimum $\boldsymbol{\theta} = \mathbf{0}$. Although one can still construct examples where multiple optima occur, they are very rare in practice. Conversely, methods that eliminate redundancies by reexpressing some of the parameters in terms of others can produce undesirable interactions with the regularization terms, giving rise to priors where some parameters are penalized more than others.

The regularization hyperparameters — σ^2 in the L_2 case, and β in the L_1 case — encode the strength in our belief that the model weights should be close to 0. The larger these parameters (both in the denominator), the broader our parameter prior, and the less strong our bias toward 0. In principle, any choice of hyperparameter is legitimate, since a prior is simply a reflection of our beliefs. **In practice, however, the choice of prior can have a significant effect on the quality of our learned model. A standard method for selecting this parameter is via a**

cross-validation procedure, as described in box 16.A: We repeatedly partition the training set, learn a model over one part with some choice of hyperparameter, and measure the performance of the learned model (for example, log-likelihood) on the held-out fragment.

20.4.2 Global Priors

conjugate prior

An alternative approach for defining priors is to search for a *conjugate prior*. Examining the likelihood function, we see that the posterior over parameters has the following general form:

$$\begin{aligned} P(\boldsymbol{\theta} \mid \mathcal{D}) &\propto P(\boldsymbol{\theta}) P(\mathcal{D} \mid \boldsymbol{\theta}) \\ &= P(\boldsymbol{\theta}) \exp\left\{ \sum_i M \boldsymbol{E}_{\mathcal{D}}[f_i] \theta_i - M \ln Z(\boldsymbol{\theta}) \right\}. \end{aligned}$$

This expression suggests that we use a family of prior distributions of the form:

$$P(\boldsymbol{\theta}) \propto \exp\left\{ \sum_i M_0 \alpha_i \theta_i - M_0 \ln Z(\boldsymbol{\theta}) \right\}.$$

This form defines a family of priors with hyperparameters $\{\alpha_i\}$. It is easy to see that the posterior is from the same family with $\alpha'_i = \alpha_i + \boldsymbol{E}_{\mathcal{D}}[f_i]$ and $M'_0 = M_0 + M$, so that this prior is conjugate to the log-linear model likelihood function.

We can think of the hyperparameters $\{\alpha_i\}$ as specifying the sufficient statistics from prior observations and of M_0 as specifying the number of these prior observations. This formulation is quite similar to the use of pseudocounts in the BDe priors for directed models (see section 17.4.3). The main difference from directed models is that this conjugate family (both the prior and the likelihood) does not decompose into independent priors for the different features.

20.5 Learning with Approximate Inference

The methods we have discussed here assume that we are able to compute the partition function $Z(\boldsymbol{\theta})$ and expectations such as $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_i]$. In many real-life applications the structure of the network does not allow for exact computation of these terms. For example, in applications to image segmentation (box 4.B), we generally use a grid-structured network, which requires exponential size clusters for exact inference.

The simplest approach for learning in intractable networks is to apply the learning procedure (say, conjugate gradient ascent) using an approximate inference procedure to compute the required queries about the distribution $P_{\boldsymbol{\theta}}$. This view decouples the question of inference from learning and treats the inference procedure as a black box during learning. The success of such an approach depends on whether the approximation method interferes with the learning. In particular, **nonconvergence of the inference method, or convergence to approximate answers, can lead to inaccurate and even oscillating estimates of the gradient, potentially harming convergence of the overall learning algorithm.** This type of situation can arise both in particle-based methods (say MCMC sampling) and in global algorithms such as belief propagation. In this section, we describe several methods that better integrate the inference into the learning outer loop in order to reduce problems such as this.

A second approach for dealing with inference-induced costs is to come up with alternative (possibly approximate) objective functions whose optimization does not require (as much) inference. Some of these techniques are reviewed in the next section. However, one of the main messages of this section is that the boundary between these two classes of methods is surprisingly ambiguous. **Approximately optimizing the likelihood objective by using an approximate inference algorithm to compute the gradient can often be reformulated as exactly optimizing an approximate objective.** When applicable, this view is often more insightful and also more usable. First, it provides more insight about the outcome of the optimization. Second, it may allow us to bound the error in the optimum in terms of the distance between the two functions being optimized. Finally, by formulating a clear objective to be optimized, we can apply any applicable optimization algorithm, such as conjugate gradient or Newton's method.

Importantly, while we describe the methods in this section relative to the plain likelihood objective, they apply almost without change to the generalizations and extensions we describe in this chapter: conditional Markov networks; parameter priors and regularization; structure learning; and learning with missing data.

20.5.1 Belief Propagation

belief propagation

A fairly popular approach for approximate inference is the *belief propagation* algorithm and its variants. Indeed, in many cases, an algorithm in this family would be used for inference in the model resulting from the learning procedure. In this case, it can be shown that **we should learn the model using the same inference algorithm that will be used for querying it. Indeed, it can be shown that using a model trained with the same approximate inference algorithm is better than using a model trained with exact inference.**

At first glance, the use of belief propgation for learning appears straightforward. We can simply run BP within every iteration of gradient ascent to compute the expected feature counts used in the gradient computation. Due to the family preservation property, each feature f_i must be a subset of a cluster $\boldsymbol{C}_i$ in the cluster graph. Hence, to compute the expected feature count $\boldsymbol{E}_{\boldsymbol{\theta}}[f_i]$, we can compute the BP marginals over $\boldsymbol{C}_i$, and then compute the expectation. In practice, however, this approach can be highly problematic. As we have seen, BP often does not converge. The marginals that we derive from the algorithm therefore oscillate, and the final results depend on the point at which we choose to stop the algorithm. As a result, the gradient computed from these expected counts is also *unstable*. This instability can be a significant problem in a gradient-based procedure, since it can gravely hurt the convergence properties of the algorithm. This problem is even more severe in the context of line-search methods, where the function evaluations can be inconsistent at different points in the line search.

unstable gradient

There are several solutions to this problem: One can use one of the convergent alternatives to the BP algorithm that still optimizes the same Bethe energy objective; one can use a convex energy approximation, such as those of section 11.3.7.2; or, as we now show, one can reformulate the task of learning with approximate inference as optimizing an alternative objective, allowing the use of a range of optimization methods with better convergence properties.

20.5.1.1 Pseudo-moment Matching

Let us begin by a simple analysis of the fixed points of the learning algorithm. At convergence, the approximate expectations must satisfy the condition of theorem 20.1; in particular, the converged BP beliefs for $\boldsymbol{C}_i$ must satisfy

$$\boldsymbol{E}_{\beta_i(\boldsymbol{C}_i)}[f_{\boldsymbol{C}_i}] = \boldsymbol{E}_{\mathcal{D}}[f_i(\boldsymbol{C}_i)].$$

Now, let us consider the special case where our feature model defines a set of fully parameterized potentials that precisely match the clusters used in the BP cluster graph. That is, for every cluster $\boldsymbol{C}_i$ in the cluster graph, and every assignment $\boldsymbol{c}_i^j$ to $\boldsymbol{C}_i$, we have a feature which is an indicator function $\boldsymbol{I}\{\boldsymbol{c}_i^j\}$, that is, it is 1 when $\boldsymbol{C}_i = \boldsymbol{c}_i^j$ and 0 otherwise. In this case, the preceding set of equalities imply that, for every assignment $\boldsymbol{c}_i^j$ to $\boldsymbol{C}_i$, we have that

$$\beta_i(\boldsymbol{c}_i^j) = \hat{P}(\boldsymbol{c}_i^j). \tag{20.13}$$

That is, at convergence of the gradient ascent algorithm, the convergence point of the underlying belief propagation must be to a set of beliefs that exactly matches the empirical marginals in the data. But if we already know the outcome of our convergence, there is no point to running the algorithm!

This derivation gives us a closed form for the BP potentials at the point when both algorithms — BP inference and parameter gradient ascent — have converged. As we have already discussed, the full-table parameterization of Markov network potentials is redundant, and therefore there are multiple solutions that can give rise to this set of beliefs. One of these solutions can be obtained by dividing each sepset in the calibrated cluster graph into one of the adjacent clique potentials. More precisely, for each sepset $\boldsymbol{S}_{i,j}$ between $\boldsymbol{C}_i$ and $\boldsymbol{C}_j$, we select the endpoint for which $i < j$ (in some arbitrary ordering), and we then define:

$$\phi_i \leftarrow \frac{\beta_i}{\mu_{i,j}}.$$

We perform this transformation for each sepset. We use the final set of potentials as the parameterization for our Markov network. We can show that a single pass of message passing in a particular order gives rise to a calibrated cluster graph whose potentials are precisely the ones in equation (20.13). Thus, in this particular special case, we can provide a closed-form solution to both the inference and learning problem. This approach is called *pseudo-moment matching*.

pseudo-moment matching

While it is satisfying that we can find a solution so effectively, the form of the solution should be considered with care. In particular, we note that the clique potentials are simply empirical cluster marginals divided by empirical sepset marginals. These quantities depend only on the local structure of the factor and not on any global aspect of the cluster graph, including its structure. For example, the BC factor is estimated in exactly the same way within the diamond network of figure 11.1a and within the chain network A—B—C—D. Of course, potentials are also estimated locally in a Bayesian network, but there the local calibration ensures that the distribution can be factorized using purely local computations. As we have already seen, this is not the case for Markov networks, and so we expect different potentials to adjust to fit each other; however, the estimation using loopy BP does not accommodate that. In a sense, this observation is not surprising, since the BP approach also ignores the more global information.

We note, however, that this purely local estimation of the parameters only holds under the very restrictive conditions described earlier. It does not hold when we have parameter priors (regularization), general features rather than table factors, any type of shared parameters (as in section 6.5), or conditional random fields. We discuss this more general case in the next section.

20.5.1.2 Belief Propagation and Entropy Approximations ⋆

We now provide a more general derivation that allows us to reformulate maximum-likelihood learning with belief propagation as a unified optimization problem with an approximate objective. This perspective opens the door to the use of better approximation algorithms.

maximum entropy

Our analysis starts from the *maximum-entropy* dual of the maximum-likelihood problem.

Maximum-Entropy:

Find $Q(\mathcal{X})$
maximizing $\boldsymbol{H}_Q(\mathcal{X})$
subject to

$$\boldsymbol{E}_Q[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k.$$

We can obtain a tractable approximation to this problem by applying the same sequence of transformations that we used in section 11.3.6 to derive belief propagation from the energy optimization problem. More precisely, assume we have a cluster graph $\mathcal{U}$ consisting of a set of clusters $\{\boldsymbol{C}_i\}$ connected by sepsets $\boldsymbol{S}_{i,j}$. Now, rather than optimize Maximum-Entropy over the space of distributions Q, we optimize over the set of possible pseudo-marginals in the *local consistency polytope Local*$[\mathcal{U}]$, as defined in equation (11.16). Continuing as in the BP derivation, we also approximate the entropy as in its *factored* form (definition 11.1):

local consistency polytope

factored entropy

$$\boldsymbol{H}_Q(\mathcal{X}) \approx \sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{U}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j}). \tag{20.14}$$

As before, this reformulation is exact when the cluster graph is a tree but is approximate otherwise.

Putting these approximations together, we obtain the following approximation to the maximum-entropy optimization problem:

Approx-Maximum-Entropy:

Find $\boldsymbol{Q}$
maximizing $\sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i) - \sum_{(\boldsymbol{C}_i — \boldsymbol{C}_j) \in \mathcal{U}} \boldsymbol{H}_{\mu_{i,j}}(\boldsymbol{S}_{i,j})$
subject to

$$\begin{aligned} \boldsymbol{E}_{\beta_i}[f_i] &= \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k \\ \boldsymbol{Q} &\in \mathit{Local}[\mathcal{U}]. \end{aligned} \tag{20.15}$$

CAMEL

This approach is called *CAMEL*, for constrained approximate maximum enropy learning.

Example 20.2

To illustrate this reformulation, consider a simple pairwise Markov network over the binary variables A, B, C, with three clusters: $\boldsymbol{C}_1 = \{A, B\}, \boldsymbol{C}_2 = \{B, C\}, \boldsymbol{C}_3 = \{A, C\}$. We assume that the log-linear model is defined by the following two features, both of which are shared over all clusters: $f_{00}(x, y) = 1$ if $x = 0$ and $y = 0$, and 0 otherwise; and $f_{11}(x, y) = 1$ if $x = 1$ and $y = 1$. Assume we have 3 data instances $[0, 0, 0]$, $[0, 1, 0]$, $[1, 0, 0]$. The unnormalized empirical counts of each feature, pooled over all clusters, is then $\boldsymbol{E}_{\hat{P}}[f_{00}] = (3 + 1 + 1)/3 = 5/3$, $\boldsymbol{E}_{\hat{P}}[f_{11}] = 0$. In this case, the optimization of equation (20.15) would take the following form:

$$
\begin{aligned}
\textbf{Find} \quad & \boldsymbol{Q} = \{\beta_1, \beta_2, \beta_3, \mu_{1,2}, \mu_{2,3}, \mu_{1,3}\} \\
\textbf{maximizing} \quad & \boldsymbol{H}_{\beta_1}(A, B) + \boldsymbol{H}_{\beta_2}(B, C) + \boldsymbol{H}_{\beta_3}(A, C) \\
& -\boldsymbol{H}_{\mu_{1,2}}(B) - \boldsymbol{H}_{\mu_{2,3}}(C) - \boldsymbol{H}_{\mu_{1,3}}(A) \\
\textbf{subject to} \quad & \\
& \sum_i \boldsymbol{E}_{\beta_i}[f_{00}] = 5/3 \\
& \sum_i \boldsymbol{E}_{\beta_i}[f_{11}] = 0 \\
& \sum_a \beta_1(a, b) - \sum_c \beta_2(b, c) = 0 \quad \forall b \\
& \sum_b \beta_2(b, c) - \sum_a \beta_3(a, c) = 0 \quad \forall c \\
& \sum_c \beta_3(a, c) - \sum_b \beta_1(a, b) = 0 \quad \forall a \\
& \sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) = 1 \quad i = 1, 2, 3 \\
& \beta_i \geq 0 \quad i = 1, 2, 3.
\end{aligned}
$$

The CAMEL optimization problem of equation (20.15) is a constrained maximization problem with linear constraints and a nonconcave objective. The problem actually has two distinct sets of constraints: the first set encodes the moment-matching constraints and comes from the learning problem; and the second set encodes the constraint that $\boldsymbol{Q}$ be in the marginal polytope and arises from the cluster-graph approximation. It thus forms a unified optimization problem that encompasses both the learning task — moment matching — and the inference task — obtaining a set of consistent pseudo-marginals over a cluster graph. Analogously, if we introduce Lagrange multipliers for these constraints (as in appendix A.5.3), they would have very different interpretations. The multipliers for the first set of constraints would correspond to weights $\boldsymbol{\theta}$ in the log-linear model, as in the max-likelihood / max-entropy duality ; those in the second set would correspond to messages $\delta_{i \to j}$ in the cluster graph, as in the BP algorithm.

This observation leads to several solution algorithms for this problem. In one class of methods, we could introduce Lagrange multipliers for all of the constraints and then optimize the resulting problem over these new variables. If we perform the optimization by a double-loop algorithm where the outer loop optimizes over $\boldsymbol{\theta}$ (say using gradient ascent) and the inner loops "optimizes" the $\delta_{i \to j}$ by iterating their fixed point equations, the result would be precisely gradient ascent over parameters with BP in the inner loop for inference.

20.5.1.3 Sampling-Based Learning ⋆

The partition function $Z(\theta)$ is a summation over an exponentially large space. One approach to approximating this summation is to reformulate it as an expectation with respect to some distribution $Q(\mathcal{X})$:

$$\begin{aligned} Z(\boldsymbol{\theta}) &= \sum_{\xi} \exp\left\{\sum_i \theta_i f_i(\xi)\right\} \\ &= \sum_{\xi} \frac{Q(\xi)}{Q(\xi)} \exp\left\{\sum_i \theta_i f_i(\xi)\right\} \\ &= \boldsymbol{E}_Q\left[\frac{1}{Q(\mathcal{X})} \exp\left\{\sum_i \theta_i f_i(\mathcal{X})\right\}\right]. \end{aligned}$$

importance sampling

This is precisely the form of the *importance sampling* estimator described in section 12.2.2. Thus, we can approximate it by generating samples from Q, and correcting appropriately via weights. We can simplify this expression if we choose Q to be $P_{\boldsymbol{\theta}^0}$ for some set of parameters $\boldsymbol{\theta}^0$:

$$\begin{aligned} Z(\boldsymbol{\theta}) &= \boldsymbol{E}_{P_{\boldsymbol{\theta}^0}}\left[\frac{Z(\boldsymbol{\theta}^0)\exp\{\sum_i \theta_i f_i(\mathcal{X})\}}{\exp\{\sum_i \theta_i^0 f_i(\mathcal{X})\}}\right] \\ &= Z(\boldsymbol{\theta}^0)\boldsymbol{E}_{P_{\boldsymbol{\theta}^0}}\left[\exp\left\{\sum_i (\theta_i - \theta_i^0) f_i(\mathcal{X})\right\}\right]. \end{aligned}$$

If we can sample instances $\xi^1, \ldots, \xi^K$ from $P_{\boldsymbol{\theta}_0}$, we can approximate the log-partition function as:

$$\ln Z(\boldsymbol{\theta}) \approx \ln\left(\frac{1}{K}\sum_{k=1}^{K} \exp\left\{\sum_i (\theta_i - \theta_i^0) f_i(\xi^k)\right\}\right) + \ln Z(\boldsymbol{\theta}^0). \tag{20.16}$$

We can plug this approximation of $\ln Z(\boldsymbol{\theta})$ into the log-likelihood of equation (20.3) and optimize it. Note that $\ln Z(\boldsymbol{\theta}^0)$ is a constant that we can ignore in the optimization, and the resulting expression is therefore a simple function of $\boldsymbol{\theta}$, which can be optimized using methods such as gradient ascent or one of its extensions. Interestingly, gradient ascent over $\boldsymbol{\theta}$ relative to equation (20.16) is equivalent to utilizing an importance sampling estimator directly to approximate the expected counts in the gradient of equation (20.4) (see exercise 20.12). However, as we discussed, it is generally more instructive and useful to view such methods as exactly optimizing an approximate objective rather than approximately optimizing the exact likelihood.

Of course, as we discussed in section 12.2.2, the quality of an importance sampling estimator depends on the difference between $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^0$: the greater the difference, the larger the variance of the importance weights. Thus, this type of approximation is reasonable only in a neighborhood surrounding $\boldsymbol{\theta}^0$.

MCMC

How do we use this approximation? One possible strategy is to iterate between two steps. In one we run a sampling procedure, such as *MCMC*, to generate samples from the current parameter set $\boldsymbol{\theta}^t$. Then in the second iteration we use some gradient procedure to find $\boldsymbol{\theta}^{t+1}$

that improve the approximate log-likelihood based on these samples. We can then regenerate samples and repeat the process. As the samples are regenerated from a new distribution, we can hope that they are generated from a distribution not too far from the one we are currently optimizing, maintaining a reasonable approximation.

20.5.2 MAP-Based Learning ⋆

MAP assignment

As another approximation to the inference step in the learning algorithm, we can consider approximating the expected feature counts with their counts in the single *MAP assignment* to the current Markov network. As we discussed in chapter 13, in many classes of models, computing a single MAP assignment is a much easier computational task, making this a very appealing approach in many settings.

More precisely, to approximate the gradient at a given parameter assignment $\boldsymbol{\theta}$, we compute

$$\boldsymbol{E}_{\mathcal{D}}[f_i(\mathcal{X})] - f_i(\xi^{\text{MAP}}(\boldsymbol{\theta})), \tag{20.17}$$

Viterbi training

where $\xi^{\text{MAP}}(\boldsymbol{\theta}) = \arg\max_{\xi} P(\xi \mid \boldsymbol{\theta})$ is the MAP assignment given the current set of parameters $\boldsymbol{\theta}$. This approach is also called *Viterbi training*.

Once again, we can gain considerable intuition by reformulating this approximate inference step as an exact optimization of an approximate objective. Some straightforward algebra shows that this gradient corresponds exactly to the approximate objective

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}), \tag{20.18}$$

or, due to the cancellation of the partition function:

$$\frac{1}{M}\sum_{m=1}^{M} \ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}). \tag{20.19}$$

To see this, consider a single data instance $\xi[m]$:

$$\begin{aligned}
\ln P(\xi[m] \mid \boldsymbol{\theta}) - \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) \\
&= [\ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta})] - [\ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta})] \\
&= \ln \tilde{P}(\xi[m] \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) \\
&= \sum_i \theta_i [f_i(\xi[m]) - f_i(\xi^{\text{MAP}}(\boldsymbol{\theta}))].
\end{aligned}$$

If we average this expression over all data instances and take the partial derivative relative to θ_i, we obtain an expression whose gradient is precisely equation (20.17).

The first term in equation (20.19) is an average of expressions of the form $\ln \tilde{P}(\xi \mid \boldsymbol{\theta})$. Each such expression is a linear function in $\boldsymbol{\theta}$, and hence their average is also linear in $\boldsymbol{\theta}$. The second term, $\tilde{P}(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta})$, may appear to be the log-probability of an instance. However, as indicated by the notation, $\xi^{\text{MAP}}(\boldsymbol{\theta})$ is itself a function of $\boldsymbol{\theta}$: in different regions of the parameter space, the MAP assignment changes. In fact, this term is equal to:

$$\ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta}) = \max_{\xi} \ln P(\xi \mid \boldsymbol{\theta}).$$

This is a maximum of linear functions, which is a convex, piecewise-linear function. Therefore, its negation is concave, and so the entire objective of equation (20.19) is also concave and hence has a global optimum.

Although reasonable at first glance, a closer examination reveals some important issues with this objective. Consider again a single data instance $\xi[m]$. Because $\xi^{\text{MAP}}(\boldsymbol{\theta})$ is the MAP assignment, it follows that $\ln P(\xi[m] \mid \boldsymbol{\theta}) \leq \ln P(\xi^{\text{MAP}}(\boldsymbol{\theta}) \mid \boldsymbol{\theta})$, and therefore the objective is always nonpositive. The maximal value of 0 can be achieved in two ways. The first is if we manage to find a setting of $\boldsymbol{\theta}$ in which the empirical feature counts match the feature counts in $\xi^{\text{MAP}}(\boldsymbol{\theta})$. This optimum may be hard to achieve: Because the counts in $\xi^{\text{MAP}}(\boldsymbol{\theta})$ are discrete, they take on only a finite set of values; for example, if we have a feature that is an indicator function for the event $X_i = x_i$, its count can take on only the values 0 or 1, depending on whether the MAP assignment has $X_i = x_i$ or not. Thus, we may never be able to match the feature counts exactly. The second way of achieving the optimal value of 0 is to set all of the parameters θ_i to 0. In this case, we obtain the uniform distribution over assignments, and the objective achieves its maximum value of 0. This possible behavior may not be obvious when we consider the gradient, but it becomes apparent when we consider the objective we are trying to optimize.

That said, we note that in the early stages of the optimization, when the expected counts are far from the MAP counts, the gradient still makes progress in the general direction of increasing the relative log-probability of the data instances. This approach can therefore work fairly well in practice, especially if not optimized to convergence.

---

protein structure

Box 20.B — Case Study: CRFs for Protein Structure Prediction. *One interesting application of CRFs is to the task of predicting the three-dimensional* structure *of proteins. Proteins are constructed as chains of residues, each containing one of twenty possible amino acids. The amino acids are linked together into a common backbone structure onto which amino-specific side-chains are attached. An important computational problem is that of predicting the side-chain conformations given the backbone. The full configuration for a side-chain consists of up to four angles, each of which takes on a continuous value. However, in practice, angles tend to cluster into bins of very similar angles, so that the common practice is to discretize the value space of each angle into a small number (usually up to three) bins, called* rotamers.

With this transformation, side-chain prediction can be formulated as a discrete optimization problem, where the objective is an energy over this discrete set of possible side-chain conformations. Several energy functions have been proposed, all of which include various repulsive and attractive terms between the side-chain angles of nearby residues, as well as terms that represent a prior and internal constraints within the side chain for an individual residue. Rosetta, a state-of-the-art system, uses a combination of eight energy terms, and uses simulated annealing to search for the minimal energy configuration. However, even this highly engineered system still achieves only moderate accuracies (around 72 percent of the discretized angles predicted correctly). An obvious question is whether the errors are due to suboptimal answers returned by the optimization algorithm, or to the design of the energy function, which may not correctly capture the true energy "preferences" of protein structures.

Yanover, Schueler-Furman, and Weiss (2007) propose to address this optimization problem using MAP inference techniques. The energy functions used in this type of model can also be viewed as the

log-potentials of a Markov network, where the variables represent the different angles to be inferred, and their values the discretized rotamers. The problem of finding the optimal configuration is then simply the MAP inference problem, and can be tackled using some of the algorithms described in chapter 13. Yanover et al. show that the TRW algorithm of box 13.A finds the provably global optimum of the Rosetta energy function for approximately 85 percent of the proteins in a standard benchmark set; this computation took only a few minutes per protein on a standard workstation. They also tackled the problem by directly solving the LP relaxation of the MAP problem using a commercial LP solver; this approach found the global optimum of the energy function for all proteins in the test set, but at a higher computational cost. However, finding the global minimum gave only negligible improvements on the actual accuracy of the predicted angles, suggesting that the primary source of inaccuracy in these models is in the energy function, not the optimization.

Thus, this problem seems like a natural candidate for the application of learning methods. The task was encoded as a CRF, whose input is a list of amino acids that make up the protein as well as the three-dimensional shape of the backbone. Yanover et al. encoded this distribution as a log-linear model whose features were the (eight) different components of the Rosetta energy function, and whose parameters were the weights of these features. Because exact inference for this model is intractable, it was trained by using a TRW variant for sum-product algorithms (see section 11.3.7.2). This variant uses a set of convex counting numbers to provide a convex approximation, and a lower bound, to the log-partition function. These properties guarantee that the learning process is stable and is continually improving a lower bound on the true objective. This new energy function improves performance from 72 percent to 78 percent, demonstrating that learning can significantly improve models, even those that are carefully engineered and optimized by a human expert. Notably, for the learned energy function, and for other (yet more sophisticated) energy functions, the use of globally optimal inference does lead to improvements in accuracy. Overall, a combination of these techniques gave rise to an accuracy of 82.6 percent, a significant improvement.

20.6 Alternative Objectives

Another class of approximations can be obtained directly by replacing the objective that we aim to optimize with one that is more tractable. To motivate the alternative objectives we present in this chapter, let us consider again the form of the log-likelihood objective, focusing, for simplicity, on the case of a single data instance ξ:

$$\begin{aligned}\ell(\boldsymbol{\theta} : \xi) &= \ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln Z(\boldsymbol{\theta}) \\ &= \ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \left(\sum_{\xi'} \tilde{P}(\xi' \mid \boldsymbol{\theta}) \right).\end{aligned}$$

Considering the first term, this objective aims to increase the log-measure (logarithm of the unnormalized probability) of the observed data instance ξ. Of course, because the log-measure is a linear function of the parameters in our log-linear representation, that goal can be achieved simply by increasing all of the parameters associated with positive empirical expectations in ξ, and decreasing all of the parameters associated with negative empirical expectations. Indeed,

we can increase the first term unboundedly using this approach. The second term, however, balances the first, since it is the logarithm of a sum of the unnormalized measures of instances, in this case, all possible instances in $Val(\mathcal{X})$. In a sense, then, we can view the log-likelihood objective as aiming to increasing the distance between the log-measure of ξ and the aggregate of the measures of all instances. We can thus view it as contrasting two terms. The key difficulty with this formulation, of course, is that the second term involves a summation over the exponentially many instances in $Val(\mathcal{X})$, and therefore requires inference in the network.

This formulation does, however, suggest one approach to approximating this objective: **perhaps we can still move our parameters in the right direction if we aim to increase the difference between the log-measure of the data instances and a *more tractable set* of other instances, one that does not require summation over an exponential space.** The *contrastive objectives* that we describe in this section all take that form.

contrastive objective

20.6.1 Pseudolikelihood and Its Generalizations

Perhaps the earliest method for circumventing the intractability of network inference is the pseudolikelihood objective. As one motivation for this approximation, consider the likelihood of a single instance ξ. Using the chain rule, we can write

$$P(\xi) = \prod_{j=1}^{n} P(x_j \mid x_1, \ldots, x_{j-1}).$$

We can approximate this formulation by replacing each term $P(x_i \mid x_1, \ldots, x_{i-1})$ by the conditional probability of x_i given all other variables:

$$P(\xi) \approx \prod_{j} P(x_j \mid x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n).$$

pseudolikelihood

This approximation leads to the *pseudolikelihood* objective:

$$\ell_{\mathrm{PL}}(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \sum_{m} \sum_{j} \ln P(x_j[m] \mid \boldsymbol{x}_{-j}[m], \boldsymbol{\theta}), \tag{20.20}$$

where $\boldsymbol{x}_{-j}$ stands for $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Intuitively, this objective measures our ability to predict each variable in the model given a full observation over all other variables. The predictive model takes a form that generalizes the *multinomial logistic CPD* of definition 5.10 and is identical to it in the case where the network contains only pairwise features — factors over edges in the network. As usual, we can use the conditional independence properties in the network to simplify this expression, removing from the right-hand side of $P(X_j \mid \boldsymbol{X}_{-j})$ any variable that is not a neighbor of X_j.

multinomial logistic CPD

At first glance, this objective may appear to be more complex than the likelihood objective. However, a closer examination shows that we have eliminated the exponential summation over instances with several summations, each of which is far more tractable. In particular:

$$\begin{aligned} P(x_j \mid \boldsymbol{x}_{-j}) = \frac{P(x_j, \boldsymbol{x}_{-j})}{P(\boldsymbol{x}_{-j})} &= \frac{\tilde{P}(x_j, \boldsymbol{x}_{-j})}{\tilde{P}(\boldsymbol{x}_{-j})} \\ &= \frac{\tilde{P}(x_j, \boldsymbol{x}_{-j})}{\sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j})}. \end{aligned}$$

The critical feature of this expression is that the global partition function has disappeared, and instead we have a local partition function that requires summing only over the values of X_j.

The contrastive perspective that we described earlier provides an alternative insight on this derivation. Consider the pseudolikelihood objective applied to a single data instance ξ:

$$\begin{aligned}\sum_j \ln P(x_j \mid \boldsymbol{x}_{-j}) &= \sum_j \left(\ln \tilde{P}(x_j, \boldsymbol{x}_{-j}) - \ln \sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j}) \right) \\ &= \sum_j \left(\ln \tilde{P}(\xi) - \ln \sum_{x'_j} \tilde{P}(x'_j, \boldsymbol{x}_{-j}) \right).\end{aligned}$$

Each of the terms in this final summation is a contrastive term, where we aim to increase the difference between the log-measure of our training instance ξ and an aggregate of the log-measures of instances that differ from ξ in the assignment to precisely one variable. In other words, we are increasing the contrast between our training instance ξ and the instances in a local neighborhood around it.

We can further simplify each of the summands in this expression, obtaining:

$$\ln P(x_j \mid \boldsymbol{x}_{-j}) = \left(\sum_{i \,:\, \mathit{Scope}[f_i] \ni X_j} \theta_i f_i(x_j, \boldsymbol{x}_{-j}) \right) - \ln \left(\sum_{x'_j} \exp \left\{ \sum_{i \,:\, \mathit{Scope}[f_i] \ni X_j} \theta_i f_i(x'_j, \boldsymbol{x}_{-j}) \right\} \right). \tag{20.21}$$

Each of these terms is precisely a log-conditional-likelihood term for a Markov network over a single variable X_j, conditioned on all the remaining variables. Thus, it follows from corollary 20.2 that the function is concave in the parameters $\boldsymbol{\theta}$. Since a sum of concave functions is also concave, we have that the pseudolikelihood objective of equation (20.20) is concave. Thus, we are guaranteed that gradient ascent over this objective will converge to the global maximum.

To compute the gradient, we use equation (20.21), to obtain:

$$\frac{\partial}{\partial \theta_i} \ln P(x_j \mid \boldsymbol{x}_{-j}) = f_i(x_j, \boldsymbol{x}_{-j}) - \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}}(X_j \mid \boldsymbol{x}_{-j})} \big[f_i(x'_j, \boldsymbol{x}_{-j}) \big]. \tag{20.22}$$

If X_j is not in the scope of f_i, then $f_i(x_j, \boldsymbol{x}_{-j}) = f_i(x'_j, \boldsymbol{x}_{-j})$ for any x'_j, and the two terms are identical, making the derivative 0. Inserting this expression into equation (20.20), we obtain:

Proposition 20.4

$$\frac{\partial}{\partial \theta_i} \ell_{\mathrm{PL}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_{j : X_j \in \mathit{Scope}[f_i]} \left(\frac{1}{M} \sum_m f_i(\xi[m]) - \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}}(X_j \mid \boldsymbol{x}_{-j}[m])} \big[f_i(x'_j, \boldsymbol{x}_{-j}[m]) \big] \right). \tag{20.23}$$

While this term looks somewhat more involved than the gradient of the likelihood in equation (20.4), it is much easier to compute: each of the expectation terms requires a summation

over only a single random variable X_j, conditioned on all of its neighbors, a computation that can generally be performed very efficiently.

What is the relationship between maximum likelihood estimation and maximum pseudo-likelihood? In one specific situation, the two estimators return the same set of parameters.

Theorem 20.3 *Assume that our data are generated by a log-linear model $P_{\boldsymbol{\theta}^*}$ that is of the form of equation (20.1). Then, as the number of data instances M goes to infinity, with probability that approaches 1, $\boldsymbol{\theta}^*$ is a global optimum of the pseudolikelihood objective of equation (20.20).*

PROOF To prove the result, we need to show that because the size of the data set tends to infinity, the gradient of the pseudolikelihood objective at $\boldsymbol{\theta}^*$ tends to zero. Owing to the concavity of the objective, this equality implies that $\boldsymbol{\theta}^*$ is necessarily an optimum of the pseudolikelihood objective. We provide a somewhat informal sketch of the gradient argument, but one that contains all the essential ideas.

Because $M \longrightarrow \infty$, the empirical distribution $\hat{P}$ gets arbitrarily close to $P_{\boldsymbol{\theta}^*}$. Thus, the statistics in the data are precisely representative of their expectations relative to $P_{\boldsymbol{\theta}^*}$. Now, consider one of the summands in equation (20.23), associated with a feature f_i. Due to the convergence of the sufficient statistics,

$$\frac{1}{M} \sum_m f_i(\xi[m]) \longrightarrow \boldsymbol{E}_{\xi \sim P_{\boldsymbol{\theta}^*}(\mathcal{X})}[f_i(\xi)].$$

Conversely,

$$\begin{aligned}
&\frac{1}{M} \sum_m \boldsymbol{E}_{x'_j \sim P_{\boldsymbol{\theta}^*}(X_j \mid \boldsymbol{x}_{-j}[m])} \left[f_i(x'_j, \boldsymbol{x}_{-j}[m]) \right] \\
&\quad = \quad \sum_{\boldsymbol{x}_{-j}} P_{\mathcal{D}}(\boldsymbol{x}_{-j}) \sum_{x'_j} P_{\boldsymbol{\theta}^*}(x'_j \mid \boldsymbol{x}_{-j}) f_i(x'_j, \boldsymbol{x}_{-j}) \\
&\quad \longrightarrow \quad \sum_{\boldsymbol{x}_{-j}} P_{\boldsymbol{\theta}^*}(\boldsymbol{x}_{-j}) \sum_{x'_j} P_{\boldsymbol{\theta}^*}(x'_j \mid \boldsymbol{x}_{-j}) f_i(x'_j, \boldsymbol{x}_{-j}) \\
&\quad = \quad \boldsymbol{E}_{\xi \sim P_{\boldsymbol{\theta}^*}}[f_i(\xi)].
\end{aligned}$$

Thus, at the limit, the empirical and expected counts are equal, so that the gradient is zero. ■

consistent

This theorem states that, like the likelihood objective, the pseudolikelihood objective is also *consistent*. If we assume that the models are nondegenerate so that the two objectives are strongly concave, the maxima are unique, and hence the two objectives have the same maximum.

While this result is an important one, it is important to be cognizant of its limitations. In particular, we note that the two assumptions are central to this argument. First, in order for the empirical and expected counts to match, the model being learned needs to be sufficiently expressive to represent the generating distribution. Second, the data distribution needs to be close enough to the generating distribution to be well captured within the model, a situation that is only guaranteed to happen at the large-sample limit. Without these assumptions, the two objectives can have quite different optima that lead to different results.

In practice, these assumptions rarely hold: our model is never a perfect representation of the true underlying distribution, and we often do not have enough data to be close to the large sample limit. Therefore, one must consider the question of how good this objective is in practice. The answer to this question depends partly on the types of queries for which we intend to use the model.  **If we plan to run queries where we condition on most of the variables and query the values of only a few, the pseudolikelihood objective is a very close match to the type of predictions we would like to make, and therefore pseudolikelihood may well provide a better training objective than likelihood.** For example, if we are trying to learn a Markov network for collaborative filtering (box 18.C), we generally take the user's preference for all items except the query item to be observed. **Conversely, if a typical query involves most or all of the variables in the model, the likelihood objective is more appropriate.** For example, if we are trying to learn a model for image segmentation (box 4.B), the segment value of all of the pixels is unobserved. (We note that this last application is a CRF, where we would generally use a conditional likelihood objective, conditioned on the actual pixel values.) In this case, a (conditional) likelihood is a more appropriate objective than the (conditional) pseudolikelihood.

However, even in cases where the likelihood is the more appropriate objective, we may have to resort to pseudolikelihood for computational reasons. In many cases, this objective performs surprisingly well. However, in others, it can provide a fairly poor approximation.

Example 20.3

Consider a Markov network over three variables X_1, X_2, Y, where each pair is connected by an edge. Assume that X_1, X_2 are very highly correlated (almost identical) and both are somewhat (but not as strongly) correlated with Y. In this case, the best predictor for X_1 is X_2, and vice versa, so the pseudolikelihood objective is likely to overestimate significantly the parameters on the X_1—X_2, and almost entirely dismiss the X_1—Y and X_2—Y edges. The resulting model would be an excellent predictor for X_2 when X_1 is observed, but virtually useless when only Y and not X_1 is observed. ■

This example is typical of a general phenomenon: Pseudolikelihood, by assuming that each variable's local neighborhood is fully observed, is less able to exploit information obtained from weaker or longer-range dependencies in the distribution.

generalized pseudolikelihood

This limitation also suggests a spectrum of approaches known as *generalized pseudolikelihood*, which can reduce the extent of this problem. In particular, in the objective of equation (20.20), rather than using a product of terms over individual variables, we can consider terms where the left-hand side consists of several variables, conditioned on the rest. More precisely, we can define a set of subsets of variables $\{\boldsymbol{X}_s : s \in \mathcal{S}\}$, and then define an objective:

$$\ell_{\text{GPL}}(\boldsymbol{\theta} : \mathcal{D}) = \frac{1}{M} \sum_m \sum_s \ln P(\boldsymbol{x}_s[m] \mid \boldsymbol{x}_{-s}[m], \boldsymbol{\theta}), \tag{20.24}$$

where $\boldsymbol{X}_{-s} = \mathcal{X} - \boldsymbol{X}_s$.

Clearly, there are many possible choices of subsets $\{\boldsymbol{X}_s\}$. For different such choices, this expression generalizes several objectives: the likelihood, the pseudolikelihood, and even the conditional likelihood. When variables are together in the same subset $\boldsymbol{X}_s$, the relationship between them is subject (at least in part) to a likelihood-like objective, which tends to induce a more correct model of the joint distribution over them. However, as for the likelihood,

this objective requires that we compute expected counts over the variables in each $\boldsymbol{X}_s$ given an assignment to $\boldsymbol{X}_{-s}$. Thus, the choice of $\boldsymbol{X}_s$ offers a trade-off between "accuracy" and computational cost. One common choice of subsets is the set of all cliques in the Markov networks, which guarantees that the factor associated with each clique is optimized in at least one likelihood-like term in the objective.

20.6.2 Contrastive Optimization Criteria

As we discussed, both likelihood and pseudolikelihood can be viewed as attempting to increase the "log-probability gap" between the log-probability of the observed instances in $\mathcal{D}$ and the logarithm of the aggregate probability of a set of instances. Building on this perspective, one can construct a range of methods that aim to increase the log-probability gap between $\mathcal{D}$ and some other instances. The intuition is that, by driving the probability of the observed data higher relative to other instances, we are tuning our parameters to predict the data better.

More precisely, consider again the case of a single training instance ξ. We can define a "contrastive" objective where we aim to maximize the log-probability gap:

$$\left(\ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi' \mid \boldsymbol{\theta})\right),$$

where ξ' is some other instance, whose selection we discuss shortly. Importantly, this expression takes a very simple form:

$$\left(\ln \tilde{P}(\xi \mid \boldsymbol{\theta}) - \ln \tilde{P}(\xi' \mid \boldsymbol{\theta})\right) = \boldsymbol{\theta}^T[\boldsymbol{f}(\xi) - \boldsymbol{f}(\xi')]. \tag{20.25}$$

Note that, for a fixed instantiation ξ', this expression is a linear function of $\boldsymbol{\theta}$ and hence is unbounded. Thus, in order for this type of function to provide a coherent optimization objective, the choice of ξ' will generally have to change throughout the course of the optimization. Even then, we must take care to prevent the parameters from growing unboundedly, an easy way of arbitrarily increasing the objective.

One can construct many variants of this type of method. Here, we briefly survey two that have been particularly useful in practice.

20.6.2.1 Contrastive Divergence

contrastive divergence

One approach whose popularity has recently grown is the *contrastive divergence* method. In this method, we "contrast" our data instances $\mathcal{D}$ with a set of randomly perturbed "neighbors" $\mathcal{D}^-$. In particular, we aim to maximize:

$$\ell_{\mathrm{CD}}(\boldsymbol{\theta} : \mathcal{D} \| \mathcal{D}^-) = \left[\boldsymbol{E}_{\xi \sim \hat{P}_{\mathcal{D}}}\left[\ln \tilde{P}_{\boldsymbol{\theta}}(\xi)\right] - \boldsymbol{E}_{\xi \sim \hat{P}_{\mathcal{D}^-}}\left[\ln \tilde{P}_{\boldsymbol{\theta}}(\xi)\right]\right], \tag{20.26}$$

where $\hat{P}_{\mathcal{D}}$ and $\hat{P}_{\mathcal{D}^-}$ are the empirical distributions relative to $\mathcal{D}$ and $\mathcal{D}^-$, respectively.

As we discussed, the set of "contrasted" instances $\mathcal{D}^-$ will necessarily differ at different stages in the search. Given a current parameterization $\boldsymbol{\theta}$, what is a good choice of instances to which we want to contrast our data instances $\mathcal{D}$? One intuition is that we want to move our parameters $\boldsymbol{\theta}$ in a direction that increases the probability of instances in $\mathcal{D}$ relative to "typical" instances in our current distribution; that is, we want to increase the probability gap between instances

$\xi \in \mathcal{D}$ and instances ξ sampled randomly from $P_{\boldsymbol{\theta}}$. Thus, we can generate a contrastive set $\mathcal{D}^-$ by sampling from $P_{\boldsymbol{\theta}}$, and then maximizing the objective in equation (20.26).

How do we sample from $\mathcal{P}_{\boldsymbol{\theta}}$? As in section 12.3, we can run a Markov chain defined by the Markov network $P_{\boldsymbol{\theta}}$, using, for example, Gibbs sampling, and initializing from the instances in $\mathcal{D}$; once the chain mixes, we can collect samples from the distribution $P_{\boldsymbol{\theta}}$. Unfortunately, sampling from the chain for long enough to achieve mixing usually takes far too long to be feasible as the inner loop of a learning algorithm. However, there is an alternative approach, which is both less expensive and more robust. Rather than run the chain defined by $P_{\boldsymbol{\theta}}$ to convergence, we initialize from the instances in $\mathcal{D}$, and run the chain only for a few steps; we then use the instances generated by these short sampling runs to define $\mathcal{D}^-$.

Intuitively, this approach has significant appeal: We want our model to give high probability to the instances in $\mathcal{D}$; our current parameters, initialized at $\mathcal{D}$, are causing us to move away from the instances in $\mathcal{D}$. Thus, we want to move our parameters in a direction that increases the probability of the instances in $\mathcal{D}$ relative to the "perturbed" instances in $\mathcal{D}^-$.

The gradient of this objective is also very intuitive, and easy to compute:

$$\frac{\partial}{\partial \theta_i} \ell_{\text{CD}}(\boldsymbol{\theta} : \mathcal{D} \| \mathcal{D}^-) = \boldsymbol{E}_{\hat{P}_{\mathcal{D}}}[f_i(\mathcal{X})] - \boldsymbol{E}_{\hat{P}_{\mathcal{D}^-}}[f_i(\mathcal{X})]. \tag{20.27}$$

Note that, if we run the Markov chain to the limit, the samples in $\mathcal{D}^-$ are generated from $P_{\boldsymbol{\theta}}$; in this case, the second term in this difference converges to $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_i]$, which is precisely the second term in the gradient of the log-likelihood objective in equation (20.4). Thus, at the limit of the Markov chain, this learning procedure is equivalent (on expectation) to maximizing the log-likelihood objective. However, in practice, the approximation that we get by taking only a few steps in the Markov chain provides a good direction for the search, at far lower computational cost. In fact, empirically it appears that, because we are taking fewer sampling steps, there is less variance in our estimation of the gradient, leading to more robust convergence.

20.6.2.2 Margin-Based Training ⋆

A very different intuition arises in settings where our goal is to use the learned network for predicting a MAP assignment. For example in our image segmentation application of box 4.B, we want to use the learned network to predict a single high-probability assignment to the pixels that will encode our final segmentation output. This type of reasoning only arises in the context of conditional queries, since otherwise there is only a single MAP assignment (in the unconditioned network). Thus, we describe the objective in this section in the context of conditional Markov networks.

Recall that, in this setting, our training set consists of a set of pairs $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^{M}$. Given an observation $\boldsymbol{x}[m]$, we would like our learned model to give the highest probability to $\boldsymbol{y}[m]$. In other words, we would like the probability $P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m])$ to be higher than any other probability $P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m])$ for $\boldsymbol{y} \neq \boldsymbol{y}[m]$. In fact, to increase our confidence in this prediction, we would like to increase the log-probability gap as much as possible, by increasing:

$$\ln P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \left[\max_{\boldsymbol{y} \neq \boldsymbol{y}[m]} \ln P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m]) \right].$$

This difference between the log-probability of the target assignment $\boldsymbol{y}[m]$ and that of the "next best" assignment is called the *margin*. The higher the margin, the more confident the model is

margin-based estimation

in selecting $\boldsymbol{y}[m]$. Roughly speaking, *margin-based estimation* methods usually aim to maximize the margin.

One way of formulating this of *max-margin* objective as an optimization problem is as follows:

$$\begin{array}{ll} \textbf{Find} & \gamma, \boldsymbol{\theta} \\ \textbf{maximizing} & \gamma \\ \textbf{subject to} & \\ & \ln P_{\boldsymbol{\theta}}(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \ln P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}[m]) \quad \geq \quad \gamma \qquad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m]. \end{array}$$

The objective here is to maximize a single parameter γ, which encodes the worst-case margin over all data instances, by virtue of the constraints, which impose that the log-probability gap between $\boldsymbol{y}[m]$ and any other assignment $\boldsymbol{y}$ (given $\boldsymbol{x}[m]$) is at least γ. Importantly, due to equation (20.25), the first set of constraints can be rewritten in a simple linear form:

$$\boldsymbol{\theta}^T(\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq \gamma.$$

With this reformulation of the constraints, it becomes clear that, if we find any solution that achieves a positive margin, we can increase the margin unboundedly simply by multiplying all the parameters through by a positive constant factor. To make the objective coherent, we can bound the magnitude of the parameters by constraining their L_2-norm: $\|\boldsymbol{\theta}\|_2^2 = \boldsymbol{\theta}^T\boldsymbol{\theta} = \sum_i \theta_i^2 = 1$; or, equivalently, we can decide on a fixed margin and try to reduce the magnitude of the parameters as much as possible. With the latter approach, we obtain the following optimization problem:

Simple-Max-Margin:

$$\begin{array}{ll} \textbf{Find} & \boldsymbol{\theta} \\ \textbf{minimizing} & \|\boldsymbol{\theta}\|_2^2 \\ \textbf{subject to} & \\ & \boldsymbol{\theta}^T(\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq 1 \qquad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m] \end{array}$$

quadratic program

convex optimization

At some level, this objective is simple: it is a *quadratic program* (QP) with linear constraints, and hence is a convex problem that can be solved using a variety of *convex optimization* methods. However, a more careful examination reveals that the problem contains a constraint for every m, and (more importantly) for every assignment $\boldsymbol{y} \neq \boldsymbol{y}[m]$. Thus, the number of constraints is exponential in the number of variables $\boldsymbol{Y}$, generally an intractable number.

constraint generation

However, these are not arbitrary constraints: the structure of the underlying Markov network is reflected in the form of the constraints, opening the way toward efficient solution algorithms. One simple approach uses *constraint generation*, a general-purpose method for solving optimization problems with a large number of constraints. Constraint generation is an iterative method, which repeatedly solves for $\boldsymbol{\theta}$, each time using a larger set of constraints. Assume we have some algorithm for performing constrained optimization. We initially run this algorithm using none of the margin constraints, and obtain the optimal solution $\boldsymbol{\theta}^0$. In most cases, this solution will not satisfy many of the margin constraints, and it is thus not a feasible solution to our original QP. We add one or more constraints that are violated by $\boldsymbol{\theta}^0$ into a set of *active constraints*. We now repeat the constrained optimization process to obtain a new solution $\boldsymbol{\theta}^1$, which is guaranteed

to satisfy the active constraints. We again examine the constraints, find ones that are violated, and add them to our active constraints. This process repeats until no constraints are violated by our solution. Clearly, since we only add constraints, this procedure is guaranteed to terminate: eventually there will be no more constraints to add. Moreover, when it terminates, the solution is guaranteed to be optimal: At any iteration, the optimization procedure is solving a relaxed problem, whose value is at least as good as that of the fully constrained problem. If the optimal solution to this relaxed problem happens to satisfy all of the constraints, no better solution can be found to the fully constrained problem.

This description leaves unanswered two important questions. First, how many constraints we will have to add before this process terminates? Fortunately, it can be shown that, under reasonable assumptions, at most a polynomial number of constraints will need to be added prior to termination. Second, how do we find violated constraints without exhaustively enumerating and checking every one? As we now show, we can perform this computation by running MAP inference in the Markov network induced by our current parameterization $\boldsymbol{\theta}$. To see how, recall that we either want to show that

$$\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) \geq \ln \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]) + 1$$

for every $\boldsymbol{y} \in \mathit{Val}(\boldsymbol{Y})$ except $\boldsymbol{y}[m]$, or we want to find an assignment $\boldsymbol{y}$ that violates this inequality constraint. Let

$$\boldsymbol{y}^{\mathit{map}} = \arg \max_{\boldsymbol{y} \neq \boldsymbol{y}[m]} \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]).$$

There are now two cases: If $\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) < \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1$, then this is a violated constraint, which can be added to our constraint set. Alternatively, if $\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) > \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1$, then, due to the selection of $\boldsymbol{y}^{\mathit{map}}$, we are guaranteed that

$$\ln \tilde{P}(\boldsymbol{y}[m], \boldsymbol{x}[m]) > \ln \tilde{P}(\boldsymbol{y}^{\mathit{map}}, \boldsymbol{x}[m]) + 1 \geq \ln \tilde{P}(\boldsymbol{y}, \boldsymbol{x}[m]) + 1,$$

for every $\boldsymbol{y} \neq \boldsymbol{y}[m]$. That is, in this second case, all of the exponentially many constraints for the m'th data instance are guaranteed to be satisfied. As written, the task of finding $\boldsymbol{y}^{\mathit{map}}$ is not a simple MAP computation, due to the constraint that $\boldsymbol{y}^{\mathit{map}} \neq \boldsymbol{y}[m]$. However, this difficulty arises only in the case where the MAP assignment is $\boldsymbol{y}[m]$, in which case we need only find the second-best assignment. Fortunately, it is not difficult to adapt most MAP solution methods to the task of finding the second-best assignment (see, for example, exercise 13.5).

The use of MAP rather than sum-product as the inference algorithm used in the inner loop of the learning algorithm can be of significance. As we discussed, MAP inference admits the use of more efficient optimization algorithms that are not applicable to sum-product. In fact, as we discussed in section 13.6, there are even cases where sum-product is intractable, whereas MAP can be solved in polynomial time.

However, the margin constraints we use here fail to address two important issues. First, we are not guaranteed that there exists a model that can correctly select $\boldsymbol{y}[m]$ as the MAP assignment for every data instance m: First, our training data may be noisy, in which case $\boldsymbol{y}[m]$ may not be the actual desired assignment. More importantly, our model may not be expressive enough to always pick the desired target assignment (and the "simple" solution of increasing its expressive power may lead to overfitting). Because of the worst-case nature of our optimization objective, when we cannot achieve a positive margin for every data instance, there is no longer

any incentive in getting a better margin for those instances where a positive margin can be achieved. Thus, the solution we obtain becomes meaningless. To address this problem, we must allow for instances to have a nonpositive margin and simply penalize such exceptions in the objective; the penalization takes the form of *slack variables* η_m that measure the extent of the violation for the m'th data instances. This approach allows the optimization to trade off errors in the labels of a few instances for a better solution overall.

A second, related problem arises from our requirement that our model achieve a uniform margin for all $\boldsymbol{y} \neq \boldsymbol{y}[m]$. To see why this requirement can be problematic, consider again our image segmentation problem. Here, $\boldsymbol{x}[m]$ are features derived from the image, $\boldsymbol{y}[m]$ is our "ground truth" segmentation, and other assignments $\boldsymbol{y}$ are other candidate segmentations. Some of these candidate segmentations differ from $\boldsymbol{y}[m]$ only in very limited ways (perhaps a few pixels are assigned a different label). In this case, we expect that a reasonable model $P_{\boldsymbol{\theta}}$ will ascribe a probability to these "almost-correct" candidates that is very close to the probability of the ground truth. If so, it will be difficult to find a good model that achieves a high margin. Again, due to the worst-case nature of the objective, this can lead to inferior models. We address this concern by allowing the required margin $\ln P(\boldsymbol{y}[m] \mid \boldsymbol{x}[m]) - \ln P(\boldsymbol{y} \mid \boldsymbol{x}[m])$ to vary with the "distance" between $\boldsymbol{y}[m]$ and $\boldsymbol{y}$, with assignments $\boldsymbol{y}$ that are more similar to $\boldsymbol{y}[m]$ requiring a smaller margin. In particular, using the ideas of the *Hamming loss*, we can define $\Delta_m(\boldsymbol{y})$ to be the number of variables $Y_i \in \boldsymbol{Y}$ such that $y_i \neq y_i[m]$, and require that the margin increase linearly in this discrepancy.

Hamming loss

Putting these two modifications together, we obtain our final optimization problem:

Max-Margin:

Find $\boldsymbol{\theta}$

maximizing $\|\boldsymbol{\theta}\|_2^2 + C \sum_m \eta_m$

subject to

$$\boldsymbol{\theta}^T (\boldsymbol{f}(\boldsymbol{y}[m], \boldsymbol{x}[m]) - \boldsymbol{f}(\boldsymbol{y}, \boldsymbol{x}[m])) \geq \Delta_m(\boldsymbol{y}) - \eta_m \quad \forall m, \boldsymbol{y} \neq \boldsymbol{y}[m].$$

Here, C is a constant that determines the balance between the two parts of the objective: how much we choose to penalize mistakes (negative margins) for some instances, versus achieving a higher margin overall.

Fortunately, the same constraint generation approach that we discussed can also be applied in this case (see exercise 20.14).

20.7 Structure Learning

model selection

We now move to the problem of *model selection*: learning a network structure from data. As usual, there are two types of solution to this problem: the constraint-based approaches, which search for a graph structure satisfying the independence assumptions that we observe in the empirical distribution; and the score-based approaches, which define an objective function for different models, and then search for a high-scoring model.

From one perspective, the constraint-based approaches appear relatively more advantageous here than they did in the case of Bayesian network learning. First, the independencies associated with separation in a Markov network are much simpler than those associated with d-separation

in a Bayesian network; therefore, the algorithms for inferring the structure are much simpler here. Second, recall that all of our scoring functions were based on the likelihood function; here, unlike in the case of Bayesian networks, even evaluating the likelihood function is a computationally expensive procedure, and often an intractable one.

On the other side, the disadvantage of the constraint-based approaches remains: their lack of robustness to statistical noise in the empirical distribution, which can give rise to incorrect independence assumptions. We also note that the constraint based approaches produce only a structure, and not a fully specified model of a distribution. To obtain such a distribution, we need to perform parameter estimation, so that we eventually encounter the computational costs associated with the likelihood function. Finally, in the context of Markov network learning, it is not clear that learning the global independence structure is necessarily the appropriate problem. In the context of learning Bayesian networks we distinguished between learning the global structure (the directed graph) and local structure (the form of each CPD). In learning undirected models we can similarly consider both the problem of learning the undirected graph structure and the particular set of factors or features that represent the parameterization of the graph. Here, however, it is quite common to find distributions that have a compact factorization yet have a complex graph structure. One extreme example is the fully connected network with pairwise potentials. Thus, in many domains we want to learn the factorization of the joint distribution, which often cannot be deduced from the global independence assumptions.

We will review both types of approach, but we will focus most of the discussion on score-based approaches, since these have received more attention.

20.7.1 Structure Learning Using Independence Tests

constraint-based structure learning

independence tests

We first consider the idea of *constraint-based structure learning*. Recall that the structure of a Markov network specifies a set of independence assertions. We now show how we can use *independence tests* to reconstruct the Markov network structure. For this discussion, assume that the generating distribution P^* is positive and can be represented as a Markov network $\mathcal{H}^*$ that is a perfect map of P^*. Thus, we want to perform a set of independence tests on P^* and recover $\mathcal{H}^*$. To make the problem tractable, we further assume that the degree of nodes in $\mathcal{H}^*$ is at most d^*.

Recall that in section 4.3.2 we considered three set sets of independencies that characterize a Markov network: global independencies that include all consequences of separation in the graph; Markov independencies that describe the independence of each variable X from the rest of the variables given its Markov blanket; and pairwise independencies that describe the independence of each nonadjacent pair of variables X, Y given all other variables. We showed there that these three definitions are equivalent in positive distributions.

Can we use any of these concepts to recover the structure of $\mathcal{H}^*$? Intuitively, we would prefer to examine a smaller set of independencies, since they would require fewer independence tests. Thus, we should focus either on the local Markov independencies or pairwise independencies.

local Markov independencies

Recall that *local Markov independencies* are of the form

$$(X \perp \mathcal{X} - \{X\} - \mathrm{MB}_{\mathcal{H}^*}(X) \mid \mathrm{MB}_{\mathcal{H}^*}(X)) \quad \forall X$$

pairwise independencies

and *pairwise independencies* are of the form

$$(X \perp Y \mid \mathcal{X} - \{X, Y\}) \;\; \forall (X — Y) \notin \mathcal{H}.$$

Unfortunately, as written, neither of these sets of independencies can be checked tractably, since both involve the entire set of variables $\mathcal{X}$ and hence require measuring the probability of exponentially many events. The computational infeasibility of this requirement is obvious. But equally problematic are the statistical issues: these independence assertions are evaluated not on the true distribution, but on the empirical distribution. Independencies that involve many variables lead to fragmentation of the data, and are much harder to evaluate without error. To estimate the distribution sufficiently well as to evaluate these independencies reliably, we would need exponentially many data points.

Thus, we need to consider alternative sets of independencies that involve only smaller subsets of variables. Several such approaches have been proposed; we review only one, as an example. Consider the network $\mathcal{H}^*$. Clearly, if X and Y are not neighbors in $\mathcal{H}^*$, then they are separated by the *Markov blanket* $\mathrm{MB}_{\mathcal{H}^*}(X)$ and also by $\mathrm{MB}_{\mathcal{H}^*}(Y)$. Thus, we can find a set $\boldsymbol{Z}$ with $|\boldsymbol{Z}| \leq \min(|\mathrm{MB}_{\mathcal{H}^*}(X)|, |\mathrm{MB}_{\mathcal{H}^*}(Y)|)$ so that $sep_{\mathcal{H}^*}(X; Y \mid \boldsymbol{Z})$ holds. On the other hand, if X and Y are neighbors in $\mathcal{H}^*$, then we cannot find such a set $\boldsymbol{Z}$. Because $\mathcal{H}^*$ is a perfect map of P^*, we can show that

Markov blanket

$$X — Y \notin \mathcal{H}^* \text{ if and only if } \exists \boldsymbol{Z}, |\boldsymbol{z}| \leq d^* \& P^* \models (X \perp Y \mid \boldsymbol{Z}).$$

Thus, we can determine whether $X — Y$ is in $\mathcal{H}^*$ using $\sum_{k=0}^{d^*} \binom{n-2}{k}$ independence tests. Each of these independence tests involves only $d^* + 2$ variables, which, for low values of d^*, can be tractable. We have already encountered this test in section 3.4.3.1, as part of our Bayesian network construction procedure. If fact, it is not hard to show that, given our assumptions and perfect independence tests, the Build-PMap-Skeleton procedure of algorithm 3.3 reconstructs the correct Markov structure $\mathcal{H}^*$ (exercise 20.15).

This procedure uses a polynomial number of tests. Thus, the procedure runs in polynomial time. Moreover, if the probability of a false answer in any single independence test is at most ϵ, then the probability that any one of the independence tests fails is at most $\sum_{k=0}^{d^*} \binom{n-2}{k} \epsilon$. Therefore, for sufficiently small ϵ, we can use this analysis to prove that we can reconstruct the correct network structure $\mathcal{H}^*$ with high probability.

While this result is satisfying at some level, there are significant limitations. **First, the number of samples required to obtain correct answers for all of the independence tests can be very large in practice. Second, the correctness of the algorithm is based on several important assumptions: that there is a Markov network that is a perfect map of P^*; that this network has a bounded degree; and that we have enough data to obtain reliable answers to the independence tests. When these assumptions are violated, this algorithm can learn incorrect network structures.**

Example 20.4

Assume that the underlying distribution P^* is a Bayesian network with a v-structure $X \rightarrow Z \leftarrow Y$. We showed in section 3.4.3 that, assuming perfect independence tests, Build-PMap-Skeleton *learns the skeleton of $\mathcal{G}^*$. However, the Markov network $\mathcal{H}^*$ that is an I-map for P^* is the moralized network, which contains, in addition to the skeleton edges, edges between parents of a joint child. These edges will not be learned correctly by this procedure. In particular, we have that $(X \perp Y \mid \emptyset)$ holds, and so the algorithm will allow us to remove the edge between X and Y, even though it exists in the true network $\mathcal{H}^*$.* ■

The failure in this example results from the fact that the distribution P^* does not have a perfect

map that is a Markov network. Because many real-life distributions do not have a perfect map that is a compact graph, the applicability of this approach can be limited.

Moreover, as we discussed, this approach focuses solely on reconstructing the network structure and does not attempt to learn the the structure of the factorization, or to estimate the parameters. In particular, we may not have enough data to reliably estimate parameters for the structure learned by this procedure, limiting its usability in practice. Nevertheless, as in the case of Bayesian network structure learning, constraint-based approaches can be a useful tool for obtaining qualitative insight into the global structure of the distribution, and as a starting point for the search in the score-based methods.

20.7.2 Score-Based Learning: Hypothesis Spaces

hypothesis space

We now move to the score-based structure learning approach. As we discussed earlier, this approach formulates structure learning as an optimization problem: We define a *hypothesis space* consisting of a set of possible networks; we also define an objective function, which is used to score different candidate networks; and then we construct a search algorithm that attempts to identify a high-scoring network in the hypothesis space. We begin in this section by discussing the choice of hypothesis space for learning Markov networks. We discuss objective functions and the search strategy in subsequent sections.

There are several ways of formulating the search space for Markov networks, which vary in terms of the granularity at which they consider the network parameterization. At the coarsest-grained, we can pose the hypothesis space as the space of different structures of the Markov network itself and measure the model complexity in terms of the size of the cliques in the network. At the next level, we can consider parameterizations at the level of the factor graph, and measure complexity in terms of the sizes of the factors in this graph. At the finest level of granularity, we can consider a search space at the level of individual features in a log-linear model, and measure sparsity at the level of features included in the model.

The more fine-grained our hypothesis space, the better it allows us to select a parameterization that matches the properties of our distribution without overfitting. For example, the factor-graph approach allows us to distinguish between a single large factor over k variables and a set of $\binom{k}{2}$ pairwise factors over the same variables, requiring far fewer parameters. The feature-based approach also allows us to distinguish between a full factor over k variables and a single log-linear feature over the same set of variables.

Conversely, the finer-grained spaces can obscure the connection to the network structure, in that sparsity in the space of features selected does not correspond directly to sparsity in the model structure. For example, introducing even a single feature $f(\boldsymbol{d})$ into the model has the structural effect of introducing edges between all of the variables in $\boldsymbol{d}$. Thus, even models with a fairly small number of features can give rise to dense connectivity in the induced network. While this is not a problem from the statistical perspective of reliably estimating the model parameters from limited data, it can give rise to significant problems from the perspective of performing inference in the model. Moreover, a finer-grained hypothesis space also means that search algorithms take smaller steps in the space, potentially increasing the cost of our learning procedure. We will return to some of these issues.

We focus our presentation on the formulation of the search space in terms of log-linear models. Here, we have a set of features Ω, which are those that can potentially have nonzero

weight. Our task is to select a log-linear model structure $\mathcal{M}$, which is defined by some subset $\Phi[\mathcal{M}] \subseteq \Omega$. Let $\Theta[\mathcal{M}]$ be the set of parameterizations $\boldsymbol{\theta}$ that are compatible with the model structure: that is, those where $\theta_i \neq 0$ only if $f_i \in \Phi[\mathcal{M}]$. A structure and a compatible parameterization define a log-linear distribution via:

$$P(\mathcal{X} \mid \mathcal{M}, \boldsymbol{\theta}) = \frac{1}{Z} \exp\left\{ \sum_{i \in \Phi[\mathcal{M}]} \theta_i f_i(\xi) \right\} = \frac{1}{Z} \exp\left\{ \boldsymbol{f}^T \boldsymbol{\theta} \right\},$$

where, because of the compatibility of $\boldsymbol{\theta}$ with $\mathcal{M}$, a feature not in $\Phi[\mathcal{M}]$ does not influence in the final vector product, since it is multiplied by a parameter that is 0.

Regardless of the formulation chosen, we may sometimes wish to impose structural constraints that restrict the set of graph structures that can be selected, in order to ensure that we learn a network with certain sparsity properties. In particular, one choice that has received some attention is to restrict the class of networks learned to those that have a certain bound on the *tree-width*. By placing a tight bound on the tree-width, we prevent an overly dense network from being selected, and thereby reduce the chance of overfitting. Moreover, because models of low tree-width allow exact inference to be performed efficiently (to some extent), this restriction also allows the computational steps required for evaluating the objective during the search to be performed efficiently. However, this approach also has limitations. First, it turns out to be nontrivial to implement, since computing the tree-width of a graph is itself an intractable problem (see theorem 9.7); even keeping the graph under the required width is not simple. Moreover, many of the distributions that arise in real-world applications cannot be well represented by networks of low tree-width.

bounded tree-width

20.7.3 Objective Functions

We now move to considering the objective function that we aim to optimize in the score-based approach. We note that our discussion in this section uses the likelihood function as the basis for the objectives we consider; however, we can also consider similar objectives based on various approximations to the likelihood (see section 20.6); most notably, the pseudolikelihood has been used effectively as a substitute for the likelihood in the context of structure learning, and most of our discussion carries over without change to that setting.

20.7.3.1 Likelihood Function

The most straightforward objective function is the likelihood of the training data. As before, we take the score to be the log-likelihood, defining:

$$\text{score}_L(\mathcal{M} : \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta[\mathcal{M}]} \ln P(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}) = \ell(\langle \mathcal{M}, \hat{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}),$$

where $\hat{\boldsymbol{\theta}}_{\mathcal{M}}$ are the maximum likelihood parameters compatible with $\mathcal{M}$.

The likelihood score measures the fitness of the model to the data. However, for the same reason discussed in chapter 18, it prefers more complex models. In particular, if $\Phi[\mathcal{M}_1] \subset \Phi[\mathcal{M}_2]$ then $\text{score}_L(\mathcal{M}_1 : \mathcal{D}) \leq \text{score}_L(\mathcal{M}_2 : \mathcal{D})$. Typically, this inequality is strict, due to the ability of the richer model to capture noise in the data.

Therefore, the likelihood score can be used only with very strict constraints on the expressive model of the model class that we are considering. Examples include bounds on the structure of the Markov network (for example, networks with low tree-width) or on the number of features used. A second option, which also provides some regularization of parameter values, is to use an alternative objective that penalizes the likelihood in order to avoid overfitting.

20.7.3.2 Bayesian Scores

Bayesian score

Recall that, for Bayesian networks, we used a *Bayesian score*, whose primary term is a marginal likelihood that integrates the likelihood over all possible network parameterizations: $\int P(\mathcal{D} \mid \mathcal{M}, \boldsymbol{\theta}) P(\boldsymbol{\theta} \mid \mathcal{M}) d\boldsymbol{\theta}$. This score accounts for our uncertainty over parameters using a Bayesian prior; it avoided overfitting by preventing overly optimistic assessments of the model fit to the training data. In the case of Bayesian networks, we could efficiently evaluate the marginal likelihood. In contrast, in the case of undirected models, this quantity is difficult to evaluate, even using approximate inference methods.

BIC score

Instead, we can use asymptotic approximations of the marginal likelihood. The simplest approximation is the *BIC score*:

$$\text{score}_{BIC}(\mathcal{M} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \hat{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) - \frac{\dim(\mathcal{M})}{2} \ln M,$$

model dimension

where $\dim(\mathcal{M})$ is the *dimension* of the model and M the number of instances in $\mathcal{D}$. This quantity measures the degrees of freedom of our parameter space. When the model has nonredundant features, $\dim(\mathcal{M})$ is exactly the number of features. When there is redundancy, the dimension is smaller than the number of features. Formally, it is the rank of the matrix whose rows are complete assignments ξ_i to $\mathcal{X}$, whose columns are features f_j, and whose entries are $f_j(\xi_i)$. This matrix, however, is exponential in the number of variables, and therefore its rank cannot be computed efficiently. Nonetheless, we can often estimate the number of nonredundant parameters in the model. As a very coarse upper bound, we note that the number of nonredundant features is always upper-bounded by the size of the full table representation of the Markov network, which is the total number of entries in the factors.

Laplace approximation

The BIC approximation penalizes each degree of freedom (that is, free parameter) by a fixed amount, which may not be the most appropriate penalty. Several more refined alternatives have been proposed. One common choice is the *Laplace approximation*, which provides a more explicit approximation to the marginal likelihood:

$$\text{score}_{Laplace}(\mathcal{M} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \tilde{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) + \ln P(\tilde{\boldsymbol{\theta}}_{\mathcal{M}} \mid \mathcal{M}) + \frac{\dim(\mathcal{M})}{2} \ln(2\pi) - \frac{1}{2} \ln |A|,$$

MAP estimation

where $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$ are the parameters for $\mathcal{M}$ obtained from *MAP estimation*:

$$\tilde{\boldsymbol{\theta}}_{\mathcal{M}} = \arg\max_{\boldsymbol{\theta}} P(\mathcal{D} \mid \boldsymbol{\theta}, \mathcal{M}) P(\boldsymbol{\theta} \mid \mathcal{M}), \tag{20.28}$$

Hessian

and A is the negative *Hessian* matrix:

$$A_{i,j} = -\frac{\partial}{\partial \theta_i \partial \theta_j} \left(\ell(\langle \mathcal{M}, \boldsymbol{\theta} \rangle : \mathcal{D}) + \ln P(\boldsymbol{\theta} \mid \mathcal{M}) \right),$$

evaluated at the point $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$.

As we discussed in section 19.4.1.1, the Laplace score also takes into account the local shape of the posterior distribution around the MAP parameters. It therefore provides a better approximation than the BIC score. However, as we saw in equation (20.5), to compute the Hessian, we need to evaluate the pairwise covariance of every feature pair given the model, a computation that may be intractable in many cases.

20.7.3.3 Parameter Penalty Scores

An alternative to approximations of the marginal likelihood are methods that simply evaluate the maximum posterior probability

$$\text{score}_{MAP}(\mathcal{M} \ : \ \mathcal{D}) = \max_{\boldsymbol{\theta} \in \Theta[\mathcal{M}]} \ell(\langle \mathcal{M}, \tilde{\boldsymbol{\theta}}_{\mathcal{M}} \rangle : \mathcal{D}) + \ln P(\tilde{\boldsymbol{\theta}}_{\mathcal{M}} \mid \mathcal{M}), \tag{20.29}$$

where $\tilde{\boldsymbol{\theta}}_{\mathcal{M}}$ are the MAP parameters for $\mathcal{M}$, as defined in equation (20.28). One intuition for this type of *MAP score* is that the prior "regularizes" the likelihood, moving it away from the maximum likelihood values. If the likelihood of these parameters is still high, it implies that the model is not too sensitive to particular choice of maximum likelihood parameters, and thus it is more likely to generalize.

MAP score

Although the regularized parameters may achieve generalization, this approach achieves model selection only for certain types of prior. To understand why, note that the MAP score is based on a distribution not over structures, but over parameters. We can view any parameterization $\boldsymbol{\theta}_{\mathcal{M}}$ as a parameterization to the "universal" model defined over our entire set of features Ω: one where features not in $\Phi[\mathcal{M}]$ receive weight 0. Assuming that our parameter prior simply ignores zero weights, we can view our score as simply evaluating different choices of parameterizations $\boldsymbol{\theta}_{\Omega}$ to this universal model.

We have already discussed several parameter priors and their effect on the learned parameters. Most parameter priors are associated with the magnitude of the parameters, rather than the complexity of the graph as a discrete data structure. In particular, as we discussed, although *L_2-regularization* will tend to drive the parameters toward zero, few will actually hit zero, and so structural sparsity will not be achieved. Thus, like the likelihood score, the L_2-regularized MAP objective will generally give rise to a fully connected structure. Therefore, this approach is generally not used in the context of model selection (at least not in isolation).

L_2-regularization

A more appropriate approach for this task is *L_1-regularization*, which does have the effect of driving model parameters toward zero, and thus can give rise to a sparse set of features. In other words, the structure that optimizes the *L_1-MAP score* is not, in general, the universal structure Ω. Indeed, as we will discuss, an L_1 prior has other useful properties when used as the basis for a structure selection objective.

L_1-regularization

L_1-MAP score

However, as we have discussed, feature-level sparsity does not necessarily induce sparsity in the network. An alternative that does tend to have this property is the *block-L_1-regularization*. Here, we partition all the parameters into groups $\boldsymbol{\theta}_i = \{\theta_{i,1}, \ldots, \theta_{i,k_i}\}$ (for $i = 1, \ldots, l$). We now define a variant of the L_1 penalty that tends to make each parameter *group* either go to zero together, or not:

block-L_1-regularization

$$-\sum_{i=1}^{l} \left| \sqrt{\sum_{j=1}^{k_i} \theta_{i,j}^2} \right|. \tag{20.30}$$

To understand the behavior of this penalty term, let us consider its derivative for the simple case where we have two parameters in the same group, so that our expression takes the form $\sqrt{\theta_1^2 + \theta_2^2}$. We now have that:

$$\frac{\partial}{\partial \theta_1}\left[-\sqrt{\theta_1^2 + \theta_2^2}\right] = -\frac{\theta_1}{\sqrt{\theta_1^2 + \theta_2^2}}.$$

We therefore see that, when θ_2 is large, the derivative relative to θ_1 is fairly small, so that there is no pressure on θ_1 to go to 0. Conversely, when θ_2 is small, the derivative relative to θ_1 tends to -1, which essentially gives the same behavior as L_1 regularization. Thus, this prior tends to have the following behavior: if the overall magnitude of the parameters in the group is small, all of them will be forced toward zero; if the overall magnitude is large, there is little downward pressure on any of them.

In our setting, we can naturally apply this prior to give rise to sparsity in network structure. Assume that we are willing to consider, within our network, factors over scopes $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_l$. For each $\boldsymbol{Y}_i$, let $f_{i,j}$, for $j = 1, \ldots, k_i$, be all of the features whose scope is $\boldsymbol{Y}_i$. We now define a block-L_1 prior where we have a block for each set of parameters $\theta_{i1}, \ldots, \theta_{ik_i}$. The result of this prior would be to select together nonzero parameters for an entire set of features associated with a particular scope.

Finally, we note that one can also use multiple penalty terms on the likelihood function. For example, a combination of a parameter penalty and a structure penalty can often provide both regularization of the parameters and a greater bias toward sparse structures.

20.7.4 Optimization Task

Having selected an objective function for our model structure, it remains to address the optimization problem.

20.7.4.1 Greedy Structure Search

local search

As in the approach used for structure learning of general Bayesian networks (section 18.4.3), the external search over structures is generally implemented by a form of *local search*. Indeed, the general form of the algorithms in appendix A.4.2 applies to our feature-based view of Markov network learning. The general template is shown in algorithm 20.1. Roughly speaking, the algorithm maintains a current structure, defined in terms of a set of features $\mathcal{F}$ in our log-linear model. At each point in the search, the algorithm optimizes the model parameters relative to the current feature set and the structure score. Using the current structure and parameters, it estimates the improvement of different structure modification steps. It then selects some subset of modifications to implement, and returns to the parameter optimization step, initializing from the current parameter setting. This process is repeated until a termination condition is reached. This general template can be instantiated in many ways, including the use of different hypothesis spaces (as in section 20.7.2) and different scoring functions (as described in section 20.7.3).

20.7.4.2 Successor Evaluation

Although this approach is straightforward at a high level, there are significant issues with its implementation. Importantly, the reasons that made this approach efficient in the context of

Algorithm 20.1 Greedy score-based structure search algorithm for log-linear models

```
Procedure Greedy-MN-Structure-Search (
  Ω,    // All possible features
  F_0,    // initial set of features
  score(· : D),    // Score
)
1   F' ← F_0    // New feature set
2   θ ← 0
3   do
4     F ← F'
5     θ ← Parameter-Optimize(F, θ, score(· : D))
6       // Find parameters that optimize the score objective, relative to
          current feature set, initializing from the current parameters
7     for each f_k ∈ F such that θ_k = 0
8       F ← F − f_k
9         // Remove inactive features
10    for each operator o applicable to F
11      Let Δ̂_o be the approximate improvement for o
12    Choose some subset O of operators based on Δ̂
13    F' ← O(F)    // Apply selected operators to F
14    while termination condition not reached
15  return (F, θ)
```

score decomposability

Bayesian networks do not apply here. In the case of Bayesian networks, evaluating the score of a candidate structure is a very easy task, which can be executed in closed form, at very low computation cost. Moreover, the Bayesian network score satisfies an important property: it *decomposes* according to the structure of the network. As we discussed, this property has two major implications. First, a local modification to the structure involves changing only a single term in the score (proposition 18.5); second, the change in score incurred by a particular change (for example, adding an edge) remains unchanged after modifications to other parts of the network (proposition 18.6). These properties allowed us to design efficient search procedure that does not need to reevaluate all possible candidates after every step, and that can cache intermediate computations to evaluate candidates in the search space quickly.

Unfortunately, none of these properties hold for Markov networks. For concreteness, consider the likelihood score, which is comparable across both network classes. First, as we discussed, even computing the likelihood of a fully specified model — structure as well as parameters — requires that we run inference for every instance in our training set. Second, to score a structure, we need to estimate the parameters for it, a problem for which there is no closed-form solution. Finally, none of the decomposition properties hold in the case of undirected models. By adding a new feature (or a set of features, for example, a factor), we change the weight ($\sum_i \theta_i f_i(\xi)$) associated with different instances. This change can be decomposed, since it is a linear function of the different features. However, this change also affects the partition function, and, as we saw in the context of parameter estimation, the partition function couples the effects of changes in

one parameter on the other. We can clearly see this phenomenon in figure 20.1, where the effect on the likelihood of modifying $f_1(b^1, c^1)$ clearly depends on the current value of the parameter for $f_2(a^1, b^1)$.

As a consequence, a local search procedure is considerably more expensive in the context of Markov networks. At each stage of the search, we need to evaluate the score for all of the candidates we wish to examine at that point in the search. This evaluation requires that we estimate the parameters for the structure, a process that itself requires multiple iterations of a numerical optimization algorithm, each involving inference over all of the instances in our training set. We can reduce somewhat the computational cost of the algorithm by using the observation that a single change to the structure of the network often does not result in drastic changes to the model. Thus, if we begin our optimization process from the current set of parameters, a reasonably small number of iterations often suffices to achieve convergence to the new set of parameters. Importantly, because all of the parameter objectives described are convex (when we have fully observable data), the initialization has no effect, and convergence to the global optimum remains guaranteed. Thus, this approach simply provides a way of speeding up the convergence to the optimal answer. (We note, however, that this statement holds only when we use exact inference; the choice of initialization can affect the accuracy of some approximate inference algorithms, and therefore the answers that we get.)

Unfortunately, although this observation does reduce the cost, the number of candidate hypotheses at each step is generally quite large. **The cost of running inference on each of the candidate successors is prohibitive, especially in cases where, to fit our target distribution well, we need to consider nontrivial structures. Thus, much of the work on the problem of structure learning for Markov networks has been devoted to reducing the computational cost of evaluating the score of different candidates during the search.** In particular, when evaluating different structure-modification operators in line 11, most algorithms use some heuristic to rank different candidates, rather than computing the exact delta-score of each operator. These heuristic estimates can be used either as the basis for the final selection, or as a way of pruning the set of possible successors, where the high-ranking candidates are then evaluated exactly. This design decision is a trade-off between the quality of our operator selection and the computational cost.

Even with the use of heuristics, the cost of taking a step in the search can be prohibitive, since it requires a reestimation of the network parameters and a reevaluation of the (approximate) delta-score for all of the operators. This suggests that it may be beneficial to select, at each structure modification step (line 12), not a single operator but a subset $\mathcal{O}$ of operators. This approach can greatly reduce the computational cost, but at a cost: our (heuristic) estimate of each operator can deteriorate significantly if we fail to take into account interactions between the effects of different operators. Again, this is a trade-off between the quality and cost of the operator selection.

20.7.4.3 Choice of Scoring Function

As we mentioned, the rough template of algorithm 20.1 can be applied to any objective function. However, the choice of objective function has significant implications on our ability to effectively optimize it. Let us consider several of the choices discussed earlier.

We first recall that both the log-likelihood objective and the L_2-regularized log-likelihood

generally give nonzero values to all parameters. In other words, if we allow the model to consider a set of features $\mathcal{F}$, an optimal model (maximum-likelihood or maximum L_2-regularized likelihood) over $\mathcal{F}$ will give a nonzero value to the parameters for all of these features. In other words, we cannot rely on these objectives to induce sparsity in the model structure. Thus, if we simply want to optimize these objectives, we should simply choose the richest model available in our hypothesis space and then optimize its parameters relative to the chosen objective.

One approach for deriving more compact models is to restrict the class of models to ones with a certain bound on the complexity (for example, networks of bounded tree-width, or with a bound on the number of edges or features allowed). However, these constraints generally introduce nontrivial combinatorial trade-offs between features, giving rise to a search space with multiple local optima, and making it generally intractable to find a globally optimal solution. A second approach is simply to halt the search when the improvement in score (or an approximation to it) obtained by a single step does not exceed a certain threshold. This heuristic is not unreasonable, since good features are generally introduced earlier, and so there is a general trend of diminishing returns. However, there is no guarantee that the solution we obtain is even close to the optimum, since there is no bound on how much the score would improve if we continue to optimize beyond the current step.

Scoring functions that explicitly penalize structure complexity — such as the BIC score or Laplace approximation — also avoid this degeneracy. Here, as in the case of Bayesian networks, we can consider a large hypothesis space and attempt to find the model in this space that optimizes the score. However, due to the discrete nature of the structure penalty, the score is discontinuous and therefore nonconcave. Thus, there is generally no guarantee of convergence to the global optimum. Of course, this limitation was also the case when learning Bayesian networks; as there, it can be somewhat alleviated by methods that avoid local maxima (such as tabu search, random restarts, or data perturbation).

However, in the case of Markov networks, we have another solution available to us, one that avoids the prospect of combinatorial search spaces and the ensuing problem of local optima. This solution is the use of L_1-regularized likelihood. As we discussed, the L_1-regularized likelihood is a concave function that has a unique global optimum. Moreover, this objective function naturally gives rise to sparse models, in that, at the optimum, many parameters have value 0, corresponding to the elimination of features from the model. We discuss this approach in more detail in the next section.

20.7.4.4 L_1-Regularization for Structure Learning

Recall that the L_1-regularized likelihood is simply the instantiation of equation (20.29) to the case of an L_1-prior:

$$\text{score}_{L_1}(\boldsymbol{\theta} \ : \ \mathcal{D}) = \ell(\langle \mathcal{M}, \boldsymbol{\theta} \rangle : \mathcal{D}) - \|\boldsymbol{\theta}\|_1. \tag{20.31}$$

Somewhat surprisingly, the L_1-regularized likelihood can be optimized in a way that guarantees convergence to the globally optimal solution. To understand why, recall that the task of optimizing the L_1-regularized log-likelihood is a convex optimization problem that has no local optima.[1] Indeed, in theory, we can entirely avoid the combinatorial search component when

1. There might be multiple global optima due to redundancy in the parameter space, but these global optima all form a single convex region. Therefore, we use the term "the global optimum" to refer to any point in this optimal region.

using this objective. We can simply introduce all of the possible features into the model and optimize the resulting parameter vector $\boldsymbol{\theta}$ relative to our objective. The sparsifying effect of the L_1 penalty will drive some of the parameters to zero. The parameters that, at convergence, have zero values correspond to features that are absent from the log-linear model. In this approach, we are effectively making a structure selection decision as part of our parameter optimization procedure. Although appealing, this approach is not generally feasible. In most cases, the number of potential features we may consider for inclusion in the model is quite large. Including all of them in the model simultaneously gives rise to an intractable structure for the inference that we use as part of the computation of the gradient.

Therefore, even in the context of the L_1-regularized likelihood, we generally implement the optimization as a double-loop algorithm where we separately consider the structure and parameters. However, there are several benefits to the L_1-regularized objective:

- We do not need to consider feature deletion steps in our combinatorial search.
- We can consider feature introduction steps in any (reasonable) order, and yet achieve convergence to the global optimum.
- We have a simple and efficient test for determining convergence.
- We can prove a PAC-learnability generalization bound for this type of learning.

We now discuss each of these points.

For the purpose of this discussion, assume that we currently have a model over a set of features $\mathcal{F}$, and assume that $\boldsymbol{\theta}^l$ optimizes our L_1-regularized objective, subject to the constraint that θ_k^l can be nonzero only if $f_k \in \mathcal{F}$. At this convergence point, any feature deletion step cannot improve the score: Consider any $f_k \in \mathcal{F}$; the case where f_k is deleted is already in the class of models that was considered when we optimized the choice of $\boldsymbol{\theta}^l$ — it is simply the model where $\theta_k^l = 0$. Indeed, the algorithm already discards features whose parameter was zeroed by the continuous optimization procedure (line 7 of algorithm 20.1). If our current optimized model $\boldsymbol{\theta}^l$ has $\theta_k^l \neq 0$, it follows that setting θ_k to 0 is suboptimal, and so deleting f_k can only reduce the score. Thus, there is no value to considering discrete feature deletion steps: features that should be deleted will have their parameters set to 0 by the continuous optimization procedure. We note that this property also holds, in principle, for other smooth objectives, such as the likelihood or the L_2-regularized likelihood; the difference is that for those objectives, parameters will generally not be set to 0, whereas the L_1 objective does tend to induce sparsity.

The second benefit arises directly from the fact that optimizing the L_1-regularized objective is a convex optimization problem. In such problems, any sequence of steps that continues to improve the objective (when possible) is guaranteed to converge to the global optimum. The restriction imposed by the set $\mathcal{F}$ induces a coordinate ascent approach: at each step, we are optimizing only the features in $\mathcal{F}$, leaving at 0 those parameters θ_k for $f_k \notin \mathcal{F}$. As long as each step continues to improve the objective, we are making progress toward the global optimum. At each point in the search, we consider the steps that we can take. If some step leads to an improvement in the score, we can take that step and continue with our search. If none of the steps lead to an improvement in the score, we are guaranteed that we have reached convergence to the global optimum. **Thus, the decision on which operators to consider at each point in the algorithm (line 12 of algorithm 20.1) is not relevant to the convergence of the**

algorithm to the true global optimum: As long as we repeatedly consider each operator until convergence, we are guaranteed that the global optimum is reached regardless of the order in which the operators are applied.

While this guarantee is an important one, we should interpret it with care. First, when we add features to the model, the underlying network becomes more complex, raising the cost of inference. Because inference is executed many times during the algorithm, adding many irrelevant features, even if they were eventually eliminated, can greatly degrade the computational performance time of the algorithm. Even more problematic is the effect when we utilize approximate inference, as is often the case. As we discussed, for many approximate inference algorithms, not only the running time but also the accuracy tend to degrade as the network becomes more complex. Because inference is used to compute the gradient for the continuous optimization, the degradation of inference quality can lead to models that are suboptimal. Moreover, because the resulting model is generally also used to estimate the benefit of adding new features, any inaccuracy can propagate further, causing yet more suboptimal features to be introduced into the model. **Hence, especially when the quality of approximate inference is a concern, it is worthwhile to select with care the features to be introduced into the model rather than blithely relying on the "guaranteed" convergence to a global optimum.**

Another important issue to be addressed is the problem of determining convergence in line 14 of Greedy-MN-Structure-Search. In other words, how do we test that none of the search operators we currently have available can improve the score? A priori, this task appears daunting, since we certainly do not want to try all possible feature addition/deletion steps, reoptimize the parameters for each of them, and then check whether the score has improved. Fortunately, there is a much more tractable solution. Specifically, we can show the following proposition:

Proposition 20.5 *Let $\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D})$ denote the gradient of the likelihood relative to θ_k, evaluated at $\boldsymbol{\theta}^l$. Let β be the hyperparameter defining the L_1 prior. Let $\boldsymbol{\theta}^l$ be a parameter assignment for which the following conditions hold:*

- *For any k for which $\theta_k^l \neq 0$ we have that*

$$\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D}) - \frac{1}{\beta}\text{sign}(\theta_k^l) = 0.$$

- *For any k for which $\theta_k^l = 0$ we have that*

$$|\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D})| < \frac{1}{2\beta}.$$

Then $\boldsymbol{\theta}^l$ is a global optimum of the L_1-regularized log-likelihood function:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \frac{1}{\beta}\sum_{i=1}^{k}|\theta_i|.$$

Proof We provide a rough sketch of the proof. The first condition guarantees that the gradient relative to any parameter for which $\theta_k^l \neq 0$ is zero, and hence the objective function cannot be improved by changing its value. The second condition deals with parameters $\theta_k^l = 0$, for which

the gradient is discontinuous at the convergence point. However, consider a point $\boldsymbol{\theta}'$ in the nearby vicinity of $\boldsymbol{\theta}$, so that $\theta'_k \neq 0$. At $\boldsymbol{\theta}'$, the gradient of the function relative to θ_k is very close to

$$\Delta_L^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^l, \mathcal{D}) - \frac{1}{\beta}\text{sign}(\theta'_k).$$

The value of this expression is positive if $\theta'_k < 0$ and negative if $\theta'_k > 0$. Thus, $\boldsymbol{\theta}^l$ is a local optimum of the L_1-regularized objective function. Because the function has only global optima, $\boldsymbol{\theta}^l$ must be a global optimum. ■

Thus, we can test convergence easily as a direct by-product of the continuous parameter optimization procedure executed at each step. We note that we still have to consider every feature that is not included in the model and compute the relevant gradient; but we do not have to go through the (much more expensive) process of trying to introduce the feature, optimizing the resulting model, and evaluating its score.

L-BFGS algorithm

So far, we have avoided the discussion of optimizing this objective. As we mentioned in section 20.3.1, a commonly used method for optimizing the likelihood is the *L-BFGS algorithm*, which uses gradient descent combined with line search (see appendix A.5.2). The problem with applying this method to the L_1-regularized likelihood is that the regularization term is not continuously differentiable: the gradient relative to any parameter θ_i changes at $\theta_i = 0$ from $+1$ to -1. Perhaps the simplest solution to this problem is to adjust the line-search procedure to avoid changing the sign of any parameter θ_i: If, during our line search, θ_i crosses from positive to negative (or vice versa), we simply fix it to be 0, and continue with the line search for the remaining parameters. Note that this decision corresponds to taking f_i out of the set of active features in this iteration. If the optimal parameter assignment has a nonzero value for θ_i, we are guaranteed that f_i will be introduced again in a later stage in the search, as we have discussed.

PAC-bound

Finally, as we mentioned, we can prove a useful theoretical guarantee for the results of L_1-regularized Markov network learning. Specifically, we can show the following *PAC-bound*:

Theorem 20.4

Let $\mathcal{X}$ be a set of variables such that $|Val(X_i)| \leq d$ for all i. Let P^* be a distribution, and $\delta, \epsilon, B > 0$. Let $\mathcal{F}$ be a set of all indicator features over all subsets of variables $\boldsymbol{X} \subset \mathcal{X}$ such that $|\boldsymbol{X}| \leq c$, and let

$$\Theta_{c,B} = \{\boldsymbol{\theta} \in \Theta[\mathcal{F}] \ : \ \|\boldsymbol{\theta}\|_1 \leq B\}$$

be all parameterizations of $\mathcal{F}$ whose L_1-norm is at most B. Let $\beta = \sqrt{c\ln(2nd/\delta)/(2M)}$. Let

$$\boldsymbol{\theta}^*_{c,B} = \arg\max_{\boldsymbol{\theta}\in\Theta_{c,B}} \boldsymbol{D}(P^* \| P_{\boldsymbol{\theta}})$$

be the best parameterization achievable within the class $\Theta_{c,B}$. For any data set $\mathcal{D}$, let

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}\in\Theta[\mathcal{F}]} \text{score}_{L_1}(\boldsymbol{\theta} \ : \ \mathcal{D}).$$

Then, for

$$M \geq \frac{2cB^2}{\epsilon^2}\ln\left(\frac{2nd}{\delta}\right),$$

with probability at least $1 - \delta$,

$$\boldsymbol{I}D(P^* \| P_{\hat{\boldsymbol{\theta}}}) \leq \boldsymbol{I}D(P^* \| P_{\boldsymbol{\theta}^{*c,B}}) + \epsilon.$$

In other words, this theorem states that, with high probability over data sets $\mathcal{D}$, the relative entropy to P^* achieved by the best L_1-regularized model is at most ϵ worse than the relative entropy achieved by the best model within the class of limited-degree Markov networks. This guarantee is achievable with a number of samples that is polynomial in ϵ, c, and B, and logarithmic in δ and d. The logarithmic dependence on n may feel promising, but we note that B is a sum of the absolute values of all network parameters; assuming we bound the magnitude of individual parameters, this terms grows linearly with the total number of network parameters. Thus, L_1-regularized learning provides us with a model that is close to optimal (within the class $\Theta_{c,B}$), using a polynomial number of samples.

20.7.5 Evaluating Changes to the Model

We now consider in more detail the candidate evaluation step that takes place in line 11 of Greedy-MN-Structure-Search. As we discussed, the standard way to reduce the cost of the candidate evaluation step is simply to avoid computing the exact score of each of the candidate successors, and rather to select among them using simpler heuristics. Many approximations are possible, ranging from ones that are very simple and heuristic to ones that are much more elaborate and provide certain guarantees.

Most simply, we can examine statistics of the data to determine features that may be worth including. For example, if two variables X_i and X_j are strongly correlated in the data, it may be worthwhile to consider introducing a factor over X_i, X_j (or a pairwise feature over one or more combinations of their values). The limitation of this approach is that it does not take into account the features that have already been introduced into the model and the extent to which they already explain the observed correlation.

grafting

A somewhat more refined approach, called *grafting*, estimates the benefit of introducing a feature f_k by compute the gradient of the likelihood relative to θ_k, evaluated at the current model. More precisely, assume that our current model is $(\mathcal{F}, \boldsymbol{\theta}^0)$. The *gradient heuristic* estimate to the delta-score (for score X) obtained by adding $f_k \notin \mathcal{F}$ is defined as:

gradient heuristic

$$\Delta_X^{\text{grad}}(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}) = \frac{\partial}{\partial \theta_k} \text{score}_X(\boldsymbol{\theta} \ : \ \mathcal{D}), \tag{20.32}$$

evaluated at the current parameters $\boldsymbol{\theta}^0$.

The gradient heuristic does account for the parameters already selected; thus, for example, it can avoid introducing features that are not relevant given the parameters already introduced. Intuitively, features that have a high gradient can induce a significant immediate improvement in the score, and therefore they are good candidates for introduction into the model. Indeed, we are guaranteed that, if θ_k has a positive gradient, introducing f_k into $\mathcal{F}$ will result in some improvement to the score. The problem with this approach is that it does not attempt to evaluate how large this improvement can be. Perhaps we can increase θ_k only by a small amount before further changes stop improving the score.

An even more precise approximation is to evaluate a change to the model by computing the score obtained in a model where we keep all other parameters fixed. Consider a step where

we introduce or delete a single feature f_k in our model. We can obtain an approximation to the score by evaluating the change in score when we change only the parameter θ_k associated with f_k, keeping all other parameters unchanged. To formalize this idea, let $(\mathcal{F}, \boldsymbol{\theta}^0)$ be our current model, and consider changes involving f_k. We define the *gain heuristic* estimate to be the change in the score of the model for different values for θ_k, assuming the other parameters are kept fixed:

gain heuristic

$$\Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}) = \text{score}_X((\theta_k, \boldsymbol{\theta}^0_{-k}) \ : \ \mathcal{D}) - \text{score}_X(\boldsymbol{\theta}^0 \ : \ \mathcal{D}), \tag{20.33}$$

where $\boldsymbol{\theta}^0_{-k}$ is the vector of all parameters other than θ^0_k. As we discussed, due to the nondecomposability of the likelihood, when we change θ_k, the current assignment θ^0_{-k} to the other parameters is generally no longer optimal. However, it is still reasonable to use this function as a heuristic to rank different steps: Parameters that give rise to a larger improvement in the objective by themselves often also induce a larger improvement when other parameters are optimized. Indeed, changing those other parameters to optimize the score can only improve it further. Thus, the change in score that we obtain when we "freeze" the other parameters and change only θ_k is a lower bound on the change in the score.

The gain function can be used to provide a lower bound on the improvement in score derived from the deletion of a feature f_k currently in the model: we simply evaluate the gain function setting $\theta_k = 0$. We can also obtain a lower bound on the value of a step where we introduce into the model a new feature f_k (that is, one for which the current parameter $\theta^0_k = 0$). The improvement we can get if we freeze all parameters but one is clearly a lower bound on the improvement we can get if we optimize over all of the parameters. Thus, the value of $\Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D})$ is a lower bound on the improvement in the objective that can be gained by setting θ_k to its chosen value and optimizing all other parameters. To compute the best lower bound, we must maximize the function relative to different possible values of θ_k, giving us the score of the best possible model when all parameters other than θ_k are frozen. In particular, we can define:

$$\text{Gain}_X(\boldsymbol{\theta}^0 \ : \ f_k, \mathcal{D}) = \max_{\theta_k} \Delta^{\text{gain}}_X(\theta_k \ : \ \boldsymbol{\theta}^0, \mathcal{D}).$$

This is a lower bound on the change in the objective obtained from introducing a $f_k \notin \mathcal{F}$.

In principle, lower bounds are more useful than simple approximations. If our lower bound of the candidate's score is higher than that of our current best model, then we definitely want to evaluate that candidate; this will result in a better current candidate and allow us to prune additional candidates and focus on the ones that seem more promising for evaluation. Upper bounds are useful as well. If we have a candidate model and obtain an upper bound on its score, then we can remove it from consideration once we evaluate another candidate with higher score; thus, upper bounds help us prune models for which we would never want to evaluate the true score. In practice, however, fully evaluating the score for all but a tiny handful of candidate structures is usually too expensive a proposition. Thus, the gain is generally used simply as an approximation rather than a lower bound.

How do we evaluate the gain function efficiently, or find its optimal value? The gain function is a univariate function of θ_k, which is a projection of the score function onto this single dimension. Importantly, all of our scoring functions — including any of the penalized likelihood functions we described — are concave (for a given set of active features). The projection

of a concave function onto a single dimension is also concave, so that this single-parameter delta-score is also concave and therefore has a global optimum.

Nevertheless, given the complexity of the likelihood function, it is not clear how this global optimum can be efficiently found. We now show how the difference between two log-likelihood terms can be considerably simplified, even allowing a closed-form solution in certain important cases. Recall from equation (20.3) that

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) = \sum_k \theta_k \boldsymbol{E}_{\mathcal{D}}[f_k] - \ln Z(\boldsymbol{\theta}).$$

Because parameters other than θ_k are the same in the two models, we have that:

$$\frac{1}{M}[\ell((\theta_k, \boldsymbol{\theta}^0_{-k}) : \mathcal{D}) - \ell(\boldsymbol{\theta}^0 : \mathcal{D})] = (\theta_k - \theta^0_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \left[\ln Z(\theta_k, \boldsymbol{\theta}^0_{-k}) - \ln Z(\boldsymbol{\theta}^0)\right].$$

The first term is a linear function in θ_k, whose coefficient is the empirical expectation of f_k in the data. For the second term, we have:

$$\begin{aligned}
\ln \frac{Z(\theta_k, \boldsymbol{\theta}^0_{-k})}{Z(\boldsymbol{\theta}^0)} &= \ln \left[\frac{1}{Z(\boldsymbol{\theta}^0)} \sum_{\xi} \exp\left\{ \sum_j \theta^0_j f_j(\xi) + (\theta_k - \theta^0_k) f_k(\xi) \right\} \right] \\
&= \ln \sum_{\xi} \frac{\tilde{P}_{\boldsymbol{\theta}^0}(\xi)}{Z(\boldsymbol{\theta}^0)} \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\xi)\right\}\right] \\
&= \ln \boldsymbol{E}_{\boldsymbol{\theta}^0}\left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right].
\end{aligned}$$

Thus, the difference of these two log-partition functions can be rewritten as a log-expectation relative to our original distribution. We can convert this expression into a univariate function of θ_k by computing (via inference in our current model $\boldsymbol{\theta}^0$) the marginal distribution over the variables $\boldsymbol{d}_k$. Altogether, we obtain that:

$$\begin{aligned}
&\frac{1}{M}[\ell((\theta_k, \boldsymbol{\theta}^0_{-k}) : \mathcal{D}) - \ell(\boldsymbol{\theta}^0 : \mathcal{D})] = \\
&\quad (\theta_k - \theta^0_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \ln \sum_{\boldsymbol{d}_k} P_{\boldsymbol{\theta}^0}(\boldsymbol{d}_k) \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right].
\end{aligned} \tag{20.34}$$

We can incorporate this simplified form into equation (20.33) for any penalized-likelihood scoring function. We can now easily provide our lower-bound estimates for feature deletions. For introducing a feature f_k, the optimal lower bound can be computed by optimizing the univariate function defined by equation (20.33) over the parameter θ_k. Because this function is concave, it can be optimized using a variety of univariate numerical optimization algorithms. For example, to compute the lower bound for an L_2-regularized likelihood, we would compute:

$$\max_{\theta_k} \left\{ \theta_k \boldsymbol{E}_{\mathcal{D}}[f_k] - \ln \sum_{\boldsymbol{d}_k} P_{\boldsymbol{\theta}^0}(\boldsymbol{d}_k) \left[\exp\left\{(\theta_k - \theta^0_k) f_k(\boldsymbol{d}_k)\right\}\right] - \frac{\theta_k^2}{2\sigma^2} \right\}.$$

However, in certain special cases, we can actually provide a closed-form solution for this optimization problem. We note that this derivation applies only in restricted cases: only in the case of generative training (that is, not for CRFs); only for the likelihood or L_1-penalized objective; and only for binary-valued features.

Proposition 20.6 *Let f_k be a binary-valued feature and let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model. Let $\hat{p}_k = \boldsymbol{E}_{\mathcal{D}}[f_k]$ be the empirical probability of f_k in $\mathcal{D}$, and $p_k^0 = P_{\boldsymbol{\theta}}(f_k)$ be its probability relative to the current model. Then:*

$$\max_{\theta_k} \left[\text{score}_L((\theta_k, \boldsymbol{\theta}^0_{-k}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{M}(\hat{p}_k \| p_k^0),$$

where the KL-divergence is the relative entropy between the two Bernoulli distributions parameterized by $\hat{p}_k$ and p_k^0 respectively.

The proof is left as an exercise (exercise 20.16).

To see why this result is intuitive, recall that when we maximize the likelihood relative to some log-linear model, we obtain a model where the expected counts match the empirical counts. In the case of a binary-valued feature, in the optimized model we have that the final probability of f_k would be the same as the empirical probability $\hat{p}_k$. Thus, it is reasonable that the amount of improvement we obtain from this optimization is a function of the discrepancy between the empirical probability of the feature and its probability given the current model. The bigger the discrepancy, the bigger the improvement in the likelihood.

A similar analysis applies when we consider several binary-valued features $f_1, \ldots, f_k$, as long as they are mutually exclusive and exhaustive; that is, as long as there are no assignments for which both $f_{i'}(\xi) = 1$ and $f_j(\xi) = 1$. In particular, we can show the following:

Proposition 20.7 *Let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model, and consider introducing into the model a complete factor ϕ over scope $\boldsymbol{d}$, parameterized with $\boldsymbol{\theta_d}$ that correspond to the different assignments to $\boldsymbol{d}$. Then*

$$\max_{\boldsymbol{\theta_d}} \left[\text{score}_L((\theta_{\boldsymbol{d}}, \boldsymbol{\theta}^0_{-\boldsymbol{d}}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{M}(\hat{P}(\boldsymbol{d}) \| P_{\boldsymbol{\theta}}(\boldsymbol{d})).$$

The proof is left as an exercise (exercise 20.17).

Although the derivations here were performed for the likelihood function, a similar closed-form solution, in the same class of cases, can also be performed for the L_1-regularized likelihood (see exercise 20.18), but not for the L_2-regularized likelihood. Intuitively, the penalty in the L_1-regularized likelihood is a linear function in each θ_k, and therefore it does not complicate the form of equation (20.34), which already contains such a term. However, the L_2 penalty is quadratic, and introducing a quadratic term into the function prevents an analytic solution.

One issue that we did not address is the task of computing the expressions in equation (20.34), or even the closed-form expressions in proposition 20.6 and proposition 20.7. All of these expressions involve expectations over the scope $\boldsymbol{D}_k$ of f_k, where f_k is the feature that we want to eliminate from or introduce into the model. Let us consider first the case where f_k is already in the model. In this case, if we use a *belief propagation* algorithm (whether a clique tree or a loopy cluster graph), the family preservation property guarantees that the feature's scope $\boldsymbol{D}_k$ is necessarily a subset of some cluster in our inference data structure. Thus, we can easily compute the necessary expectations. However, for a feature not currently in the model, we would not generally expect its scope to be included in any cluster. If not, we must somehow compute expectations of sets of variables that are not together in the same cluster. In the case of clique trees, we can use the out-of-clique inference methods described in section 10.3.3.2. For the case of loopy cluster graphs, this problem is more challenging (see exercise 11.22).

20.8 Summary

In this chapter, we discussed the problem of learning undirected graphical models from data. The key challenge in learning these models is that the global partition function couples the parameters, with significant consequences: There is no closed-form solution for the optimal parameters; moreover, we can no longer optimize each of the parameters independently of the others. Thus, even simple maximum-likelihood parameter estimation is no longer trivial. For the same reason, full Bayesian estimation is computationally intractable, and even approximations are expensive and not often used in practice.

Following these pieces of bad news, there are some good ones: the likelihood function is concave, and hence it has no local optima and can be optimized using efficient gradient-based methods over the space of possible parameterizations. We can also extend this method to MAP estimation when we are given a prior over the parameters, which allows us to reduce the overfitting to which maximum likelihood is prone.

The gradient of the likelihood at a point $\boldsymbol{\theta}$ has a particularly compelling form: the gradient relative to the parameter θ_i corresponding to the feature f_i is the difference between the empirical expectation of f_i in the data and its expectation relative to the distribution $P_{\boldsymbol{\theta}}$. While very intuitive and simple in principle, the form of the gradient immediately gives rise to some bad news: to compute the gradient at the point $\boldsymbol{\theta}$, we need to run inference over the model $P_{\boldsymbol{\theta}}$, a costly procedure to execute at every gradient step.

This complexity motivates the use of myriad alternative approaches: ones involving the use of approximate inference for computing the gradient; and ones that utilize a different objective than the likelihood. Methods in the first class included using message passing algorithms such as belief propagation, and methods based on sampling. We also showed that many of the methods that use approximate inference for optimizing the likelihood can be reformulated as exactly optimizing an approximate objective. This perspective can offer significant insight. For example, we showed that learning with belief propagation can be reformulated as optimizing a joint objective that involves both inference and learning; this alternative formulation is more general and allows the use of alternative optimization methods that are more stable and convergent than using BP to estimate the gradient.

Methods that use an approximate objective include pseudolikelihood, contrastive divergence, and maximum-margin (which is specifically geared for discriminative training of conditional models). Importantly, both likelihood and these objectives can be viewed as trying to increase the distance between the log-probability of assignments in our data and those of some set other assignments. This "contrastive" view provides a different view of these objectives, and it suggests that they are only representatives of a much more general class of approximations.

The same analysis that we performed for optimizing the likelihood can also be extended to other cases. In particular, we showed a very similar derivation for conditional training, where the objective is to maximize the likelihood of a set of target variables $\boldsymbol{Y}$ given some set of observed feature variables $\boldsymbol{X}$.

We also showed that similar approaches can be applied to learning with missing data. Here, the optimization task is no longer convex, but the gradient has a very similar form and can be optimized using the same gradient-ascent methods. However, as in the case of Bayesian network learning with missing data, the likelihood function is generally multimodal, and so the gradient ascent algorithm can get stuck in local optima. Thus, we may need to resort to techniques such

as data perturbation or random restarts.

We also discussed the problem of structure learning of undirected models. Here again, we can use both constraint-based and score-based methods. Owing to the difficulties arising from the form of the likelihood function, full Bayesian scoring, where we score a model by integrating over all of the parameters, is intractable, and even approximations are generally impractical. Thus, we generally use a simpler scoring function, which combines a likelihood term (measuring fit to data) with some penalty term. We then search over some space of structures for ones that optimize this objective. For most objectives, the resulting optimization problem is combinatorial with multiple local optima, so that we must resort to heuristic search. One notable exception is the use of an L_1-regularized likelihood, where the penalty on the absolute value of the parameters tends to drive many of the parameters to zero, and hence often results in sparse models. This objective allows the structure learning task to be formulated as a convex optimization problem over the space of parameters, allowing the optimization to be performed efficiently and with guaranteed convergence to a global optimum. Of course, even here inference is still an unavoidable component in the inner loop of the learning algorithm, with all of the ensuing difficulties.

As we mentioned, the case of discriminative training is a setting where undirected models are particularly suited, and are very commonly used. However, it is important to carefully weigh the trade-offs of generative versus discriminative training. As we discussed, there are significant differences in the computational cost of the different forms of training, and the trade-off can go either way. More importantly, as we discussed in section 16.3.2, **generative models incorporate a higher bias by making assumptions — ones that are often only approximately correct — about the underlying distribution. Discriminative models make fewer assumptions, and therefore tend to require more data to train; generative models, due to the stronger bias, often perform better in the sparse-data regime. But incorrect modeling assumptions also hurt performance; therefore, as the amount of training data grows, the discriminative model, which makes fewer assumptions, often performs better. This difference between the two classes of models is particularly significant when we have complex features whose correlations are hard to model.** However, it is important to remember that models trained discriminatively to predict $\boldsymbol{Y}$ given $\boldsymbol{X}$ will perform well primarily in this setting, and even slight changes may lead to a degradation in performance. For example, a model for predicting $P(Y \mid X_1, X_2)$ would not be useful for predicting $P(Y \mid X_1)$ in situations where X_2 is not observed. In general, discriminative models are much less flexible in their ability to handle missing data.

We focused most of our discussion of learning on the problem of learning log-linear models defined in terms of a set of features. Log-linear models are a finer-grained representation than a Markov network structure or a set of factors. Thus, they can make better trade-offs between model complexity and fit to data. However, sparse log-linear models (with few features) do not directly correspond to sparse Markov network structures, so that we might easily end up learning a model that does not lend itself to tractable inference. It would be useful to consider the development of Markov network structure learning algorithms that more easily support efficient inference. Indeed, some work has been done on learning Markov networks of bounded tree-width, but networks of low tree-width are often poor approximations to the target distribution. Thus, it would be interesting to explore alternative approaches that aim at structures that support approximate inference.

This chapter is structured as a core idea with a set of distinct extensions that build on it: The core idea is the use of the likelihood function and the analysis of its properties. The extensions include conditional likelihood, learning with missing data, the use of parameter priors, approximate inference and/or approximate objectives, and even structure learning. In many cases, these extensions are orthogonal, and we can easily combine them in various useful ways. For example, we can use parameter priors with conditional likelihood or in the case of missing data; we can also use them with approximate methods such as pseudolikelihood, contrastive divergence or in the objective of equation (20.15). Perhaps more surprising is that we can easily perform structure learning with missing data by adding an L_1-regularization term to the likelihood function of equation (20.8) and then using the same ideas as in section 20.7.4.4. In other cases, the combination of the different extensions is more involved. For example, as we discussed, structure learning requires that we be able to evaluate the expected counts for variables that are not in the same family; this task is not so easy if we use an approximate algorithm such as belief propagation. As another example, it is not immediately obvious how we can extend the pseudolikelihood objective to deal with missing data. These combinations provide useful directions for future work.

20.9 Relevant Literature

iterative proportional scaling

iterative proportional fitting

Log-linear models and contingency tables have been used pervasively in a variety of communities, and so key ideas have often been discovered multiple times, making a complete history too long to include. Early attempts for learning log-linear models were based on the *iterative proportional scaling* algorithm and its extension, *iterative proportional fitting*. These methods were first developed for contingency tables by Deming and Stephan (1940) and applied to log-linear models by Darroch and Ratcliff (1972). The convex duality between the maximum likelihood and maximum entropy problems appears to have been proved independently in several papers in diverse communities, including (at least) Ben-Tal and Charnes (1979); Dykstra and Lemke (1988); Berger, Della-Pietra, and Della-Pietra (1996). It appears that the first application of gradient algorithms to maximum likelihood estimation in graphical models is due to Ackley, Hinton, and Sejnowski (1985) in the context of Boltzmann machines. The importance of the method used to optimize the likelihood was highlighted in the comparative study of Minka (2001a); this study focused on learning for logistic regression, but many of the conclusions hold more broadly. Since then, several better methods have been developed for optimizing likelihood. Successful methods include conjugate gradient, L-BFGS (Liu and Nocedal 1989), and stochastic meta-descent (Vishwanathan et al. 2006).

Conditional random fields were first proposed by Lafferty et al. (2001). They have since been applied in a broad range of applications, such as labeling multiple webpage on a website (Taskar et al. 2002), image segmentation (Shental et al. 2003), or information extraction from text (Sutton and McCallum 2005). The application to protein-structure prediction in box 20.B is due to Yanover et al. (2007).

The use of approximate inference in learning is an inevitable consequence of the intractability of the inference problem. Several papers have studied the interaction between belief propagation and Markov network learning. Teh and Welling (2001) and Wainwright et al. (2003b) present methods for certain special cases; in particular, Wainwright, Jaakkola, and Willsky (2003b) derive

the pseudo-moment matching argument. Inspired by the moment-matching behavior of learning with belief propagation, Sutton and McCallum (2005); Sutton and Minka (2006) define the piecewise training objective that directly performs moment matching on all network potentials. Wainwright (2006) provides a strong argument, both theoretical and empirical, for using the same approximate inference method in training as will be used in performing the prediction using the learned model. Indeed, he shows that, if an approximate method is used for inference, then we get better performance guarantees if we use that same method to train the model than if we train a model using exact inference. He also shows that it is detrimental to use an unstable inference algorithm (such as sum-product BP) in the inner loop of the learning algorithm. Ganapathi et al. (2008) define the unified CAMEL formulation that encompasses learning and inference in a single joint objective, allowing the nonconvexity of the BP objective to be taken out of the inner loop of learning.

Although maximum (conditional) likelihood is the most commonly used objective for learning Markov networks, several other objectives have been proposed. The earliest is pseudolikelihood, proposed by Besag (1977b), of which several extensions have been proposed (Huang and Ogata 2002; McCallum et al. 2006). The asymptotic consistency of both the likelihood and the pseudolikelihood objectives is shown by Gidas (1988). The statistical efficiency (convergence as a function of the number of samples) of the pseudolikelihood estimator has also been analyzed (for example, (Besag 1977a; Geyer and Thompson 1992; Guyon and Künsch 1992; Liang and Jordan 2008)).

The use of margin-based estimation methods for probabilistic models was first proposed by Collins (2002) in the context of parsing and sequence modeling, building on the voted-perceptron algorithm (Freund and Schapire 1998). The methods described in this chapter build on a class of large-margin methods called *support vector machines* (Shawe-Taylor and Cristianini 2000; Hastie et al. 2001; Bishop 2006), which have the important benefit of allowing a large or even infinite feature space to be used and trained very efficiently. This formulation was first proposed by Altun, Tsochantaridis, and Hofmann (2003); Taskar, Guestrin, and Koller (2003), who proposed two different approaches for addressing the exponential number of constraints. Altun et al. use a constraint-generation scheme, which was subsequently proven to require at most a polynomial number of steps (Tsochantaridis et al. 2004). Taskar et al. use a closed-form polynomial-size reformulation of the optimization problem that uses a clique tree-like data structure. Taskar, Chatalbashev, and Koller (2004) also show that this formulation also allows tractable training for networks where conditional probability products are intractable, but the MAP assignment can be found efficiently. The contrastive divergence approach was introduced by Hinton (2002); Teh, Welling, Osindero, and Hinton (2003), and was shown to work well in practice in various studies (for example, (Carreira-Perpignan and Hinton 2005)). This work forms part of a larger trend of training using a range of alternative, often contrastive, objectives. LeCun et al. (2007) provide an excellent overview of this area.

support vector machine

Much discussion has taken place in the machine learning community on the relative merits of discriminative versus generative training. Some insightful papers of particular relevance to graphical models include the work of Minka (2005) and LeCun et al. (2007). Also of interest are the theoretical analyses of Ng and Jordan (2002) and Liang and Jordan (2008) that discuss the statistical efficiency of discriminative versus generative training, and provide theoretical support for the empirical observation that generative models, even if not consistent with the true underlying distribution, often work better in the sparse data case, but discriminative models

tend to work better as the amount of data grows.

The work of learning Markov networks with hidden variables goes back to the seminal paper of Ackley, Hinton, and Sejnowski (1985), who used gradient ascent to train Boltzmann machines with hidden variables. This line of work, largely dormant for many years, has seen a resurgence in the work on *deep belief networks* (Hinton et al. 2006; Hinton and Salakhutdinov 2006), a training regime for a multilayer restricted Boltzmann machine that iteratively tries to learn deeper and deeper hidden structure in the data.

deep belief networks

Parameter priors and regularization methods for log-linear models originate in statistics, where they have long been applied to a range of statistical models. Many of the techniques described here were first developed for traditional statistical models such as linear or logistic regression, and then extended to the general case of Markov networks and CRFs. See Hastie et al. (2001) for some background on this extensive literature.

The problem of learning the structure of Markov networks has not received as much attention as the task of Bayesian network structure learning. One line of work has focused on the problem of learning a Markov network of bounded tree-width, so as to allow tractable inference. The work of Chow and Liu (1968) shows that the maximum-likelihood tree-structured network can be found in quadratic time.

A tree is a network of tree-width 1. Thus, the obvious generalization of is to learning the class of Markov networks whose tree-width is at most k. Unfortunately, there is a sharp threshold phenomenon, since Srebro (2001) proves that for any tree-width k greater than 1, finding the maximum likelihood tree-width-k network is $\mathcal{NP}$-hard. Interestingly, Narasimhan and Bilmes (2004) provide a constraint-based algorithm for PAC-learning Markov networks of tree-width at most k: Their algorithm is guaranteed to find, with probability $1 - \delta$, a network whose relative entropy is within ϵ of optimal, in polynomial time, and using a polynomial number of samples. Importantly, their result does not contradict the hardness result of Srebro, since their analysis applies only in the consistent case, where the data is derived from a k-width network. This discrepancy highlights again the significant difference between learnability in the consistent and the inconsistent case. Several search-based heuristic algorithms for learning models with small tree-width have been proposed (Bach and Jordan 2001; Deshpande et al. 2001); so far, none of these algorithms have been widely adopted, perhaps because of the limited usefulness of bounded tree-width networks.

Abbeel, Koller, and Ng (2006) provide a different PAC-learnability result in the consistent case, for networks of bounded connectivity. Their constraint-based algorithm is guaranteed to learn, with high probability, a network $\tilde{\mathcal{M}}$ whose (symmetric) relative entropy to the true distribution ($\boldsymbol{D}(\tilde{P}\|P^*) + \boldsymbol{D}(P^*\|\tilde{P})$) is at most ϵ. The complexity, both in time and in number of samples, grows exponentially in the maximum number of assignments to any local neighborhood (a factor and its Markov blanket). This result is somewhat surprising, since it shows that the class of low-connectivity Markov networks (such as grids) is PAC-learnable, even though inference (including computing the partition function) can be intractable.

Of highest impact has been the work on using local search to optimize a (regularized) likelihood. This line of work originated with the seminal paper of Della Pietra, Della Pietra, and Lafferty (1997), who defined the single-feature gain, and proposed the gain as an effective heuristic for feature selection in learning Markov network structure. McCallum (2003) describes some heuristic approximations that allow this heuristic to be applied to CRFs. The use of L_1-regularization for feature selection originates from the Lasso model proposed for linear re-

gression by Tibshirani (1996). It was first proposed for logistic regression by Perkins et al. (2003); Goodman (2004). Perkins et al. also suggested the gradient heuristic for feature selection and the L_1-based stopping rule. L_1-regularized priors were first proposed for learning log-linear distributions by Riezler and Vasserman (2004); Dudík, Phillips, and Schapire (2004). The use of L_1-regularized objectives for learning the structure of general Markov networks was proposed by Lee et al. (2006). Building on the results of Dudík et al., Lee et al. also showed that the number of samples required to achieve close-to-optimal relative entropy (within the target class) grows only polynomially in the size of the network. Importantly, unlike the PAC-learnability results mentioned earlier, this result also holds in the inconsistent case.

Pseudolikelihood has also been used as a criterion for model selection. Ji and Seymour (1996) define a pseudolikelihood-based objective and show that it is asymptotically consistent, in that the probability of selecting an incorrect model goes to zero as the number of training examples goes to infinity. However, they did not provide a tractable algorithm for finding the highest scoring model in the superexponentially large set of structures. Wainwright et al. (2006) suggested the use of an L_1-regularized pseudolikelihood for model selection, and also proved a theorem that provides guarantees on the near-optimality of the learned model, using a polynomial number of samples. Like the result of Lee et al. (2006), this result applies also in the inconsistent case.

This chapter has largely omitted discussion of the Bayesian learning approach for Markov networks, for both parameter estimation and structure learning. Although an exact approach is computationally intractable, some interesting work has been done on approximate methods. Some of this work uses MCMC methods to sample from the parameter posterior. Murray and Ghahramani (2004) propose and study several diverse methods; owing to the intractability of the posterior, all of these methods are approximate, in that their stationary distribution is only an approximation to the desired parameter posterior. Of these, the most successful methods appear to be a method based on Langevin sampling with approximate gradients given by contrastive divergence, and a method where the acceptance probability is approximated by replacing the log partition function with the Bethe free energy. Two more restricted methods (Møller et al. 2006; Murray et al. 2006) use an approach called "perfect sampling" to avoid the need for estimating the partition function; these methods are elegant but of limited applicability. Other approaches approximate the parameter posterior by a Gaussian distribution, using either expectation propagation (Qi et al. 2005) or a combination of a Bethe and a Laplace approximation (Welling and Parise 2006a). The latter approach was also used to approximate the Bayesian score in order to perform structure learning (Welling and Parise 2006b). Because of the fundamental intractability of the problem, all of these methods are somewhat complex and computationally expensive, and they have therefore not yet made their way into practical applications.

20.10 Exercises

Exercise 20.1

Consider the network of figure 20.2, where we assume that some of the factors share parameters. Let $\boldsymbol{\theta}_i^y$ be the parameter vector associated with all of the features whose scope is Y_i, Y_{i+1}. Let $\boldsymbol{\theta}_{i,j}^{xy}$ be the parameter vector associated with all of the features whose scope is Y_i, X_j.

a. Assume that, for all i, i', $\boldsymbol{\theta}_i^y = \boldsymbol{\theta}_{i'}^y$, and that for all i, i' and j, $\boldsymbol{\theta}_{i,j}^{xy} = \boldsymbol{\theta}_{i',j}^{xy}$. Derive the gradient update for this model.

b. Now (without the previous assumptions) assume for all i and j, j', $\boldsymbol{\theta}^{xy}_{i,j} = \boldsymbol{\theta}^{xy}_{i,j'}$. Derive the gradient update for this model.

Exercise 20.2

relational Markov network

In this exercise, we show how to learn Markov networks with shared parameters, such as a *relational Markov network* (RMN).

a. Consider the log-linear model of example 6.18, where we assume that the *Study-Pair* relationship is determined in the relational skeleton. Thus, we have a single template feature, with a single weight, which is applied to all study pairs. Derive the likelihood function for this model, and the gradient.
b. Now provide a formula for the likelihood function and the gradient for a general RMN, as in definition 6.14.

Exercise 20.3

Assume that our data are generated by a log-linear model $P_{\boldsymbol{\theta}^*}$ that is of the form of equation (20.1). Show that, as the number of data instances M goes to infinity, with probability that approaches 1, $\boldsymbol{\theta}^*$ is a global optimum of the likelihood objective of equation (20.3). (Hint: Use the characterization of theorem 20.1.)

Exercise 20.4

Use the techniques described in this chapter to provide a method for performing maximum likelihood estimation for a CPD whose parameterization is a generalized linear model, as in definition 5.10.

Exercise 20.5⋆

Show using Lagrange multipliers and the definitions of appendix A.5.4 that the problem of maximizing $\boldsymbol{H}_Q(\mathcal{X})$ subject to equation (20.10) is dual to the problem of maximizing the log likelihood $\max \ell(\boldsymbol{\theta} : \mathcal{D})$.

Exercise 20.6⋆

In this problem, we will show an analogue to theorem 20.2 for the problems of maximizing conditional likelihood and maximizing conditional entropy.

Consider a data set $\mathcal{D} = \{(\boldsymbol{y}[m], \boldsymbol{x}[m])\}_{m=1}^M$ as in section 20.3.2, and define the following conditional entropy maximization problem:

Maximum-Conditional-Entropy:

Find	$Q(\boldsymbol{Y} \mid \boldsymbol{X})$
maximizing	$\sum_{m=1}^M \boldsymbol{H}_Q(\boldsymbol{Y} \mid \boldsymbol{x}[m])$
subject to	

$$\sum_{m=1}^M \boldsymbol{E}_{Q(\boldsymbol{Y} \mid \boldsymbol{x}[m])}[f_k] = \sum_{m=1}^M f_k(\boldsymbol{y}[m], \boldsymbol{x}[m]) \quad i = 1, \ldots, k. \tag{20.35}$$

Show that $Q^*(\boldsymbol{Y} \mid \boldsymbol{X})$ optimizes this objective if and only if $Q^* = P_{\hat{\boldsymbol{\theta}}}$ where $P_{\hat{\boldsymbol{\theta}}}$ maximizes $\ell_{\boldsymbol{Y} \mid \boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D})$ as in equation (20.6).

Exercise 20.7⋆

iterative proportional scaling

One of the earliest approaches for finding maximum likelihood parameters is called *iterative proportional scaling* (IPS). The idea is essentially to use coordinate ascent to improve the match between the empirical feature counts and the expected feature counts. In other words, we change θ_k so as to make $\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_k]$ closer to $\boldsymbol{E}_{\mathcal{D}}[f_k]$. Because our model is multiplicative, it seems natural to multiply the weight of instances where $f_k(\xi) = 1$ by the ratio between the two expectations. This intuition leads to the following update rule:

$$\theta'_k \leftarrow \theta_k + \ln \frac{\boldsymbol{E}_{\mathcal{D}}[f_k]}{\boldsymbol{E}_{P_\theta}[f_k]}. \tag{20.36}$$

The IPS algorithm iterates over the different parameters and updates each of them in turn, using this update rule.

Somewhat surprisingly, one can show that each iteration increases the likelihood until it reaches a maximum point. Because the likelihood function is concave, there is a single maximum, and the algorithm is guaranteed to find it.

Theorem 20.5

Let $\boldsymbol{\theta}$ be a parameter vector, and $\boldsymbol{\theta}'$ the vector that results from it after an application of equation (20.36). Then $\ell(\boldsymbol{\theta}' : \mathcal{D}) \geq \ell(\boldsymbol{\theta} : \mathcal{D})$ with equality if only if $\frac{\partial}{\partial \theta_k} \ell(\boldsymbol{\theta}^t : \mathcal{D}) = 0$.

In this exercise, you will prove this theorem for the special case where f_k is binary-valued. More precisely, let $\Delta(\theta_k)$ denote the change in likelihood obtained from modifying a single parameter θ_k, keeping the others fixed. This expression was computed in equation (20.34). You will now show that the IPS update step for θ_k maximizes a lower bound on this single parameter gain.

Define

$$\begin{aligned}\tilde{\Delta}(\theta_k) &= (\theta'_k - \theta_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \frac{Z(\boldsymbol{\theta}')}{Z(\boldsymbol{\theta})} + 1 \\ &= (\theta'_k - \theta_k)\boldsymbol{E}_{\mathcal{D}}[f_k] - \boldsymbol{E}_{P_{\boldsymbol{\theta}}}[1 - f_k] - e^{\theta'_k - \theta_k}\boldsymbol{E}_{P_{\boldsymbol{\theta}'}}[f_k] + 1.\end{aligned}$$

a. Show that $\Delta(\theta'_k) \geq \tilde{\Delta}(\theta'_k)$. (Hint: use the bound $\ln(x) \leq x - 1$.)

b. Show that $\theta_k + \ln \frac{\boldsymbol{E}_{\mathcal{D}}[f_k]}{\boldsymbol{E}_{P_{\boldsymbol{\theta}}}[f_k]} = \arg\max_{\theta'_k} \tilde{\Delta}(\theta'_k)$.

c. Use these two facts to conclude that IPS steps are monotonically nondecreasing in the likelihood, and that convergence is achieved only when the log-likelihood is maximized.

d. This result shows that we can view IPS as performing coordinatewise ascent on the likelihood surface. At each iteration we make progress along on dimension (one parameter) while freezing the others. Why is coordinate ascent a wasteful procedure in the context of optimizing the likelihood?

Exercise 20.8

hyperbolic prior

Consider the following *hyperbolic prior* for parameters in log-linear models.

$$P(\theta) = \frac{1}{(e^{\theta} + e^{-\theta})/2}.$$

a. Derive a gradient-based update rule for this parameter prior.

b. Qualitatively describe the expected behavior of this parameter prior, and compare it to those of the L_2 or L_1 priors discussed in section 20.4. In particular, would you expect this prior to induce sparsity?

Exercise 20.9

piecewise training

We now consider an alternative local training method for Markov networks, known as *piecewise training*. For simplicity, we focus on Markov networks parameterized via full table factors. Thus, we have a set of factors $\phi_c(\boldsymbol{X}_c)$, where $\boldsymbol{X}_c$ is the scope of factor c, and $\phi_c(\boldsymbol{x}_c^j) = \exp(\theta_{cj})$. For a particular parameter assignment $\boldsymbol{\theta}$, we define $Z_c(\boldsymbol{\theta})$ to be the local partition function for this factor in isolation:

$$Z_c(\boldsymbol{\theta}_c) = \sum_{\boldsymbol{x}_c} \phi_c(\boldsymbol{x}_c),$$

where $\boldsymbol{\theta}_c$ is the parameter vector associated with the factor $\phi_c(\boldsymbol{X}_c)$. We can approximate the global partition function in the log-likelihood objective of equation (20.3) as a product of the local partition functions, replacing $Z(\theta)$ with $\prod_c Z_c(\boldsymbol{\theta}_c)$.

a. Write down the form of the resulting objective, simplify it, and derive the assignment of parameters that optimizes it.
b. Compare the result of this optimization to the result of the pseudo-moment matching approach described in section 20.5.1.

Exercise 20.10⋆

CAMEL

In this exercise, we analyze the following simplification of the *CAMEL* optimization problem of equation (20.15):

Simple-Approx-Maximum-Entropy:

Find $\boldsymbol{Q}$
maximizing $\sum_{\boldsymbol{C}_i \in \mathcal{U}} \boldsymbol{H}_{\beta_i}(\boldsymbol{C}_i)$
subject to

$$\begin{aligned} \boldsymbol{E}_{\beta_i}[f_i] &= \boldsymbol{E}_{\mathcal{D}}[f_i] \quad i = 1, \ldots, k \\ \sum_{\boldsymbol{c}_i} \beta_i(\boldsymbol{c}_i) &= 1 \quad i = 1, \ldots, k \\ \boldsymbol{Q} &\geq 0 \end{aligned}$$

Here, we approximate both the objective and the constraints. The objective is approximated by the removal of all of the negative entropy terms for the sepsets. The constraints are relaxed by removing the requirement that the potentials in $\boldsymbol{Q}$ be locally consistent (sum-calibrated) — we now require only that they be legal probability distributions.

Show that this optimization problem is the Lagrangian dual of the piecewise training objective in exercise 20.9.

Exercise 20.11⋆

multiconditional training

Consider a setting, as in section 20.3.2, where we have two sets of variables $\boldsymbol{Y}$ and $\boldsymbol{X}$. *Multiconditional training* provides a spectrum between pure generative and pure discriminative training by maximizing the following objective:

$$\alpha \ell_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{\theta} : \mathcal{D}) + (1 - \alpha)\ell_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{\theta} : \mathcal{D}). \tag{20.37}$$

Consider the model structure shown in figure 20.2, and a partially labeled data set $\mathcal{D}$, where in each instance m we observe all of the feature variables $\boldsymbol{x}[m]$, but only the target variables in $\boldsymbol{O}[m]$.

Write down the objective of equation (20.37) for this case and compute its derivative.

Exercise 20.12⋆

Consider the problem of maximizing the approximate log-likelihood shown in equation (20.16).

a. Derive the gradient of the approximate likelihood, and show that it is equivalent to utilizing an importance sampling estimator directly to approximate the expected counts in the gradient of equation (20.4).
b. Characterize properties of the maximum point (when the gradient is 0). Is such a maximum always attainable? Prove or suggest a counterexample.

Exercise 20.13⋆⋆

One approach to providing a lower bound to the log-likelihood is by upper-bounding the partition function. Assume that we can decompose our model as a convex combination of (hopefully) simpler models, each with a weight α_k and a set of parameters ψ^k. We define these submodels as follows: $\psi^k(\boldsymbol{\theta}) = \boldsymbol{w}^k \bullet \boldsymbol{\theta}$, where we require that, for any feature i,

$$\sum_k \alpha_k w_i^k = 1. \tag{20.38}$$

a. Under this assumption, prove that

$$\ln Z(\boldsymbol{\theta}) \leq \sum_k \alpha_k \ln Z(\boldsymbol{w}^k \bullet \boldsymbol{\theta}). \tag{20.39}$$

This result allows us to define an approximate log-likelihood function:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) \geq \ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_i \theta_i \boldsymbol{E}_{\mathcal{D}}[f_i] - \sum_k \alpha_k \ln Z(\boldsymbol{w}^k \bullet \boldsymbol{\theta}).$$

b. Assuming that the submodels are more tractable, we can efficiently evaluate this lower bound, and also compute its derivatives to be used during optimization. Show that

$$\frac{\partial}{\partial \theta_i}\ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \boldsymbol{E}_{\mathcal{D}}[f_i] - \sum_k \alpha_k \boldsymbol{w}_i^k \boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i]. \tag{20.40}$$

c. We can provide a bound on the error of this approximation. Specifically, show that:

$$\frac{1}{M}\ell(\boldsymbol{\theta} : \mathcal{D}) - \ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D}) = \sum_k \alpha_k \boldsymbol{D}(P_{\boldsymbol{\theta}} \| P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}),$$

where the KL-divergence measures are defined in terms of the natural logarithm. Thus, we see that the error is an average of the divergence between the true distribution and each of the approximating submodels.

d. The justification for this approach is that we can make the submodels simpler than the original model by having some parameters be equal to 0, thereby eliminating the resulting feature from the model structure. Other than this constraint, however, we still have considerable freedom in choosing the submodel weight vectors $\boldsymbol{w}^k$. Assume that each weight vector $\{\boldsymbol{w}_k\}$ maximizes $\ell_{\text{convex}}(\boldsymbol{\theta} : \mathcal{D})$ subject to the constraint of equation (20.38) plus additional constraints requiring that certain entries w_i^k be equal to 0. Show that if i, k and l are such that $\theta_i \neq 0$ and neither w_i^k nor w_i^l is constrained to be zero, then

$$\boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i] = \boldsymbol{E}_{P_{\boldsymbol{w}^l \bullet \boldsymbol{\theta}}}[f_i].$$

Conclude from this result that for each such i and k, we have that

$$\boldsymbol{E}_{P_{\boldsymbol{w}^k \bullet \boldsymbol{\theta}}}[f_i] = \boldsymbol{E}_{\mathcal{D}}[f_i].$$

Exercise 20.14

Consider a particular parameterization $(\boldsymbol{\theta}, \eta)$ to Max-margin. Show how we can use second-best MAP inference to either find a violated constraint or guarantee that all constraints are satisfied.

Exercise 20.15

Let $\mathcal{H}^*$ be a Markov network where the maximum degree of a node is d^*. Show that if we have an infinitely large data set $\mathcal{D}$ generated from $\mathcal{H}^*$ (so that independence tests are evaluated perfectly), then the Build-PMap-Skeleton procedure of algorithm 3.3 reconstructs the correct Markov structure $\mathcal{H}^*$.

Exercise 20.16⋆

Prove proposition 20.6. (Hint: Take the derivative of equation (20.34) and set it to zero.)

Exercise 20.17

In this exercise, you will prove proposition 20.7, which allows us to find a closed-form optimum to multiple features in a log-linear model.

a. Prove the following proposition.

Proposition 20.8 *Let $\boldsymbol{\theta}^0$ be a current setting of parameters for a log-linear model, and suppose that $f_1, \ldots, f_l$ are mutually exclusive binary features, that is, there is no ξ and $i \neq j$, so that $f_i(\xi) = 1$ and $f_j(\xi) = 1$. Then,*

$$\max_{\theta_1,\ldots,\theta_l} \left[\text{score}_L((\theta_1, \ldots, \theta_l, \boldsymbol{\theta}^0_{-\{1,\ldots,l\}}) \ : \ \mathcal{D}) - \text{score}_L(\boldsymbol{\theta}^0 \ : \ \mathcal{D})\right] = \boldsymbol{I}\boldsymbol{D}(\hat{\boldsymbol{p}} \| \boldsymbol{p}^0),$$

where $\hat{\boldsymbol{p}}$ is a distribution over $l+1$ values with $\hat{p}_i = \boldsymbol{E}_{\mathcal{D}}[f_i]$, and $\boldsymbol{p}^0$ is a distribution with $p^0(i) = P_{\boldsymbol{\theta}}(f_i)$.

b. Use this proposition to prove proposition 20.7.

Exercise 20.18

Derive an analog to proposition 20.6 for the case of the L_1 regularized log-likelihood objective.

Part IV

Actions and Decisions

21 *Causality*

21.1 Motivation and Overview

So far, we have been somewhat ambivalent about the relation between Bayesian networks and causality. On one hand, from a formal perspective, all of the definitions refer only to probabilistic properties such as conditional independence. The BN structure may be directed, but the directions of the arrows do not have to be meaningful. They can even be antitemporal. Indeed, we saw in our discussion of I-maps that we can take any ordering on the nodes and create a BN for any distribution. On the other hand, it is common wisdom that a "good" BN structure should correspond to causality, in that an edge $X \rightarrow Y$ often suggests that X "causes" Y, either directly or indirectly. The motivation for this statement is pragmatic: Bayesian networks with a causal structure tend to be sparser and more natural. However, as long as the network structure is capable of representing the underlying joint distribution correctly, the answers that we obtain to probabilistic queries are the same, regardless of whether the network structure corresponds to some notion of causal influence.

Given this observation, is there any deeper value to imposing a causal semantics on a Bayesian network? In this chapter, we discuss a type of reasoning for which a causal interpretation of the network is critical — reasoning about situations where we *intervene* in the world, thereby interfering in the natural course of events. For example, we may wish to know if an intervention where we prevent smoking in all public places is likely to decrease the frequency of lung cancer. To answer such queries, we need to understand the causal relationships between the variables in our model.

In this chapter, we provide a framework for interpreting a Bayesian network as a *causal model* whose edges have causal significance. Not surprisingly, this interpretation distinguishes between models that are equivalent in their ability to represent probabilistic correlations. Thus, although the two networks $X \rightarrow Y$ and $Y \rightarrow X$ are equivalent *as probabilistic models*, they will turn out to be very different *as causal models*.

21.1.1 Conditioning and Intervention

As we discussed, for standard probabilistic queries it does not matter whether our model is causal or not. It matters only that it encode the "right" distribution. The difference between causal models and probabilistic models arise when we care about *interventions* in the model — situations where we do not simply observe the values that variables take but can take actions

that can manipulate these values.

In general, actions can affect the world in a variety of ways, and even a single action can have multiple effects. Indeed, in chapter 23, we discuss models that directly incorporate agent actions and allow for a range of effects. In this chapter, however, our goal is to isolate the specific issue of understanding causal relationships between variables. 

ideal intervention

One approach to modeling causal relationships is using the notion of *ideal interventions* — interventions of the form $do(Z := z)$, which force the variable Z to take the value z, and have no other *immediate* effect. An ideal intervention is equivalent to a dedicated action whose only effect is setting Z to z. However, we can consider such an ideal intervention even when such an action does not exist in the world. For example, consider the question of whether a particular mutation in a person's DNA causes a particular disease. This causal question can be formulated as the question of whether an ideal intervention, whose only effect is to generate this mutation in a person's DNA, would lead to the disease. Note that, even if such a process were ethical, current technology does not permit an action whose only effect is to mutate the DNA in all cells of a human organism. However, understanding the causal connection between the mutation and the disease can be a critical step toward finding a cure; the ideal intervention provides us with a way of formalizing this question and trying to provide an answer.

More formally, we consider a new type of "conditioning" on an event of the form $do(Z := z)$, often abbreviated $do(z)$; this information corresponds to settings where an agent directly manipulated the world, to set the variable Z to take the value z with probability 1. We are now interested in answering queries of the form $P(Y \mid do(z))$, or, more generally, $P(\boldsymbol{Y} \mid do(\boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x})$. These queries are called *intervention queries*. They correspond to settings where we set the variables in $\boldsymbol{Z}$ to take the value $\boldsymbol{z}$, observe the values $\boldsymbol{x}$ for the variables in $\boldsymbol{X}$, and wish to find the distribution over the variables $\boldsymbol{Y}$. Such queries arise naturally in a variety of settings:

intervention query

- **Diagnosis and Treatment:** "If we get a patient to take this medication, what are her chances of getting well?" This query can be formulated as $P(H \mid do(M := m^1))$, where H is the patient's health, and $M = m^1$ corresponds to her taking the medication. Note that this query is not the same as $P(H \mid m^1)$. For example, if patients who take the medication on their own are more likely to be health-conscious, and therefore healthier in general, the chances of $P(H \mid m^1)$ may be higher than is warranted for the patient in question.
- **Marketing:** "If we lower the price of hamburgers, will people buy more ketchup?" Once again, this query is not a standard observational query, but rather one in which we intervene in the model, and thereby possibly change its behavior.
- **Policy Making:** "If we lower the interest rates, will that give rise to inflation?"
- **Scientific Discovery:** "Does smoking cause cancer?" When we formalize it, this query is an intervention query, meaning: "If we were to force someone to smoke, would they be more likely to get cancer?"

A different type of causal query arises in situations where we *already* have some information about the true state of the world, and want to inquire about the state the world *would be* in had we been able to intervene and set the values of certain variables. For example, we might want to know "Would the U.S. have joined World War II had it not been for the attack on Pearl Harbor?" Such queries are called *counterfactual queries*, because they refer to a world that we

counterfactual query

know did not happen. Intuitively, our interpretation for such a query is that it refers to a world that differs only in this one respect. Thus, in this *counterfactual* world, Hitler would still have come into power in Germany, Poland would still have been invaded, and more. On the other hand, events that are direct causal consequences of the variable we are changing are clearly going to be different. For example, in the counterfactual world, the USS *Arizona* (which sank in the attack) would not (with high probability) currently be at the bottom of Pearl Harbor.

At first glance, counterfactual analysis might seem somewhat pointless and convoluted (who cares about what would have happened?). However, such queries actually arise naturally in several settings:

- **Legal liability cases:** "Did the driver's intoxicated state cause the accident?" In other words, would the accident have happened had the driver not been drunk? Here, we may want to preserve many other aspects of the world, for example, that it was a rainy night (so the road was slippery).
- **Treatment and Diagnosis:** "We are faced with a car that does not start, but where the lights work; will replacing the battery make the car start?" Note that this is not an intervention query; it is a counterfactual query: we are actually asking whether the car would be working now had we replaced the battery. As in the previous example, we want to preserve as much of our scenario as possible. For example, given our observation that the lights work, the problem probably is not with the battery; we need to account for this conclusion when reasoning about the situation where the battery has been replaced.

Even without a formal semantics for a causal model, we can see that the answer to an intervention query $P(\boldsymbol{Y} \mid do(\boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x})$ is generally quite different from the answer to its corresponding probabilistic query $P(\boldsymbol{Y} \mid \boldsymbol{Z} = \boldsymbol{z}, \boldsymbol{X} = \boldsymbol{x})$.

Example 21.1

Let us revisit our simple Student *example of section 3.1.3.1, and consider a particular student Gump. As we have already discussed, conditioning on an observation that Gump receives an A in the class increases the probability that he has high intelligence, his probability of getting a high SAT score, and his probability of getting a good job.*

By contrast, consider a situation where Gump is lazy, and rather than working hard to get an A in the class, he pays someone to hack into the university registrar's database and change his grade in the course to an A. In this case, what is his probability of getting a good job? Intuitively, the company where Gump is applying only has access to Gump's transcript; thus, the company's response to a manipulated grade would be the same as the response to an authentic grade. Therefore, we would expect $P(J \mid \text{do}(g^1)) = P(J \mid g^1)$. *What about the other two probabilities? Intuitively, we feel that the manipulation to Gump's grade should not affect our beliefs about his intelligence, nor about his SAT score. Thus, we would expect* $P(i^1 \mid \text{do}(g^1)) = P(i^1)$ *and* $P(s^1 \mid \text{do}(g^1)) = P(s^1)$. ■

Why is our response to these queries different? In all three cases, there is a strong correlation between Gump's grade and the variable of interest. However, we perceive the correlation between Gump's grade and his job prospects as being *causal*. Thus, changes to his grade will directly affect his chances of being hired. The correlation between intelligence and grade arises because of an opposite causal connection: intelligence is a causal factor in grade. The correlation between Gump's SAT score and grade arises due to a third mechanism — their joint dependence on Gump's intelligence. Manipulating Gump's grade does not change his intelligence or his chances

of doing well in the class. In this chapter, we describe a formal framework of causal models that provides a rigorous basis for answering such queries and allows us to distinguish between these different cases. As we will see, this framework can be used to answer both intervention and counterfactual queries. However, the latter require much finer-grained information, which may be difficult to acquire in practice.

21.1.2 Correlation and Causation

As example 21.1 illustrates, a correlation between two variables X and Y can arise in multiple settings: when X causes Y, when Y causes X, or when X and Y are both effects of a single cause. This observation immediately gives rise to the question of identifiability of causal models: If we observe two variables X, Y to be probabilistically correlated in some observed distribution, what can we infer about the causal relationship between them. As we saw, different relationships give rise to very different answers to causal queries.

This problem is greatly complicated by the broad range of reasons that may lead to an observed correlation between two variables X and Y. As we saw in example 21.1, when some variable W causally affects both X and Y, we generally observe a correlation between them. If we know about the existence of W and can observe it, we can disentangle the correlation between X and Y that is induced by W and compute the residual correlation between X and Y that may be attributed to a direct causal relationship. **In practice, however, there is a huge set of possible *latent variables*, representing factors that exist in the world but that we cannot observe and often are not even aware of. A latent variable may induce correlations between the observed variables that do not correspond to causal relations between them, and hence forms a *confounding factor* in our goal of determining causal interactions.**

latent variable

confounding factor

As we discussed in section 19.5, when our task is pure probabilistic reasoning, latent variables need not be modeled explicitly, since we can always represent the joint distribution over the observable variables only using a probabilistic graphical model. Of course, this marginalization process can lead to more complicated models (see, for example, figure 16.1), and may therefore be undesirable. We may therefore choose to model certain latent variables explicitly, in order to simplify the resulting network structure. Importantly, however, for the purpose of answering probabilistic queries, we do not need to model all latent variables. As long as our model $\mathcal{B}_{obs}$ over the observable variables allows us to capture exactly the correct marginal distribution over the observed variables, we can answer any query as accurately with $\mathcal{B}_{obs}$ as with the true network, where the latent variables are included explicitly.

However, as we saw, the answer to a causal query over X, Y is quite different when a correlation between them is due to a causal relationship and when it is induced by a latent variable. Thus, for the purposes of causal inference, it is critical to disentangle the component in the correlation between X and Y that is due to causal relationships and the component due to these confounding factors. Unfortunately, this requirement poses a major challenge, since it is virtually impossible, in complex real-world settings, to identify all of the relevant latent variables and quantify their effects.

Example 21.2 *Consider a situation where we observe a significant positive correlation in our patient population between taking PeptAid, an antacid medication (T), and the event of subsequently developing a*

stomach ulcer (O). Because taking PeptAid precedes the ulcer, we might be tempted to conclude that PeptAid causes stomach ulcers. However, an alternative explanation is that the correlation can be attributed to a latent common cause — preulcer discomfort: individuals suffering from preulcer discomfort were more likely to take PeptAid and ultimately more likely to develop ulcers. Even if we account for this latent variable, there are many others that can have a similar effect. For example, some patients who live a more stressful lifestyle may be more inclined to eat irregular meals and therefore more likely to require antacid medication; the same patients may also be more susceptible to stomach ulcers. ■

selection bias

Latent variables are only one type of mechanism that induces a noncausal correlation between variables. Another important class of confounding factors involves *selection bias*. Selection bias arises when the population that the distribution represents is a segment of the population that exhibits atypical behavior.

Example 21.3

Consider a university that sends out a survey to its alumni, asking them about their history at the institution. Assume that the observed distribution reveals a negative correlation between students who participated in athletic activities (A) and students whose GPA was high (G). Can we conclude from this finding that participating in athletic activities reduces one's GPA? Or that students with a high GPA tend not to participate in athletic activities? An alternative explanation is that the respondents to the survey ($S = s^1$) are not a representative segment population: Students who did well in courses tended to respond, as did students who participated in athletic activities (and therefore perhaps enjoyed their time at school more); students who did neither tended not to respond. In other words, we have a causal link from A to S and from G to S. In this case, even if A and G are independent in the overall distribution over the student population, we may have a correlation in the subpopulation of respondents. This is an instance of standard intercausal reasoning, where $P(a^1 \mid s^1) > P(a^1 \mid g^1, s^1)$. But without accounting for the possible bias in selecting our population, we may falsely explain the correlation using a causal relationship. ■

There are many other examples where correlations might arise due to noncausal reasons. One reason involves a mixture of different populations.

Example 21.4

It is commonly accepted that young girls develop verbal ability at an earlier age than boys. Conversely, boys tend to be taller and heavier than girls. There is certainly no (known) correlation between height and verbal ability in either girls or boys separately. However, if we simply measure height and verbal ability across all children (of the same age), then we may well see a negative correlation between verbal ability and height. ■

This type of situation is a special case of a latent variable, denoting the class to which the instance belongs (gender, in this case). However, it deserves special mention both because it is quite common, and because these class membership variables are often not perceived as "causes" and may therefore be ignored when looking for a confounding common cause.

A similar situation arises when the distribution we obtain arises from two time series, each of which has a particular trend.

Example 21.5

Consider data obtained by measuring, in each year over the past century, the average height of the adult population in the world in that year (H), and the total size of the polar caps in that year (S).

Because average population height has been increasing (due to improved nutrition), and the total size of the polar caps has been decreasing (due to global warming), we would observe a negative correlation between H and S in these data. However, we would not want to conclude that the size of the polar caps causally influences average population height. ■

In a sense, this situation is also an instance of a latent variable, which in this case is time.

Thus, we see that the correlation between a pair of variables X and Y may be a consequence of multiple mechanisms, where some are causal and others are not. To answer a causal query regarding an intervention at X, we need to disentangle these different mechanisms, and to isolate the component of the correlation that is due to the *causal effect* of X on Y. A large part of this chapter is devoted to addressing this challenge.

causal effect

21.2 Causal Models

We begin by providing a formal framework for viewing a Bayesian network as a causal model. A *causal model* has the same form as a probabilistic Bayesian network. It consists of a directed acyclic graph over the random variables in the domain. The model asserts that each variable X is governed by a *causal mechanism* that (stochastically) determines its value based on the values of its parents. That is, the value of X is a (stochastic) function of the values of its parents.

causal model

causal mechanism

A causal mechanism takes the same form as a standard CPD. For a node X and its parents $\boldsymbol{U}$, the causal model has a stochastic function from the values of $\boldsymbol{U}$ to the values of X. In other words, for each value $\boldsymbol{u}$ of $\boldsymbol{U}$, it specifies a distribution over the values of X. The difference is in the interpretation of the edges. In a causal model, we assume that X's parents are its direct causes (relative to the variables represented in the model). In other words, we assume that causality flows in the direction of the edges, so that X's value is actually determined via the stochastic function implied by X's CPD.

The assumption that CPDs correspond to causal mechanisms forms the basis for the treatment of intervention queries. When we intervene at a variable X, setting its value to x, we *replace* its original causal mechanism with one that dictates that it take the value x. This manipulation corresponds to replacing X's CPD with a different one, where $X = x$ with probability 1, regardless of anything else.

Example 21.6

For instance, in example 21.1, if Gump changes his grade to an A by hacking into the registrar's database, the result is a model where his grade is no longer determined by his performance in the class, but rather set to the value A, regardless of any other aspects of the situation. An appropriate graphical model for the postintervention situation is shown in figure 21.1a. In this network, the Grade variable no longer depends on Intelligence or Difficulty, nor on anything else. It is simply set to take the value A with probability 1. ■

mutilated network

The model in figure 21.1a is an instance of the *mutilated network*, a concept introduced in definition 12.1. Recall that, in the mutilated network $\mathcal{B}_{\boldsymbol{Z}=\boldsymbol{z}}$, we eliminate all incoming edges into each variable $Z_i \in \boldsymbol{Z}$, and set its value to be z_i with probability 1.

Based on this intuition, we can now define a causal model as a model that can answer intervention queries using the appropriate mutilated network.

Definition 21.1

causal model

A causal model *$\mathcal{C}$ over $\mathcal{X}$ is a Bayesian network over $\mathcal{X}$, which, in addition to answering proba-*

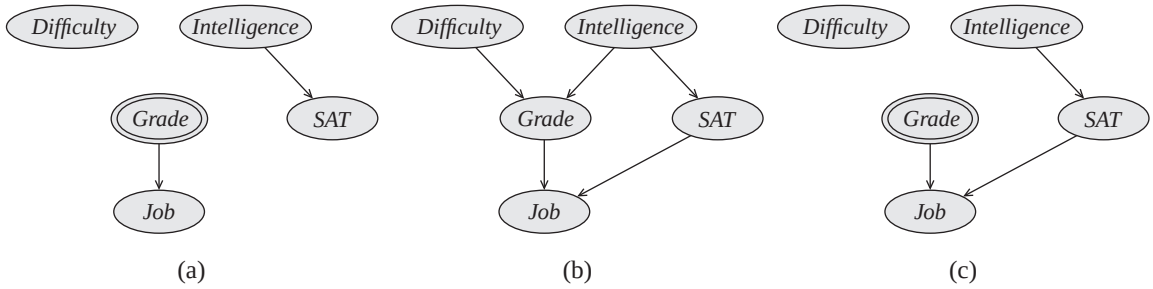


Figure 21.1 Mutilated Student **networks representing interventions** (a) Mutilated Student network with an intervention at G. (b) An expanded Student network, with an additional arc $S \rightarrow J$. (c) A mutilated network from (b), with an intervention at G.

intervention query

bilistic queries, can also answer intervention queries $P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{z}), \boldsymbol{x})$*, as follows:*

$$P_{\mathcal{C}}(\boldsymbol{Y} \mid \text{do}(\boldsymbol{z}), \boldsymbol{x}) = P_{\mathcal{C}_{\boldsymbol{Z}=\boldsymbol{z}}}(\boldsymbol{Y} \mid \boldsymbol{x}).$$

■

Example 21.7

It is easy to see that this approach deals appropriately with example 21.1. Let $\mathcal{C}^{\text{student}}$ *be the appropriate causal model. When we intervene in this model by setting Gump's grade to an A, we obtain the mutilated network shown in figure 21.1a. The distribution induced by this network over Gump's SAT score is the same as the prior distribution over his SAT score in the original network. Thus,*

$$P(S \mid \text{do}(G := g^1)) = P_{\mathcal{C}^{\text{student}}_{G=g^1}}(S) = P_{\mathcal{C}^{\text{student}}}(S),$$

as we would expect. Conversely, the distribution induced by this network on Gump's job prospects is $P_{\mathcal{C}^{\text{student}}}(J \mid G = g^1)$. ■

Note that, in general, the answer to an intervention query does not necessarily reduce to the answer to some observational query.

Example 21.8

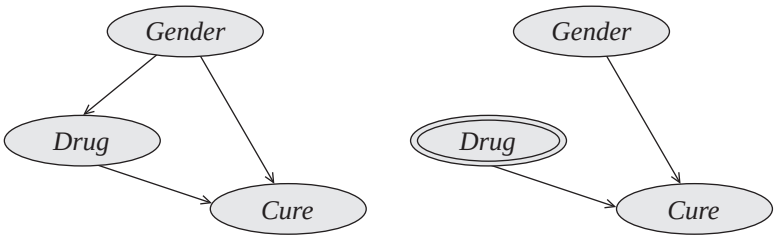
Assume that we start out with a somewhat different Student *network, as shown in figure 21.1b, which contains an edge from the student's SAT score to his job prospects (for example, because the recruiter can also base her hiring decision on the student's SAT scores). Now, the query* $P_{\mathcal{C}^{\text{student}}}(J \mid \text{do}(g^1))$ *is answered by the mutilated network of figure 21.1c. In this case, the answer to the query is clearly not* $P_{\mathcal{C}^{\text{student}}}(J)$*, due to the direct causal influence of his grade on his job prospects. On the other hand, it is also not equal to* $P_{\mathcal{C}^{\text{student}}}(J \mid g^1)$*, because this last expression also includes the influence via the evidential trail* $G \leftarrow I \rightarrow S \rightarrow J$*, which does not apply in the mutilated model.* ■

The ability to provide a formal distinction between observational and causal queries can help resolve some apparent paradoxes that have been the cause of significant debate. One striking example is *Simpson's paradox*, a variant of which is the following:

Simpson's paradox

Example 21.9

Consider the problem of trying to determine whether a drug is beneficial in curing a particular disease within some population of patients. Statistics show that, within the population, 57.5 percent



Causal network for Simpson's paradox

Mutilated network for answering $P(c^1|\text{do}(d^1))$

Figure 21.2 Causal network for Simpson's paradox

of patients who took the drug (D) are cured (C), whereas only 50 percent of the patients who did not take the drug are cured. Given these statistics, we might be inclined to believe that the drug is beneficial. However, more refined statistics show that within the subpopulation of male patients, 70 percent who took the drug are cured, whereas 80 percent of those who did not take the drug are cured. Moreover, within the subpopulation of female patients, 20 percent of who took the drug are cured, whereas 40 percent of those who did not take the drug are cured. Thus, despite the apparently beneficial effect of the drug on the overall population, the drug appears to be detrimental to both men and women! More precisely, we have that:

$$\begin{aligned} P(c^1 \mid d^1) &> P(c^1 \mid d^0) \\ P(c^1 \mid d^1, G = \textit{male}) &< P(c^1 \mid d^0, G = \textit{male}) \\ P(c^1 \mid d^1, G = \textit{female}) &< P(c^1 \mid d^0, G = \textit{female}). \end{aligned}$$

How is this possible?

This case can occur because taking the drug is correlated with gender: men are much more likely to take the drug than women. In this particular example, 75 percent of men take the drug, whereas only 25 percent of women do. With these parameters, even if men and women are equally represented in the population of patients, we obtain the surprising behavior described earlier.

The conceptual difficulty behind this paradox is that it is not clear which statistics one should use when deciding whether to prescribe the drug to a patient: the general ones or the ones conditioned on gender. In particular, it is not difficult to construct examples where this reversal continues, in that conditioning on yet another variable leads one to the conclusion that the drug is beneficial after all, and conditioning on one more reverses the conclusion yet again. So how can we decide which variables we should condition on?

The causal framework provides an answer to this question. The appropriate query we need to answer in determining whether to prescribe the drug is not $P(c^1 \mid d^1)$, but rather $P(c^1 \mid \text{do}(d^1))$. Figure 21.2 shows the causal network corresponding to this situation, and the mutilated network required for answering the query. We will show how we can use the structure of the causal network to answer queries such as $P(c^1 \mid \text{do}(d^1))$. As we will see in example 21.14, the answer to this query shows that the drug is not beneficial, as expected. ■

21.3 Structural Causal Identifiability

The framework of the previous section provides us with a mechanism for answering intervention queries, given a fully specified causal model. However, fully specifying a causal model is often impossible. As we discussed, there are often a multitude of (generally unknown) factors that are latent, and that can induce correlations between the variables. In most cases, we cannot fully specify the causal connection between the latent variables and the variables in our model.

In general, the only thing that we can reasonably hope to obtain is the marginal distribution over the observed variables in the model. As we discussed, for probabilistic queries, this marginal distribution suffices (assuming it can be acquired with reasonable accuracy). However, for intervention queries, we must disentangle the causal influence of X and Y from other factors leading to correlations between them. It is far from clear how we could ever accomplish this goal.

Example 21.10

Consider a pair of variables X, Y with an observed correlation between them, and imagine that our task is to determine $P(Y \mid \text{do}(X))$. Let us even assume that X temporally precedes Y, and therefore we know that Y cannot cause X. However, if we consider the possibility that at least some of the correlation between X and Y is due to a hidden common cause, we have no way of determining how much effect perturbing X would have on Y. If all of the correlation is due to a causal link, then $P(Y \mid \text{do}(X)) = P(Y \mid X)$; conversely, if all of the correlation is due to the hidden common cause, then $P(Y \mid \text{do}(X)) = P(Y)$. And, in general, any value between those two distributions is possible. ■

Thus, given that latent variables are inevitable in many settings, it appears that the situation regarding causal queries is hopeless. Fortunately, as we now show, **we can sometimes answer causal questions in models involving latent variables using observed correlations alone**. More precisely, in this section, we attempt to address to question of which intervention queries are *identifiable*, that is, can be answered using only conditional probabilities involving observable variables. Probabilities over observed variables can be estimated from data or elicited from an expert. Thus, if we can reduce the answer to a query to an expression involving only such probabilities, we may be able to provide a robust and accurate answer to it.

identifiability

21.3.1 Query Simplification Rules

The key observation in this section is that the structure of a causal model give rise to certain equivalence rules over interventional queries, which allow one query to be replaced by an equivalent one that may have a simpler form. By applying one or more of these simplification steps, we may be able to convert one causal query to another query that involves no interventions, and can therefore be answered using observational data alone.

augmented causal model

decision variable

These rules can be defined in terms of an *augmented causal model* that encodes the possible effect of interventions explicitly within the graph structure. More precisely, we view the process of an intervention in terms of a new *decision variable* (see chapter 23) $\widehat{Z}$ that determines whether we intervene at Z, and if so, what its value is. The variable $\widehat{Z}$ takes on values in $\{\epsilon\} \cup Val(Z)$. If $\widehat{Z} = \epsilon$, then Z behaves as a random variable whose distribution is determined by its usual CPD $P(Z \mid \text{Pa}_Z)$; if $\widehat{Z} = z$, then it deterministically sets the value of Z to be z with probability 1. Let $\widehat{\boldsymbol{Z}}$ denote the set $\{\widehat{Z} \ : \ Z \in \boldsymbol{Z}\}$.

Note that, in those cases where Z's value is deterministically set by one parent, all of Z's other parents $\boldsymbol{U}$ become irrelevant, and so we can effectively remove all edges from $\boldsymbol{U}$ to Z. Let $\mathcal{G}^\dagger$ be the augmented model for $\mathcal{G}$. Let $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$ be the graph obtained from $\mathcal{G}^\dagger$ except that every $Z \in \boldsymbol{Z}$ has only the single parent $\widehat{Z}$. Note that $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$ is similar to the mutilated network used in definition 21.1 to define the semantics of intervention queries; the only difference is that we now make the interventions themselves explicit. As we will now see, this difference allows us to study the effect of one intervention within a model that contain others.

intervention query simplification

Based on this construction, we now define three *query simplification* rules. The first simply allows us to insert or delete observations into a query.

Proposition 21.1 *Let $\mathcal{C}$ be a causal model over the graph structure $\mathcal{G}$. Then:*

$$P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{W} = \boldsymbol{w}) = P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x}),$$

if $\boldsymbol{W}$ is d-separated from $\boldsymbol{Y}$ given $\boldsymbol{Z}, \boldsymbol{X}$ in the graph $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$.

This rule is a simple consequence of the fact that probabilities of intervention queries are defined relative to the graph $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$, and the correspondence between independence and d-separation in this graph.

The second rule is subtler, and it allows us to replace an intervention with the corresponding observation.

Proposition 21.2 *Let $\mathcal{C}$ be a causal model over the graph structure $\mathcal{G}$. Then:*

$$P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \text{do}(\boldsymbol{X} := \boldsymbol{x}), \boldsymbol{W} = \boldsymbol{w}) = P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{W} = \boldsymbol{w}),$$

if $\boldsymbol{Y}$ is d-separated from $\widehat{\boldsymbol{X}}$ given $\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{W}$ in the graph $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$.

Intuitively, this rule holds because it tells us that we get no more information regarding $\boldsymbol{Y}$ from the fact that an intervention took place at $\boldsymbol{X}$ than the values $\boldsymbol{x}$ themselves. In other words, knowing that $\boldsymbol{X} = \boldsymbol{x}$, we do not care whether these values were obtained as a result of an intervention or not. This criterion is also equivalent to asking whether $\boldsymbol{X}$ have *requisite CPD* for the query $P(\boldsymbol{Y} \mid do(\boldsymbol{Z} := \boldsymbol{z}), \boldsymbol{X} = \boldsymbol{x}, \boldsymbol{W} = \boldsymbol{w})$, as defined in exercise 3.20. The relationship is not surprising, because an intervention at a variable $X \in \boldsymbol{X}$ corresponds exactly to changing its CPD; if our query is oblivious to changes in the CPD (given X), then we should not care whether there was an intervention at X or not.

requisite CPD

As a simple example, consider the case where $\boldsymbol{Z} = \emptyset$ and $\boldsymbol{W} = \emptyset$, and where we have single variables X, Y. In this case, the rule reduces to the assertion that

$$P(Y \mid do(X := x)) = P(Y \mid X = x),$$

if Y is d-separated from $\widehat{X}$ given X in the graph $\mathcal{G}^\dagger$. The separation property holds if the only trails between X and Y in $\mathcal{G}$ emanate causally from X (that is, go through its children). Indeed, in this case, intervening at X has the same effect on Y as observing X. Conversely, if we have an active trail from $\widehat{X}$ to Y given X, then it must go through a v-structure activated by X. In this case, Y is not a descendant of X, and an observation of X has a very different effect than an intervention at X. The proof of this theorem is left as an exercise (exercise 21.1).

Example 21.11

Consider again the network of figure 21.1b, and assume that the student somehow manages to cheat on the SAT exam and get a higher SAT score. We are interested in the effect on the student's grade, that is, in evaluating a query of the form $P(G \mid \text{do}(S), J, I)$. Consider the augmented model $\mathcal{G}^\dagger$, which contains a new decision parent $\widehat{S}$ for the node S. In this graph, we would have that G is d-separated from $\widehat{S}$ given S, J and I. Thus, the theorem would allow us to conclude that $P(G \mid \text{do}(S), J, I) = P(G \mid S, J, I)$. To provide another intuition for this equality, note that, in the original graph, there are two possible trails between G and S: $G \rightarrow J \leftarrow S$ and $G \leftarrow I \rightarrow S$. The effects of the first trail remain unchanged in the mutilated network: there is no difference between an intervention at S and an observation at S, relative to the trail $G \rightarrow J \leftarrow S$. There is a difference between these two cases relative to the trail $G \leftarrow I \rightarrow S$, but that trail is blocked by the observation at I, and hence there is no effect to changing the intervention at S to an observation. Note that this last argument would not apply to the query $P(G \mid \text{do}(S), J)$, and indeed, the theorem would not allow us to conclude that $P(G \mid \text{do}(S), J) = P(G \mid S, J)$. ■

The final rule allows us to introduce or delete interventions, in the same way that proposition 21.1 allows us to introduce or delete observations.

Proposition 21.3

Let $\mathcal{C}$ be a causal model over the graph structure $\mathcal{G}$. Then:

$$P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \text{do}(\boldsymbol{X} := \boldsymbol{x}), \boldsymbol{W} = \boldsymbol{w}) = P(\boldsymbol{Y} \mid \text{do}(\boldsymbol{Z} := \boldsymbol{z}), \boldsymbol{W} = \boldsymbol{w}),$$

if $\boldsymbol{Y}$ is d-separated from $\widehat{\boldsymbol{X}}$ given $\boldsymbol{Z}, \boldsymbol{W}$ in the graph $\mathcal{G}^\dagger_{\overline{\boldsymbol{Z}}}$.

This analysis can also be interpreted in terms of requisite CPDs. Here, the premise is equivalent to stating that the CPDs of the variables in $\boldsymbol{X}$ are not requisite for the query even when their values are not observed. In this case, we can ignore both the knowledge of the intervention and the knowledge regarding the values imposed by the intervention.

Again, we can obtain intuition by considering the simpler case where $\boldsymbol{Z} = \emptyset$, $\boldsymbol{W} = \emptyset$, and $\boldsymbol{X}$ is a single variable X. Here, the rule reduces to:

$$P(\boldsymbol{Y} \mid do(X := x)) = P(\boldsymbol{Y}),$$

if $\boldsymbol{Y}$ is d-separated from $\widehat{X}$ in the graph $\mathcal{G}^\dagger$. The intuition behind this rule is fairly straightforward. Conditioning on $\widehat{X}$ corresponds to changing the causal mechanism for X. If the d-separation condition holds, this operation provably has no effect on $\boldsymbol{Y}$. From a different perspective, changing the causal mechanism for X can only affect $\boldsymbol{Y}$ causally, via X's children. If there are no trails from $\widehat{X}$ to $\boldsymbol{Y}$ *without* conditioning on X, then there are no causal paths from X to $\boldsymbol{Y}$. Not surprisingly, this condition is equivalent to the graphical criterion for identifying requisite probability nodes (see exercise 21.2), which also test whether the CPD of a variable X can affect the outcome of a given query; in this case, the CPDs of the variable X is determined by whether there is an intervention at X.

Example 21.12

Let us revisit figure 21.1b, and a query of the form $P(S \mid \text{do}(G))$ (taking $\boldsymbol{Y} = S$, $\boldsymbol{X} = G$, and $\boldsymbol{Z}, \boldsymbol{W} = \emptyset$). Consider the augmented model $\mathcal{G}^\dagger$, which contains a new decision parent $\widehat{G}$ for the node S. The node $\widehat{G}$ is d-separated from S in this graph, so that we can conclude $P(S \mid \text{do}(G)) = P(S)$, as we would expect. On the other hand, if our query is $P(S \mid \text{do}(G), J)$

(so that now $\boldsymbol{W} = J$), then G itself is an ancestor of our evidence J. In this case, there is an active trail from $\widehat{G}$ to S in the network; hence, the rule does not apply, and we cannot conclude $P(S \mid \text{do}(G), J) = P(S \mid J)$. Again, this is as we would expect, because when we observe J, the fact that we intervened at G is clearly relevant to S, due to standard intercausal reasoning. ■

21.3.2 Iterated Query Simplification

The rules in the previous section allow us to simplify a query in certain cases. But their applicability appears limited, since there are many queries where none of the rules apply directly. A key insight, however, is that we can also perform other transformations on the query, allowing the rules to be applied.

Example 21.13

To illustrate this approach, consider again the example of example 21.8, which involves the query $P(J \mid \text{do}(G))$ in the network of figure 21.1b. As we discussed, none of our rules apply directly to this query: Obviously we cannot eliminate the intervention — $P(J \mid \text{do}(G)) \neq P(J)$. We also cannot turn the intervention into an observation — $P(J \mid \text{do}(G)) \neq P(J \mid G)$; intuitively, the reason is that intervening at G only affects J via the single edge $G \rightarrow J$, whereas conditioning G also influences J by the indirect trail $G \leftarrow I \rightarrow S \rightarrow J$. This trail is called a back-door trail, *since it leaves G by the "back door."*

However, we can use standard probabilistic reasoning to conclude that:

$$P(J \mid \text{do}(G)) = \sum_S P(J \mid \text{do}(G), S) P(S \mid \text{do}(G)).$$

Both of the terms in the summation can *be further simplified. For the first term, we have that the only active trail from G to J is now the direct edge $G \rightarrow J$. More formally, J is d-separated from G given S in the graph where outgoing arcs from G have been deleted. Thus, we can apply proposition 21.2, and conclude:*

$$P(J \mid \text{do}(G), S) = P(J \mid G, S).$$

For the second term, we have already shown in example 21.12 that $P(S \mid \text{do}(G)) = P(S)$. Putting the two together, we obtain that:

$$P(J \mid \text{do}(G)) = \sum_S P(J \mid G, S) P(S).$$

■

This example illustrates a process whereby we introduce conditioning on some set of variables:

$$P(\boldsymbol{Y} \mid do(\boldsymbol{X}), \boldsymbol{Z}) = \sum_{\boldsymbol{W}} P(\boldsymbol{Y} \mid do(\boldsymbol{X}), \boldsymbol{Z}, \boldsymbol{W}) P(\boldsymbol{W} \mid do(\boldsymbol{X}), \boldsymbol{Z}).$$

Even when none of the transformation rule apply to the query $P(\boldsymbol{Y} \mid do(\boldsymbol{X}), \boldsymbol{Z})$, they may apply to each of the two terms in the summation of the transformed expression.

back-door trail

The example illustrates one special case of this transformation. A *back-door trail* from X to Y is an active trail that leaves X via a parent of X. For a query $P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$, a set $\boldsymbol{W}$ satisfies the *back-door criterion* if no node in $\boldsymbol{W}$ is a descendant of $\boldsymbol{X}$, and $\boldsymbol{W}$ blocks all back-door

back-door criterion

paths from $\boldsymbol{X}$ to $\boldsymbol{Y}$. Using an argument identical to the one in the example, we can show that if a set $\boldsymbol{W}$ satisfies the back-door criterion for a query $P(\boldsymbol{Y} \mid do(\boldsymbol{X}))$, then

$$P(\boldsymbol{Y} \mid do(\boldsymbol{X})) = \sum_{\boldsymbol{W}} P(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{W}) P(\boldsymbol{W}). \tag{21.1}$$

The back-door criterion can be used to address Simpson's paradox, as described in example 21.9.

Example 21.14

Consider again the query $P(c^1 \mid \mathrm{do}(d^1))$. The variable G (Gender) introduces a back-door trail between C and D. We can account for its influence using equation (21.1):

$$P(c^1 \mid \mathrm{do}(d^1)) = \sum_{g} P(c^1 \mid d^1, g) P(g).$$

We therefore obtain that:

$$\begin{aligned} P(c^1 \mid \mathrm{do}(d^1)) &= 0.7 \cdot 0.5 + 0.2 \cdot 0.5 = 0.45 \\ P(c^1 \mid \mathrm{do}(d^0)) &= 0.8 \cdot 0.5 + 0.4 \cdot 0.5 = 0.6. \end{aligned}$$

And therefore, we should not prescribe the drug. ■

More generally, by repeated application of these rules, we can sometimes simplify fairly complex queries, obtaining answers in cases that are far from obvious at first glance.

Box 21.A — Case Study: Identifying the Effect of Smoking on Cancer. *In the early 1960s, following a significant increase in the number of smokers that occurred around World War II, people began to notice a substantial increase in the number of cases of lung cancer. After a great many studies, a correlation was noticed between smoking and lung cancer. This correlation was noticed in both directions: the frequency of smokers among lung cancer patients was substantially higher than in the general population, and the frequency of lung cancer patients within the population of smokers was substantially higher than within the population of nonsmokers.*

These results, together with some experiments of injecting tobacco products into rats, led the Surgeon General, in 1964, to issue a report linking cigarette smoking to cancer and, most particularly, lung cancer. His claim was that the correlation found is causal, namely: If we ban smoking, the rate of cancer cases will be roughly the same as the one we find among nonsmokers in the population.

These studies came under severe attacks from the tobacco industry, backed by some very prominent statisticians. The claim was that the observed correlations can also be explained by a model in which there is no causal connection between smoking and lung cancer. Instead, an unobserved genotype might exist that simultaneously causes cancer and produces an inborn craving for nicotine. In other words, there were two hypothesized models, shown in figure 21.A.1a,b.

The two models can express precisely the same set of distributions over the observable variables S, C. Thus, they can do an equally good job of representing the empirical distribution over these variables, and there is no way to distinguish between them based on observational data alone. Moreover, both models will provide the same answer to standard probabilistic queries such as $P(c^1 \mid$

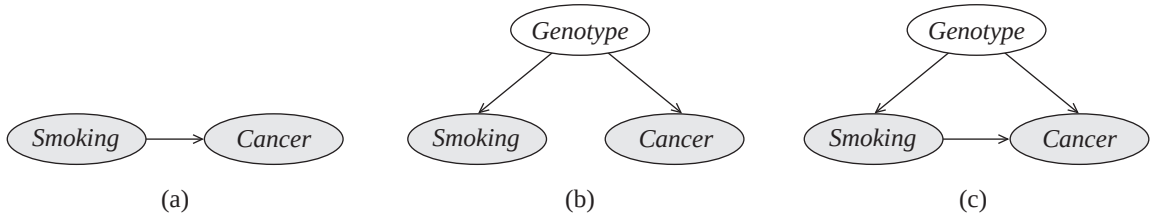


Figure 21.A.1 — Three candidate models for smoking and cancer. (a) a direct causal influence; (b) indirect influence via a latent common parent *Genotype*; (c) incorporating both types of influence.

s^1). *However, relative to interventional queries, these models have very different consequences. According to the Surgeon General's model, we would have:*

$$P(c^1 \mid \text{do}(S := s^1)) = P(c^1 \mid s).$$

In other words, if we force people to smoke, their probability of getting cancer is the same as the probability conditioned on smoking, which is much higher than the prior probability. On the other hand, according to the tobacco industry model, we have that

$$P(c^1 \mid \text{do}(S := s^1)) = P(c^1).$$

In other words, making the population smoke or stop smoking would have no effect on the rate of cancer cases.

Pearl (1995) proposes a formal analysis of this dilemma, which we now present. He proposes that we combine these two models into a single joint model, which accommodates for both possible types of interactions between smoking and cancer, as shown in figure 21.A.1c. We now need to assess, from the marginal distribution over the observed variables alone, the parameterization of the various links. Unfortunately, it is impossible to determine the parameters of these links from observational data alone, since both of the original two models (in figure 21.A.1a,b) can explain the data perfectly.

However, if we refine the model somewhat, introducing an additional assumption, we can provide such estimates. Assume that we determine that the effect of smoking on cancer is not a direct one, but occurs through the accumulation of tar deposits in the lungs, as shown in figure 21.A.2a. Note that this model makes the assumption that the accumulation of tar in the lungs is not *directly affected by the latent Genotype variable. As we now show, if we can measure the amount of tar deposits in the lungs of various individuals (for example, by X-ray or in autopsies), we can determine the probability of the intervention query* $P(c^1 \mid \text{do}(s^1))$ using observed correlations alone.

We are interested in $P(c^1 \mid \text{do}(s^1))$*, which is an intervention query whose mutilated network is* $\mathcal{G}^\dagger_{\overline{S}}$ *in figure 21.A.2b. Standard probabilistic reasoning shows that:*

$$P(C \mid \text{do}(s^1)) \quad = \quad \sum_t P(C \mid \text{do}(s^1), t) P(t \mid \text{do}(s^1)).$$

We now consider and simplify each term in the summation separately.

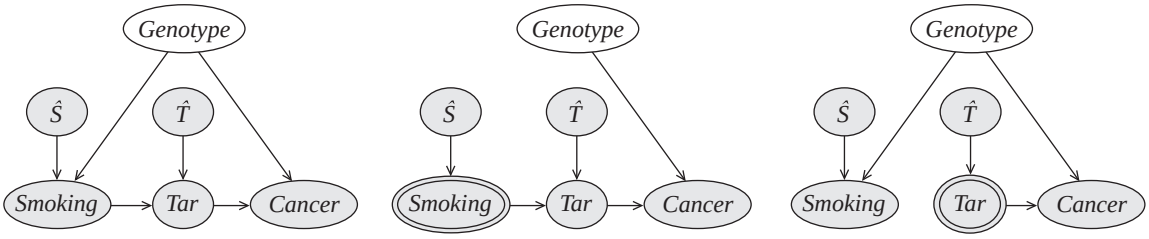


Figure 21.A.2 — Determining causality between smoking and cancer. Augmented network for a model in which the effect of smoking on cancer is indirect, mediated via tar accumulation in the lungs, and mutilated variants for two possible interventions.

The second term, which measures the effect of Smoking on Tar, can be simplified directly using our rule for converting interventions to observations, stated in proposition 21.2. Here, $\widehat{S}$ is d-separated from T given S in the graph $\mathcal{G}^\dagger$, shown in figure 21.A.2a. It follows that:

$$P(t \mid \text{do}(s)) \quad = \quad P(t \mid s).$$

Intuitively, the only active trail from $\widehat{S}$ to T goes via S, and the effect of that trail is identical regardless of whether we condition on S or intervene at S.

Now, let us examine the first term, $P(C \mid \text{do}(s^1), t)$, which measures the effect of Tar on Cancer in the presence of our intervention on S. Unfortunately, we cannot directly convert the intervention at S to an observation, since C is not d-separated from $\widehat{S}$ given S, T in $\mathcal{G}^\dagger$. However, we can *convert the observation at T to an intervention, because C is d-separated from $\widehat{T}$ given S, T in the graph $\mathcal{G}^\dagger_{\overline{S}}$.*

$$P(C \mid \text{do}(s^1), t) = P(C \mid \text{do}(s^1), \text{do}(t)).$$

We can now eliminate the intervention at S from this expression using proposition 21.3, which applies because C is d-separated from $\widehat{S}$ given T in the graph $\mathcal{G}^\dagger_{\overline{T}}$ (figure 21.A.2c), obtaining:

$$P(C \mid \text{do}(s^1), \text{do}(t)) = P(C \mid \text{do}(t)).$$

Considering this last expression, we can apply standard probabilistic reasoning and introduce conditioning on S:

$$\begin{aligned} P(C \mid \text{do}(t)) &= \sum_{s'} P(C \mid \text{do}(t), s') P(s' \mid \text{do}(t)) \\ &= \sum_{s'} P(C \mid t, s') P(s' \mid \text{do}(t)) \\ &= \sum_{s'} P(C \mid t, s') P(s'). \end{aligned}$$

The second equality is an application of proposition 21.2, which applies because C is d-separated from $\widehat{T}$ given T, S in $\mathcal{G}^\dagger$. The final equality is a consequence of proposition 21.3, which holds because S is d-separated from $\widehat{T}$ in $\mathcal{G}^\dagger$.

Putting everything together, we get:

$$\begin{aligned} P(c \mid \text{do}(s^1)) &= \sum_t P(c \mid \text{do}(s^1), t) P(t \mid \text{do}(s^1)) \\ &= \sum_t P(c \mid \text{do}(s^1), t) P(t \mid s^1) \\ &= \sum_t P(t \mid s^1) \sum_{s'} P(c \mid t, s') P(s'). \end{aligned}$$

Thus, if we agree that tar in the lungs is the intermediary between smoking and lung cancer, we can uniquely determine the extent to which smoking causes lung cancer even in the presence of a confounding latent variable!

Of course, this statement is more useful as a thought experiment than as a practical computational tool, both because it is unlikely that smoking affects cancer only via tar accumulation in the lungs and because it would be very difficult in practice to measure this intermediate variable. Nevertheless, this type of analysis provides insight on the extent to which understanding of the underlying causal model allows us to identify the value of intervention queries even in the presence of latent variables.

These rules provide a powerful mechanism for answering intervention queries, even when the causal model involves latent variables (see, for example, box 21.A). More generally, using these rules, we can show that the query $P(Y \mid do(x))$ is identifiable in each of the models shown in figure 21.3 (see exercise 21.4). In these figures, we have used a bidirected dashed arrow, known as a *bow pattern*, to denote the existence of a latent common cause between two variables. For example, the model in (d) has the same structure as in our Smoking model of figure 21.A.2a. (box 21.A). This notation simplifies the diagrams considerably, and it is therefore quite commonly used. Note that none of the diagrams contain a bow pattern between X and one of its children. In general, a necessary condition for identifiability is that the graph contain no bow pattern between X and a child of X that is an ancestor of Y. The reason is that, if such a bow pattern exists between X and one of its children W, we have no mechanism for distinguishing X's direct influence on W and the indirect correlation induced by the latent variable, which is their common parent.

bow pattern

It is interesting to note that, in the models shown in (a), (b), and (e), Y has a parent Z whose effect on Y is not identifiable, yet the effect of X on Y, including its effect via Z, is identifiable. For example, in (a), $P(Y \mid X) = P(Y \mid do(X))$, and so there is no need to disentangle the influence that flows through the $Z \rightarrow Y$ edge, and the influence that flows through the bow pattern. These examples demonstrate that, to identify the influence of one variable on another, it is not necessary to identify every edge involved in the interactions between them.

Figure 21.4 shows a few examples of models where the influence of X on Y is not identifiable. Interestingly, the model in (g) illustrates the converse to the observation we just stated: identification of every edge involved in the interaction between X and Y does not suffice to

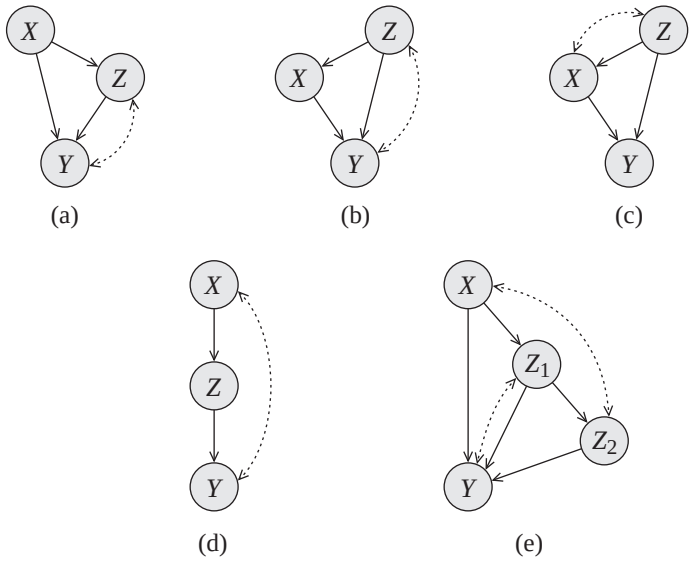


Figure 21.3 Examples of models where $P(Y \mid \textbf{\textit{do}}(X))$ is identifiable. The bidirected dashed arrows denote cases where a latent variable affects both of the linked variables.

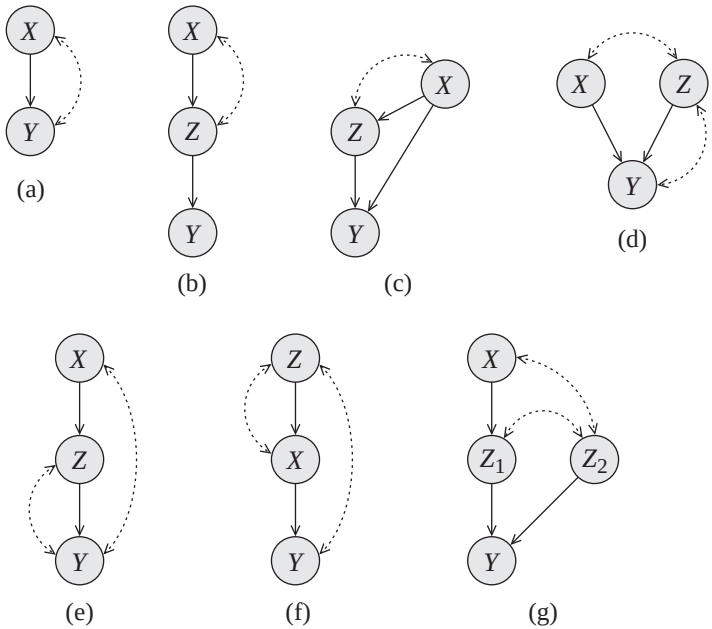


Figure 21.4 Examples of models where $P(Y \mid \textbf{\textit{do}}(X))$ is not identifiable. The bidirected dashed arrows denote cases where a latent variable affects both of the linked variables.

identify their interaction. In particular, in this model, we can identify all of $P(Z_1 \mid do(X))$, $P(Z_2 \mid do(X))$, $P(Y \mid do(Z_1))$, and $P(Y \mid do(Z_2))$, but we cannot identify $P(Y \mid do(X))$ (see exercise 21.5).

Note that the removal of any edge (directed or bidirected) from a causal diagram of this type can only help in the identifiability of causal effects, since it can only deactivate active trails. Conversely, adding edges can only reduce identifiability. Hence, any subgraph of the graphs in figure 21.3 is also identifiable, and any extension of the graphs in figure 21.4 is nonidentifiable. Moreover, it is possible to show that the graphs in figure 21.3 are maximal, in the sense that adding any edge renders $P(Y \mid do(X))$ unidentifiable; similarly, the graphs in figure 21.4 are minimal, in that the query is identifiable in any of their (strict) subgraphs. Similarly, the introduction of mediating observed variables onto any edge in a causal graph — transforming an arc $A \rightarrow B$ into a path $A \rightarrow Z \rightarrow B$ for a new observed variable Z — can only increase our ability to identify causal effects.

Finally, we note that the techniques in this section provide us with methods for answering only queries that are identifiable. Unfortunately, unidentifiable queries are quite common in practice. In section 21.5, we describe methods that allow us to provide partial answers for queries that are not identifiable.

21.4 Mechanisms and Response Variables ⋆

The underlying intuition for our definition of a causal model is that the graph structure defines a causal progression, where the value of each variable is selected, in order, based on the values of its parents, via some causal mechanism. So far, the details of this causal mechanism have remained implicit. We simply assumed that it induces, for each variable X, a conditional probability distribution $P(X \mid \mathrm{Pa}_X)$.

This approach suffices in cases, as in the previous section, where we can compute the value of a causal query in terms of standard probabilistic queries. In general, however, this identification may not be possible. In such cases, we have to reason explicitly about the mechanism governing the behavior of the variables in the model and the ways in which the latent variables may influence the observed variables. By obtaining a finer-grained analysis of these mechanisms, we may be able to provide some analysis for intervention queries that are not identifiable from probabilities over the observable variables alone.

A second reason for understanding the mechanism in more detail is to answer additional types of queries: counterfactual queries, and certain types of diagnostic queries. As we discussed, our intuition for a counterfactual query is that we want to keep as much as we can between the real and the counterfactual world. By understanding the exact mechanism that governs each variable, we can obtain more reliable inferences about what took place in the true world, so as to preserve as much as possible when reasoning about the counterfactual events.

Example 21.15 *Consider a significantly simplified causal model for Gump's job prospects, where we have only the edge $G \rightarrow J$ — Gump's job prospects depend only on his grade. Let us also assume that grades are binary-valued — high and low. Our counterfactual query is as follows: "Gump received a low grade. He applied to Acme Consulting, and he was not hired. Would Acme have hired him had he managed to hack into the registrar's computer system and change his grade to a high one?"*

Consider two very different models of the world. In the first, there are two (equally likely)

populations of companies. Those in the first population are not in hiring mode, and they always reject Gump without looking at his grade. Companies in the second population are desperate for employees and will take anyone, regardless of their grade. The basic intuition of counterfactual queries is that, when we move to the counterfactual world, Gump does not get another "random draw" of company. We want to answer the counterfactual query assuming that Acme's recruiting response model is the same in both cases. Under this assumption, we can conclude that Acme was not in hiring mode, and that Gump's outcome would have been no different had he managed to change his grade.

In a second model, there are also two (equally likely) populations of companies. In the first, the company is highly selective, and it hires a student if and only if his grades are high. In the second population, the recruiter likes to hire underdogs because he can pay them less, and he hires Gump if and only if his grades are low. In this setting, if we again preserve the hiring response of the company, we would conclude that Acme must use a selective recruiting policy, and so Gump would have been hired had he managed to change his grade to a high one.

Note that these two models are identical from a probabilistic perspective. In both cases, $P(J \mid G) = (0.5, 0.5)$. *But they are very different with respect to answering counterfactual queries.* ■

troubleshooting

The same type of situation applies in a setting where we act in the world, and where we wish to reason about the probabilities of different events before and after our action. For example, consider the problem of *troubleshooting* a broken device with multiple components, where a fault in each can lead to the observed failure mode. Assume that we replace one of the components, and we wish to reason about the probabilities in the system after the replacement action. Intuitively, if the problem was not with the component we replaced, then the replacement action should have no effect: the probability that the device remains broken should be 1. The mechanism by which different combinations of faults can lead to the observed behavior is quite complex, and we are generally uncertain about which one is currently the case. However, we wish to have this mechanism persist between the prerepair and postrepair state. Box 21.C describes the use of this approach in a real-world diagnostic setting.

As these examples illustrate, to answer certain types of queries, we need to obtain a detailed specification of the causal mechanism that determines the values of different variables in the model. In general, we can argue that (quantum effects aside) any randomness in the model is the cumulative effect of latent variables that are simply unmodeled. For example, consider even a seemingly random event such as the outcome of a coin toss. We can imagine this event depending on a large number of exogenous variables, such as the rotation of the coin when it was tossed, the forces applied to it, any air movement in the room, and more. Given all of these factors, the outcome of the coin toss is arguably deterministic. As another example, a company's decision on whether to hire Gump can depend on the company's current goals and funding level, on the existence of other prospective applicants, and even on whether the recruiter likes the color of Gump's tie. Based on this intuition, we can divide the variables into two groups. The

endogenous variable

endogenous variables are those that we choose to include in the model; the exogenous variables are unmodeled, latent variables. We can now argue, as a hypothetical thought experiment, that the exogenous variables encompass all of the stochasticity in the world; therefore, given all of the exogenous variables, each endogenous variable (say the company's hiring decision) is fully determined by its endogenous parents (Gump's grade in our example).

Clearly, we cannot possibly encode a complete specification of a causal model with all of the relevant exogenous variables. In most cases, we are not even aware of the entire set of relevant exogenous variables, far less able to specify their influence on the model. However, for our purposes, we can abstract away from specific exogenous variables and simply focus on their effect on the endogenous variables. To understand this transformation, consider the following example.

Example 21.16

Consider again the simple setting of example 21.15. Let $\boldsymbol{U}$ be the entire set of exogenous variables affecting the outcome of the variable J in the causal model $G \rightarrow J$. We can now assume that the value of J is a deterministic function of $G, \boldsymbol{U}$ — for each assignment $\boldsymbol{u}$ to $\boldsymbol{U}$ and g to G, the variable J deterministically takes the value $f_J(g, \boldsymbol{u})$. We are interested in modeling only the interaction between G and J. Thus, we can partition the set of assignments $\boldsymbol{u}$ into four classes, based on the mapping μ that they induce between G and J:

- *$\mu^J_{1 \mapsto 1, 0 \mapsto 1}$ — those where $f_J(\boldsymbol{u}, g^1) = j^1$ and $f_J(\boldsymbol{u}, g^0) = j^1$; these are the "always hire" cases, where Gump is hired regardless of his grade.*
- *$\mu^J_{1 \mapsto 1, 0 \mapsto 0}$ — those where $f_J(\boldsymbol{u}, g^1) = j^1$ and $f_J(\boldsymbol{u}, g^0) = j^0$; these are the "rational recruiting" cases, where Gump is hired if and only if his grade is high.*
- *$\mu^J_{1 \mapsto 0, 0 \mapsto 1}$ — those where $f_J(\boldsymbol{u}, g^1) = j^0$ and $f_J(\boldsymbol{u}, g^0) = j^1$; these are the "underdog" cases, where Gump is hired if and only if his grade is low.*
- *$\mu^J_{1 \mapsto 0, 0 \mapsto 0}$ — those where $f_J(\boldsymbol{u}, g^1) = j^0$ and $f_J(\boldsymbol{u}, g^0) = j^0$; these are the "never hire" cases, where Gump is not hired regardless of his grade.*

Each assignment $\boldsymbol{u}$ induces precisely one of these four different mappings between G and J. For example, we might have a situation where, if Gump wears a green tie with pink elephants, he is never hired regardless of his grade, which is the last case in the previous list.

For the purposes of reasoning about the interaction between G and J, we can abstract away from modeling specific exogenous variables $\boldsymbol{U}$, and simply reason about how likely we are to encounter each of the four categories of functions induced by assignments $\boldsymbol{u}$. ■

Generalizing from this example, we define as follows:

Definition 21.2

response variable

Let X be a variable and $\boldsymbol{Y}$ be a set of parents. A response variable *for X given $\boldsymbol{Y}$ is a variable U^X whose domain is the set of all possible functions $\mu(\boldsymbol{Y})$ from $Val(\boldsymbol{Y})$ to $Val(X)$. That is, let $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m$ be an enumeration of $Val(\boldsymbol{Y})$ (for $m = |Val(\boldsymbol{Y})|$). For a tuple of (generally not distinct) values $x_1, \ldots, x_m \in Val(X)$, we use $\mu^X_{(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m) \mapsto (x_1, \ldots, x_m)}$ to denote the function that assigns x_i to $\boldsymbol{y}_i$, for $i = 1, \ldots, m$. The domain of U^X contains one such function for each such tuple, giving a total of k^m functions of this form (for $|Val(X)| = k$).* ■

Example 21.17

In example 21.16, we introduce two response variables, U^G and U^J. Because G has no endogenous parents, the variable U^G is degenerate: its values are simply g^1 and g^0. The response variable U^J takes one of the four values $\mu^J_{1 \mapsto 1, 1 \mapsto 1}$, $\mu^J_{1 \mapsto 1, 1 \mapsto 0}$, $\mu^J_{1 \mapsto 0, 1 \mapsto 1}$, $\mu^J_{1 \mapsto 0, 1 \mapsto 0}$, each of which defines a function from G to J. Thus, given U^J and G, the value of J is fully determined. For example, if $U^J = \mu^J_{1 \mapsto 1, 1 \mapsto 0}$, and $G = g^1$, then $J = j^1$. ■

Definition 21.3 functional causal model

A functional causal model *$\mathcal{C}$ over a set of endogenous variables $\mathcal{X}$ is a causal model defined over two sets of variables: the variables $\mathcal{X}$, and a set of response variables $\mathcal{U} = \{U^X : X \in \mathcal{X}\}$. Each variable $X \in \mathcal{X}$ has a set of parents $\mathrm{Pa}_X \subset \mathcal{X}$, and a response variable parent U^X for X given Pa_X. The model for each variable $X \in \mathcal{X}$ is deterministic: When $U^X = \mu$ and $\mathrm{Pa}_X = \boldsymbol{y}$, then $X = \mu(\boldsymbol{y})$ with probability 1.*

The model $\mathcal{C}$ also defines a joint probability distribution over the response variables, defined as a Bayesian network over $\mathcal{U}$. Thus, each response variable U has a set of parents $\mathrm{Pa}_U \subset \mathcal{U}$, and a CPD $P(U \mid \mathrm{Pa}_U)$. ■

Functional causal models provide a much finer-grained specification of the underlying causal mechanisms than do standard causal models, where we only specify a CPD for each endogenous variable. In our $G \to J$ example, rather than specifying a CPD for J, we must specify a distribution over U^J. A table representation of the CPD has two independent parameters — one for each assignment to G. The distribution $P(U^J)$ has three independent parameters — there are four possible values for U^J, and their probabilities must sum to 1. While, in this case, the blowup in the representation may not appear too onerous, the general case is far worse. In general, consider a variable X with parents $\boldsymbol{Y}$, and let $k = |Val(X)|$ and $m = |Val(\boldsymbol{Y})|$. The total number of possible mappings from $Val(\boldsymbol{Y})$ to $Val(\boldsymbol{X})$ is k^m — each function selects one of X's k values for each of the m assignments to $\boldsymbol{Y}$. Thus, the total number of independent parameters in (an explicit representation of) $P(U^X)$ is $k^m - 1$. By comparison, a table-CPD $P(X \mid \boldsymbol{Y})$ requires $m(k-1)$ independent parameters only.

Example 21.18

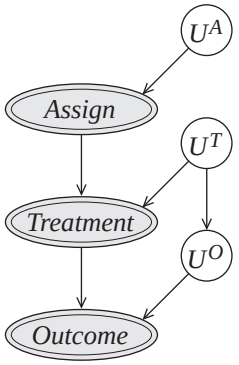
Consider the network in figure 21.5a, representing a causal model for a randomized clinical trial, where some patients are randomly assigned a medication and others a placebo. The model contains three binary-valued endogenous variables: A indicates the treatment assigned to the patient; T indicates the treatment actually received; and O indicates the observed outcome (positive or negative). The model contains three response variables: U^A, U^T, and U^O. Intuitively, U^A (which has a uniform distribution) represents the stochastic event determining the assignment to the two groups (medication and placebo); U^T determines the patient's model for complying with the prescribed course of treatment; and U^O encodes the form of the patient's response to different treatments.

The domain of U^T consists of the following four functions:

- $\mu^T_{1\mapsto 1,0\mapsto 1}$ — *always taker;*
- $\mu^T_{1\mapsto 1,0\mapsto 0}$ — *complier (takes medicine if and only if prescribed);*
- $\mu^T_{1\mapsto 0,0\mapsto 1}$ — *defier (takes medicine if and only if not prescribed);*
- $\mu^T_{1\mapsto 0,0\mapsto 0}$ — *never taker.*

Similarly, the domain of U^O consists of the following four functions:

- $\mu^O_{1\mapsto 1,0\mapsto 1}$ — *always well;*
- $\mu^O_{1\mapsto 1,0\mapsto 0}$ — *helped (recovers if and only if takes medicine);*
- $\mu^O_{1\mapsto 0,0\mapsto 1}$ — *hurt (recovers if and only if does not take medicine);*
- $\mu^O_{1\mapsto 0,0\mapsto 0}$ — *never well.*



always taker	complier	defier	never taker
0.05	0.7	0.05	0.2

$P(U^T)$

U^T	always well	helped	hurt	never well
always taker	0.2	0.5	0.1	0.2
complier	0.5	0.4	0.05	0.05
defier	0.3	0.4	0.1	0.2
never taker	0.6	0.3	0.05	0.05

$P(U^O \mid U^T)$

U^T	U^O	probability
always taker	always well	0.01
always taker	helped	0.025
always taker	hurt	0.005
always taker	never well	0.01
complier	always well	0.35
complier	helped	0.28
complier	hurt	0.035
complier	never well	0.035
defier	always well	0.015
defier	helped	0.02
defier	hurt	0.005
defier	never well	0.02
never taker	always well	0.12
never taker	helped	0.06
never taker	hurt	0.01
never taker	never well	0.01

$P(U^T, U^O)$

Figure 21.5 A simple functional causal model for a clinical trial. (a) Network structure. (b) Sample CPDs and resulting joint distribution for the response variables U^T and U^O.

The model makes explicit two important assumptions. First, the assigned treatment A does not influence the response O directly, but only through the actual treatment T; this conditional independence is achieved by the use of a placebo in the control group, so that the patients do not know to which of the two groups they were assigned. The second assumption is that A is marginally independent from $\{U^T, U^O\}$, which is ensured via the randomization of the assignment to the two groups. Note, however, that U^T and U^O are not *independent, reflecting the fact that factors determining a patient's decision to comply with the treatment can also affect his final outcome. (For example, patients who are not very ill may neglect to take their medication, and may be more likely to get well regardless.) Thus, to specify this model fully, we need to define a joint probability distribution over U^T, U^O — a total of fifteen independent parameters.* ■

We see that a full specification of even a simple functional causal model is quite complex.

21.5 Partial Identifiability in Functional Causal Models ⋆

In section 21.3, we considered the task of answering causal queries involving interventions. As we discussed, unlike probability queries, to answer intervention queries we generally need to consider the effect of latent variables on the observable variables. A functional causal model $\mathcal{C}$ provides a complete specification of a causal model, including the effect of the latent variables, and can thus be used to answer any intervention query. As in definition 21.1, we mutilate the network by eliminating all incoming arcs to intervened variables, and then do inference on the resulting network.

However, this "solution" does not address the fundamental problem. As we discussed in section 21.3, our accumulated statistics are generally only over the observable variables. We will rarely have any direct observations of the finer-grained distributions over the response variables. Thus, it is unrealistic to assume that we can construct a fully specified functional causal model. So, what is the point in defining these constructs if we can never obtain them? As we show in this section and the next one, these networks, even if never fully elicited, can help us provide at least partial answers to both intervention and counterfactual queries.

The basis for this approach rests on the fact that the distributions of response variables are related to the conditional probabilities for the endogenous variables. Thus, we can use our information about the latter to *constrain* the former. To understand this relationship, consider the following example:

response variable constraints

Example 21.19

Let us revisit example 21.16, and consider the observed probability $P(j^1 \mid g^1)$ — the probability that Gump gets a job if he gets a high grade. Given $G = g^1$, we get j^1 in two cases: in the "always hire" case — $U^J = \mu^J_{1\mapsto 1,0\mapsto 1}$, and in the "rational recruiting" case — $U^J = \mu^J_{1\mapsto 1,0\mapsto 0}$. These two choices for U^J are indistinguishable in the context g^1, but would have led to different outcomes had Gump had a low grade. As a consequence, we have that:

$$\begin{aligned}
P(j^1 \mid g^1) &= P(\mu^J_{1\mapsto 1,0\mapsto 1}) + P(\mu^J_{1\mapsto 1,0\mapsto 0}) \\
P(j^1 \mid g^0) &= P(\mu^J_{1\mapsto 1,0\mapsto 1}) + P(\mu^J_{1\mapsto 0,0\mapsto 1}) \\
P(j^0 \mid g^1) &= P(\mu^J_{1\mapsto 0,0\mapsto 1}) + P(\mu^J_{1\mapsto 0,0\mapsto 0}) \\
P(j^0 \mid g^0) &= P(\mu^J_{1\mapsto 1,0\mapsto 0}) + P(\mu^J_{1\mapsto 0,0\mapsto 0}).
\end{aligned}$$

We see that each conditional probability is a sum of some subset of the probabilities associated with the response variable. ■

In the case where the response variable U^X is marginally independent of other response variables, we can generalize this example to provide a full formulation of the constraints that relate the observed conditional probabilities $P(X \mid \boldsymbol{Y})$ and the distributions over U^X:

$$P(x_i \mid \boldsymbol{y}_i) = \sum_{\bar{x}_{-i} \in Val(X)^{m-1}} P(\mu^X_{\boldsymbol{y}_i \to x_i, \bar{\boldsymbol{y}}_{-i} \mapsto \bar{x}_{-i}}), \tag{21.2}$$

where $\bar{\boldsymbol{y}}_{-i}$ is the tuple of all assignments $\boldsymbol{y}_j$ except for $j = i$, and similarly for $\bar{x}_{-i}$. The more general case, where response variables may be correlated, is a little more complex, due to the richer parameterization of the model.

Example 21.20

Consider again example 21.18. In this case, our functional causal model has sixteen unknown response parameters, specifying the joint distribution $P(U^T, U^O)$. By observing the statistics over the endogenous variables, we can evaluate the distribution $P(T, O \mid A)$, which can be used to constrain $P(U^T, U^O)$. Let $\nu_{(1,0)\mapsto(i,j),(1,0)\mapsto(k,l)}$ (for $i, j, k, l \in \{0, 1\}$ denote $P(\mu^T_{1\mapsto i,0\mapsto j}, \mu^O_{1\mapsto k,0\mapsto l})$. Using reasoning similar to that of example 21.19, we can verify that these two sets of probabilities are related by the following linear equalities:

$$P(t^i, o^j \mid a^k) = \sum_{i',j' \in \{0,1\}} \nu_{(k,1-k)\mapsto(i,i'),(i,1-i)\mapsto(j,j')} \qquad \forall i, j, k. \tag{21.3}$$

For example,

$$P(t^1, o^0 \mid a^1) = \sum_{i',j' \in \{0,1\}} \nu_{(1,0)\mapsto(1,i'),(1,0)\mapsto(0,j')};$$

that is, to be consistent with the assignment a^1, t^1, o^0, the U^T function should map a^1 to t^1, but it can map a^0 arbitrarily, and the U^O function should map t^1 to o^0, but it can map t^0 arbitrarily. ■

Thus, we see that the observed probabilities over endogenous variables impose constraints on the possible distributions of the response variables. These constraints, in turn, impose constraints over the possible values of various queries of interest. By reasoning about the possible values for the distributions over the response variables, we can often obtain reasonably precise bounds over the values of queries of interest.

Example 21.21

Continuing our clinical trial example, assume that we are interested in determining the extent to which taking the medication improves a patient's chances of a cure. Naively, we might think to answer this question by evaluating $P(o^1 \mid t^1) - P(o^1 \mid t^0)$. This approach would be incorrect, since it does not account for the correlation between compliance and cure. For example, if patients who choose to comply with the treatment are generally sicker and therefore less likely to get well regardless, then the cure rate in the population of patients for which $T = t^1$ will be misleadingly low, giving a pessimistic estimate of the efficacy of treatment.

A correct answer is obtained by the corresponding intervention query,

$$P(o^1 \mid \text{do}(t^1)) - P(o^1 \mid \text{do}(t^0)),$$

average causal effect

which measures the increase in cure probability between a patient who was forced not to take the treatment and one who was forced to take it. This query is also known as the average causal effect *of T on O and is denoted $\text{ACE}(T \to O)$. Unfortunately, the causal model for this situation contains a bidirected arc between T and O, due to the correlation between their responses. Therefore, the influence of T on O is not identifiable in this model, and, indeed, none of the simplification rules of section 21.3 apply in this case. However, it turns out that we can still obtain surprisingly meaningful bounds over the value of this query.*

We begin by noting that

$$\begin{aligned} P(o^1 \mid \text{do}(t^1)) &= P(\mu^O_{1\mapsto 1,0\mapsto 1}) + P(\mu^O_{1\mapsto 1,0\mapsto 0}) \\ P(o^1 \mid \text{do}(t^0)) &= P(\mu^O_{1\mapsto 1,0\mapsto 1}) + P(\mu^O_{1\mapsto 0,0\mapsto 1}), \end{aligned}$$

so that

$$\mathrm{ACE}(T \to O) = P(o^1 \mid \mathrm{do}(t^1)) - P(o^1 \mid \mathrm{do}(t^0)) = P(\mu^O_{1\mapsto 1,0\mapsto 0}) - P(\mu^O_{1\mapsto 0,0\mapsto 1}). \quad (21.4)$$

These are just marginal probabilities of the distribution $P(U^T, U^O)$, and they can therefore be written as a linear combination of the sixteen ν parameters representing the entries in this joint distribution.

The set of possible values for $\boldsymbol{\nu}$ is determined by the constraints equation (21.3), which defines eight linear equations constraining various subsets of these parameters to sum up to probabilities in the observed distribution. We also need to require that $\boldsymbol{\nu}$ be nonnegative.

Altogether, we have a linear formulation of $\mathrm{ACE}(T \to O)$ in terms of the ν parameters, and a set of linear constraints on these parameters — the equalities of equation (21.3) and the nonnegativity inequalities. We can now use linear programming techniques to obtain both the maximum and the minimum value of the function representing $\mathrm{ACE}(T \to O)$ subject to the constraints. This gives us bounds over the possible value that $\mathrm{ACE}(T \to O)$ can take in any functional causal model consistent with our observed probabilities.

In this fairly simple problem, we can even provide closed-form expressions for these bounds. Somewhat simpler bounds that are correct but not tight are:

$$\mathrm{ACE}(T \to O) \quad \geq \quad P(o^1 \mid a^1) - P(o^1 \mid a^0) - P(o^1, t^0 \mid a^1) - P(o^0, t^1 \mid a^0). \quad (21.5)$$

$$\mathrm{ACE}(T \to O) \quad \leq \quad P(o^1 \mid a^1) - P(o^1 \mid a^0) + P(o^0, t^0 \mid a^1) + P(o^1, t^1 \mid a^0). \quad (21.6)$$

Both bounds have a base quantity of $P(o^1 \mid a^1) - P(o^1 \mid a^0)$; this quantity, sometimes called the encouragement, *represents the difference in cure rate between the group that was prescribed the medication and the group that was not, ignoring the issue of how many in each group actually took the prescribed treatment. In a regime of full compliance with the treatment, the encouragement would be equivalent to $\mathrm{ACE}(T \to O)$. The correction factors in each of these equations provide a bound on the extent to which noncompliance can affect this estimate. Note that the total difference between the upper and lower bounds is*

$$P(o^0, t^0 \mid a^1) + P(o^1, t^1 \mid a^0) + P(o^1, t^0 \mid a^1) + P(o^0, t^1 \mid a^0) = P(t^0 \mid a^1) + P(t^1 \mid a^0),$$

natural bounds

which is precisely the total rate of noncompliance. These bounds are known as the natural bounds. *They are generally not as tight as the bounds we would obtain from the linear program, but they are tight in cases where no patient is a defier, that is, $P(\mu^T_{1\mapsto 0,0\mapsto 1}) = 0$. In many cases, they provide surprisingly informative bounds on the result of the intervention query, as shown in box 21.B.* ■

Box 21.B — Case Study: The Effect of Cholestyramine. *A study conducted as part of the Lipid Research Clinics Coronary Primary Prevention Trial produced data about a drug trial relating to a drug called cholestyramine. In a portion of this data set (337 subjects), subjects were randomized into two treatment groups of roughly equal size; in one group, all subjects were prescribed the drug (a^1), while in the other group, all subjects were prescribed a placebo (a^0). Patients who were in the*

control group did not have access to the drug. The cholesterol level of each patient was measured several times over the years of treatment, and the average was computed. In this case, both the actual consumption of the drug and the resulting cholesterol level are continuous-valued.

Balke and Pearl (1994a) provide a formal analysis of this data set. In their analysis, a patient was said to have received treatment (t^1) if his consumption was above the midpoint between minimal and maximal consumption among patients in the study. A patient was said to have responded to treatment (o^1) if his average cholesterol level throughout the treatment was at least 28 points lower than his measurement prior to treatment.

The resulting data exhibited the following statistics:

$$\begin{array}{ll} P(t^1, o^1 \mid a^1) = 0.473 & P(t^1, o^1 \mid a^0) = 0 \\ P(t^1, o^0 \mid a^1) = 0.139 & P(t^1, o^0 \mid a^0) = 0 \\ P(t^0, o^1 \mid a^1) = 0.073 & P(t^0, o^1 \mid a^0) = 0.081 \\ P(t^0, o^0 \mid a^1) = 0.315 & P(t^0, o^0 \mid a^0) = 0.919. \end{array}$$

The encouragement in this case is $P(o^1 \mid a^1) - P(o^1 \mid a^0) = 0.0473 + 0.073 - 0.081 = 0.465$. According to equation (21.5) and 21.6, we obtain:

$$\begin{array}{rcl} \text{ACE}(T \rightarrow O) & \geq & 0.465 - 0.073 - 0 = 0.392 \\ \text{ACE}(T \rightarrow O) & \leq & 0.465 + 0.315 + 0 = 0.78. \end{array}$$

The difference between these bounds represents the noncompliance rate of $P(t^0 \mid a^1) = 0.388$. Thus, despite the fact that 38.8 percent of the subjects deviated from their treatment protocol, we can still assert (ignoring possible errors resulting from using statistics over a limited population) that, when applied uniformly to the population, cholestyramine increases by at least 39.2 percent the probability of reducing a patient's cholesterol level by 28 points or more.

21.6 Counterfactual Queries ⋆

Part of our motivation for moving to functional causal models derived from the issue of answering counterfactual queries. We now turn our attention to such queries, and we show how functional causal models can be used to address them. As we discussed, similar issues arise in the context of diagnostic reasoning, where we wish to reason about the probabilities of various events after taking a repair action; see box 21.C.

21.6.1 Twinned Networks

Recall that a counterfactual query considers a scenario that actually took place in the real world and a corresponding scenario in a counterfactual world, where we modify the chain of events via some causal intervention. Thus, we are actually reasoning in parallel about two different worlds: the real one, and the counterfactual one. A variable X may take on one value in the real world, and a different value in the counterfactual world. To distinguish between these two values, we use X to denote the random variable X in the true world, and X' to denote X in the *counterfactual world*. Intuitively, both the real and the counterfactual worlds are governed

counterfactual world

by the same causal model, except that one involves an intervention.

However, a critical property of the desired output for a counterfactual query was that it involves minimal modification to the true world. As a degenerate case, consider the following counterfactual query relating to example 21.18: "A patient in the study took the prescribed treatment and did not get well. Would he have gotten well had we forced him to comply with the treatment?" Formally, we can write this query as $P(O' = o^1 \mid T = t^1, O = o^0, do(T' := t^1))$. Our intuition in this case says that we change nothing in the counterfactual world, and so the outcome should be exactly the same. But even this obvious assertion has significant consequences. Clearly, we cannot simply generate a new random assignment to the variables in the counterfactual world. For example, we want the patient's prescribed course of treatment to be the same in both worlds. We also want his response to the treatment to be the same.

The notion of response variables allows us to make this intuition very precise. A response variable U_X in a functional causal model summarizes the effect of all of the stochastic choices that can influence the choice of value for the endogenous variable X. When moving to the counterfactual world, all of these stochastic choices should remain the same. Thus, we assume that the values of the response variables are selected at random in the real world and preserved unchanged in the counterfactual world. Note that, when the counterfactual query includes a nondegenerate intervention $do(X := x')$ — one where x' is different from the value of x in the true world — the values of endogenous variables can change in the counterfactual world. However, the mechanism by which they choose these values remains constant.

Example 21.22

Consider the counterfactual query $P(O' = o^1 \mid T = t^0, O = o^0, \text{do}(T' := t^1))$. This query represents a situation where the patient did not take the prescribed treatment and did not get well, and we wish to determine the probability that the patient would have gotten well had he complied with the treatment. Assume that the true world is such that the patient was assigned to the treatment group, so that $U^A = a^1$. Assume also that he is in the "helped" category, so that the response variable $U^O = \mu^O_{1\mapsto 1, 0\mapsto 0}$. As we assumed, both of these are conserved in the counterfactual world. Thus, given $T' = t^1$, and applying the deterministic functions specified by the response variables, we obtain that $A' = a^1$, $T' = t^1$, and $O' = o^1$, so that the patient would have recovered had he complied. ■

In general, of course, we do not know the value of the response variables in the real world. However, a functional causal model specifies a prior distribution over their values. Some values are not consistent with our observations in the real scenario, so the distribution over the response variables is conditioned accordingly. The resulting posterior can be used for inference in the counterfactual world.

Example 21.23

Consider again our clinical trial of example 21.18. Assume that the trial is randomized, so that $P(U^A = a^1) = P(U^A = a^0) = 0.5$. Also, assume that the medication is unavailable outside the treatment group, so that $P(U^T = \mu^T_{1\mapsto 1, 0\mapsto 1}) = 0$. One possible joint distribution for $P(U^T, U^O)$ is shown in figure 21.5b. Assume that we have a patient who, in the real world, was assigned to the treatment group, did not comply with the treatment, and did not get well; we are interested in the probability that he would have gotten well had he complied. Thus, our query is:

$$P(O' = o^1 \mid A = a^1, T = t^0, O = o^0, \text{do}(T' := t^1)).$$

Our observations in the real world are consistent only with the following values for the response

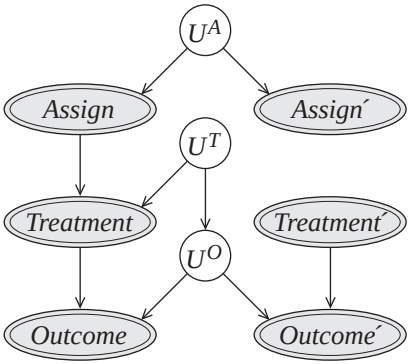


Figure 21.6 Twinned counterfactual network for figure 21.5, with an intervention at T'.

variables: U^T can be "defier" or "never taker"; U^O can be "helped" or "never well." Conditioning our joint distribution, we obtain the following joint posterior:

U^T	U^O	*probability*
defier	*helped*	$4/13$
defier	*never well*	$2/13$
never taker	*helped*	$6/13$
never taker	*never well*	$1/13$

In the counterfactual world, we intervene by forcing the patient to comply with the treatment. Therefore, his outcome is determined by the value that U^O defines for t^1. This value is o^1 when the patient is of category "helped," and o^0 when he is of category "never well." Using the posterior we computed, we conclude that the answer to the query is $10/13$. ■

This type of reasoning can be obtained as a direct consequence of inference in a joint causal network that incorporates both the real and the counterfactual world. Specifically, consider the following construction:

Definition 21.4 counterfactual twinned network

Let $\mathcal{C}$ be a functional causal model over the endogenous variables $\mathcal{X}$ and the corresponding response variables $\mathcal{U} = \{U^X : X \in \mathcal{X}\}$. We define the counterfactual twinned network *to be a functional causal model over the variables $\mathcal{X} \cup \mathcal{U} \cup \{X' : X \in \mathcal{X}\}$, such that:*

- *if $\mathrm{Pa}_X = \boldsymbol{Y}$, then $\mathrm{Pa}_{X'} = \boldsymbol{Y}'$,*
- *X' also has the response variable U^X as a parent,*
- *the deterministic function governing X' is identical to that of X.* ■

We can answer counterfactual queries for our original causal model by constructing the twinned causal model and then answering the counterfactual query as an intervention query in that network, in exactly the same way as we would for any functional causal network.

Example 21.24

Returning to the query of example 21.23, we first construct the twinned network for figure 21.5, and then use it to answer the intervention query $P(O' = o^1 \mid A = a^1, T = t^0, O = o^0, \text{do}(T' := t^1))$ as a simple intervention query in the resulting network. Specifically, we mutilate the counterfactual network so as to eliminate incoming arcs into T', condition on our evidence $A = a^1, T = t^0, O = o^0$, and simply run probabilistic inference. The network is shown in figure 21.6. It is straightforward to verify that the answer is precisely what we obtained in the preceding analysis. ■

Note that this approach is very general, and it allows us to answer a broad range of queries. We can account for interventions in the real network as well as the counterfactual one and allow for evidence in both. For example, we can ask, "Assume that we had a patient in the control (placebo) group who did not get well; if I had assigned that patient to the treatment group and observed that he did not get well, what is the probability that he complied with treatment (in the counterfactual world)?" This query is formally written as $P(T' = t^1 \mid A = a^0, O = o^0, do(A' := a^1), O' = o^0)$, and it can be answered using inference in the twinned network.

troubleshooting

Box 21.C — Case Study: Persistence Networks for Diagnosis. *One real-world application of the twinned-network analysis arises in the setting of* troubleshooting *(see also box 5.A). In this application, described in Breese and Heckerman (1996), we have a network with multiple (unobserved) variables $X_1, \ldots, X_k$ that denote the possible failure of various components in a system. Our task is to repair the system. In an ideal setting, our repair actions can correspond to ideal interventions: we take one of the failure variables X_k and set it to a value denoting no failure:* $\text{do}(X_i := x_i^0)$.

Now, assume that we have some set of observations $\boldsymbol{e}$ about the current state of the system, and want to evaluate the benefit of some repair action $\text{do}(X_i := x_i^0)$. *How do we compute the probability that the repair action fixes the problem, or the distribution over the remaining failure variables following the repair action? It is not difficult to see that an intervention query is not appropriate in this setting: The evidence that we had about the failure symptoms prior to the repair is highly relevant for determining these probabilities, and it must be taken into account when computing the posterior following the intervention. For example, if the probability of x_i^0 (i not broken) was very high given $\boldsymbol{e}$, chances are the system is still broken following the repair. The right query, to determine the postintervention distribution for a variable Y, is a counterfactual one:*

$$P(Y' \mid \boldsymbol{e}, \text{do}(X_i' := x_i^0)).$$

Under certain (fairly strong) assumptions — noisy-or CPDs and a single fault only — one can avoid constructing the twinned network and significantly reduce the cost of this computation. (See exercise 21.6.) We return to the issue of troubleshooting networks in box 23.C.

21.6.2 Bounds on Counterfactual Queries

Although a functional causal model gives us enormous ability to answer sophisticated counterfactual queries, such a model is rarely available, as we discussed. However, we can apply the same idea as in section 21.5 to provide bounds on the probabilities of the response variables,

and hence on the answers to counterfactual queries. We demonstrate this approach on a fictitious example, which also serves to illustrate the differences between different types of causal analysis.

Example 21.25

The marketer of PeptAid, an antacid medication, randomly mailed out product samples to 10 percent of the population. A follow-on market survey determined, for each individual, whether he or she received the sample (A), whether he or she took it (T), and whether he or she subsequently developed an ulcer (O). The accumulated data exhibited the following statistics:

$$\begin{array}{ll} P(t^1, o^1 \mid a^1) = 0.14 & P(t^1, o^1 \mid a^0) = 0.32 \\ P(t^1, o^0 \mid a^1) = 0.17 & P(t^1, o^0 \mid a^0) = 0.32 \\ P(t^0, o^1 \mid a^1) = 0.67 & P(t^0, o^1 \mid a^0) = 0.04 \\ P(t^0, o^0 \mid a^1) = 0.02 & P(t^0, o^0 \mid a^0) = 0.32. \end{array}$$

The functional causal model for this situation is identical to that of figure 21.5a.

Examining these numbers, we see a strong correlation between individuals who consumed PeptAid and those who developed ulcers: $P(o^1 \mid t^1) = 0.5$, whereas $P(o^1 \mid t^0) = 0.26$. Moreover, the probability of developing ulcers was 45 percent greater in individuals who received the PeptAid samples than in those who did not: $P(o^1 \mid a^1) = 0.81$, whereas $P(o^1 \mid a^0) = 0.36$. Thus, using the observational data alone, we might conclude that PeptAid causes ulcers, and that its manufacturer is legally liable for damages to the affected population.

As we discussed in example 21.2, an immediate counterargument is that the high positive correlation is due to some latent common cause such as preulcer discomfort. Indeed, one can show that, in this case, PeptAid actually helps reduce the risk of ulcers: a causal analysis along the lines of example 21.21 shows that

$$-0.23 \leq \text{ACE}(T \to O) \leq -0.15.$$

That is, the average causal effect of PeptAid consumption is to reduce an individual's chance of getting an ulcer by at least 15 percent.

However, now consider a particular patient George who received the sample, consumed it, and subsequently developed an ulcer. We would like to know whether the patient would not have developed the ulcer had he not received the sample. In this case, the relevant query is a counterfactual, which has the form:

$$P(O' = o^0 \mid A = a^1, T = t^1, O = o^1, \text{do}(A' := a^0)).$$

Given our evidence $A = a^1, T = t^1, O = o^1$, only the responses "complier" and "always taker" are possible for U^T, and only the responses "hurt" and "never well" for U^O (where here "never well" means "always ulcer", or $\mu^O_{1\mapsto 1, 0\mapsto 1}$). Of these, only the combination ("complier,""hurt") is consistent with the query assertion $O' = o^0$: if George is a "never well," he would have developed ulcers regardless; similarly, if George is an "always taker," he would have taken PeptAid regardless, with the same outcome. We conclude that the probability of interest is equal to:

$$\frac{P(U^T = \textit{complier}, U^O = \textit{hurt})}{P(T = t^1, O = o^1 \mid A = a^1)}.$$

Because the numerator is fixed, this expression is linear in the $\boldsymbol{\nu}$ parameters, and so we can compute bounds using linear programming. The resulting bounds show that:

$$P(O' = o^0 \mid A = a^1, T = t^1, O = o^1, \text{do}(A' := a^0)) \geq 0.93;$$

thus, at least 93 percent of patients in George's category — those who received PeptAid, consumed it, and developed an ulcer — would not have developed an ulcer had they not received the sample! ■

This example illustrates the subtleties of causal analysis, and the huge differences between the answers to apparently similar queries. Thus, care must be taken when applying causal analysis to understand precisely which query it is that we really wish to answer.

21.7 Learning Causal Models

So far, we have focused on the problem of using a given causal model to answer causal queries such as intervention or counterfactual queries. In this section, we discuss the problem of learning a causal model from data.

As usual, there are several axes along which we can partition the space of learning tasks.

Perhaps the most fundamental axis is the notion of what we mean by a causal model. Most obvious is a causal network with standard CPDs. Here, we are essentially learning a standard Bayesian network, except that we are willing to ascribe to it causal semantics and use it to answer interventional queries. However, we can also consider other choices. For example, at one extreme, we can consider a functional causal model, where our network has response variables and a fully specified parameterization for them; clearly, this problem is far more challenging, and such rich models are much more difficult to identify from available data. At the other extreme, we can simplify our problem considerably by abstracting away all details of the parameterization and focus solely on determining the causal structure.

Second, we need to determine whether we are given the structure and so have to deal only with parameter estimation, or whether we also have to learn the structure.

observational data

interventional data

A third axis is the type of data from which we learn the model. In the case of probabilistic models, we assumed that our data consist of instances sampled randomly from some generating distribution P^*. Such data are called *observational*. In the context of causal models, we may also have access to *interventional data*, where (some or all of) our data instances correspond to cases where we intervene in the model by setting the values of some variables and observe the values of the others. As we will discuss, interventional data provide significant power in disambiguating different causal models that can lead to identical observational patterns. Unfortunately, in many cases, interventions are difficult to perform, and sometimes are even illegal (or immoral), whereas observational data are usually much more plentiful.

A final axis involves the assumptions that we are willing to make regarding the presence of factors that can confound causal effects. If we assume that there are no confounding factors — latent variables or selection bias — the problem of identifying causal structure becomes significantly simpler. In particular, under fairly benign assumptions, we can fully delineate the cases in which we can infer causal direction from observational data. When we allow these confounding factors, the problem becomes significantly harder, and the set of cases where we can reach nontrivial conclusions becomes much smaller.

Of course, not all of the entries in this many-dimensional grid are interesting, and not all have been explored. To structure our discussion, we begin by focusing on the task of learning a causal model, which allows us to infer the causal direction for the interactions between the variables and to answer interventional queries. We consider first the case where there are no confounding factors, so that all relevant variables are observed in the data. We discuss both the case of learning only from observational data, then introduce the use of interventional data. We then discuss the challenges associated with latent variables and approaches for dealing with

them, albeit in a limited way. Finally, we move to the task of learning a functional causal model, where even determining the parameterization for a fixed structure is far from trivial.

21.7.1 Learning Causal Models without Confounding Factors

We now consider the problem of learning a causal model from data. At some level, it appears that there is very little to say here that we have not already said. After all, a causal model is essentially a Bayesian network, and we have already devoted three chapters in this book to the problem of learning the structure and the parameters of such networks from data. Although many of the techniques we developed in these chapters will be useful to us here, they do not provide a full solution to the problem of learning causal models. To understand why, recall that our objective in the task of learning probabilistic models was simply to provide a good fit to the true underlying distribution. Any model that fit that distribution (reasonably well) was an adequate solution.

As we discussed in section 21.1.2, there are many different probabilistic models that can give rise to the same marginal distribution over the observed variables. While these models are (in some sense) equivalent with respect to probabilistic queries, as causal models, they generally give rise to very different conclusions for causal queries. Unfortunately, distinguishing between these structures is often impossible. Thus, our task in this section is to obtain the strongest possible conclusion about the causal models that could have given rise to the observed data.

21.7.1.1 Key Assumptions

There are two key assumptions that underlie methods that learn causal models from observational data. They relate the independence assumptions that hold in the true underlying distribution P^* with the causal relationships between the variables in the domain. Assume that P^* is derived from a causal network whose structure is the graph $\mathcal{G}^*$.

causal Markov assumption

The first, known as the *causal Markov assumption*, asserts that, in P^*, each variable is conditionally independent of its non-effects (direct or indirect) given its direct causes. Thus, each variable is conditionally independent of its nondescendants given its parents in $\mathcal{G}^*$. This assumption is precisely the same as the local Markov assumptions of definition 3.1, except that arcs are given a causal interpretation. We can thus restate this assumption as asserting that the causal network $\mathcal{G}^*$ is an I-map for the distribution P^*.

While the difference between the local Markov and causal Markov assumptions might appear purely syntactic, it is fundamental from a philosophical perspective. The local Markov assumptions for Bayesian networks are simply phenomenological: they state properties that a particular distribution has. The causal Markov assumption makes a statement about the world: If we relate variables by the "causes" relationship, these independence assumptions will hold in the empirical distribution we observe in the world.

The justification for this assumption is that causality is local in time and space, so that the direct causes of a variable (stochastically) determine its value. Current quantum theory and experiments show that this assumption does not hold at the quantum level, where there are nonlocal variables that appear to have a direct causal effect on others. While these cases do not imply that the causal Markov assumption does not hold, they do suggest that we may see more violations of this assumption at the quantum level. However, in practice, the causal Markov

assumption appears to be a reasonable working assumption in most macroscopic systems.

faithfulness assumption

The second assumption is the *faithfulness assumption*, which states that the only conditional independencies in P^* are those that arise from d-separation in the corresponding causal graph $\mathcal{G}^*$. When combined with the causal Markov assumption, the consequence is that $\mathcal{G}^*$ is a perfect map of P^*. As we discussed in section 3.4.2, there are many examples where the faithfulness assumption is violated, so that there are independencies in P^* not implied by the structure of $\mathcal{G}^*$; however, these correspond to particular parameter values, which (as stated in theorem 3.5) are a set of measure zero within the space of all possible parameterizations. As we discussed, there are still cases where one of these parameterizations arises naturally. However, for the purposes of learning causality, we generally need to make both of these assumptions.

21.7.1.2 Identifying the Model

With these assumptions in hand, we can now assume that our data set consists of samples from P^*, and our task is simply to identify a perfect map for P^*. Of course, as we observed in section 3.4.2, the perfect map for a distribution P^* is not unique. **Because we might have several different structures that are equivalent in the independence assumptions they impose, we cannot, based on observational data alone, distinguish between them. Thus, we cannot, in general, determine a unique causal structure for our domain, even given infinite data.** For the purpose of answering probabilistic queries, this limitation is irrelevant: any of these structures will perform equally well. However, for causal queries, determining the correct direction of the edges in the network is critical. Thus, at best, we can hope to identify the equivalence class of $\mathcal{G}^*$.

constraint-based structure learning

The class of *constraint-based structure learning* methods is suitable to this task. Here, we take the independencies observed in the empirical distribution and consider them as representative of P^*. Given enough data instances, the empirical distribution $\hat{P}$ will reflect exactly the independencies that hold in P^*, so that an independence oracle will provide accurate answers about independencies in P^*. The task now reduces to that of identifying an I-equivalence class that is consistent with these observed independencies. For the case without confounding factors, we have already discussed such a method: the Build-PDAG procedure described in section 18.2.

class PDAG

Recall that Build-PDAG constructs a *class PDAG*, which represents an entire I-equivalence class of network structures. In the class PDAG, an edge is oriented $X \rightarrow Y$ if and only if it is oriented in that way in every graph that is a member of the I-equivalence class. Thus, the algorithm does not make unwarranted conclusions about directionality of edges.

Even with this conservative approach, we can often infer the directionality of certain edges.

Example 21.26

Assume that our underlying distribution P^* is represented by the causal structure of the Student *network shown in figure 3.3. Assuming that the empirical distribution $\hat{P}$ reflects the independencies in P^*, the* Build-PDAG *will return the PDAG shown in figure 21.7a. Intuitively, we can infer the causal directions for the arcs $D \rightarrow G$ and $I \rightarrow G$ because of the v-structure at G; we can infer the causal direction for $G \rightarrow J$ because the opposite orientation $J \rightarrow G$ would create a v-structure involving J, which induces a different set of independencies. However, we are unable to infer a causal direction for the edge I—S, because both orientations are consistent with the observed independencies.* ■

In some sense, the constraint-based approaches are ideally suited to inferring causal direction;

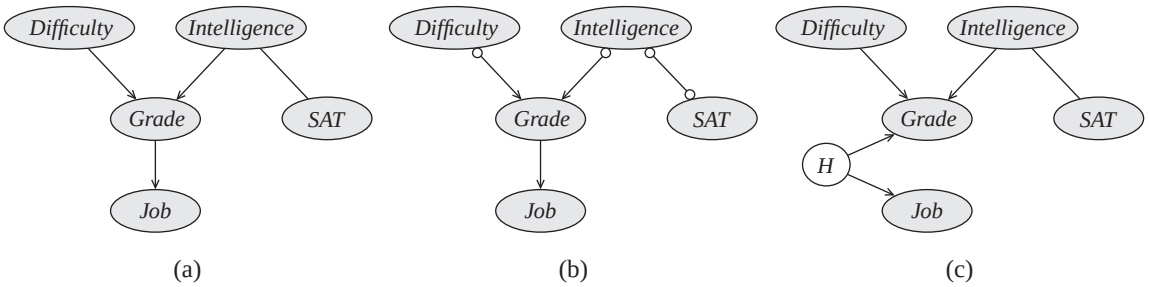


Figure 21.7 Models corresponding to the equivalence class of the Student **network.** (a) A PDAG, representing its equivalence class when all relevant variables are observed. (b) A PAG, representing its equivalence class when latent variables are a possibility. (c) An unsuccessful attempt to "undirect" the $G \rightarrow J$ arc.

in fact, these algorithms were mostly developed for the purpose of causal discovery. However, as we discussed in section 18.2, these approaches are fairly sensitive to mistakes in the independence tests over the distribution. This property can make them somewhat brittle, especially when the data set size is small relative to the number of variables.

Score-based methods allow us to factor in our confidence in different independencies, finding a solution that is more globally consistent. Thus, an alternative approach is to apply model selection with some appropriate score and then construct the I-equivalence class of the network $\mathcal{G}$ produced by the structure learning algorithm. The I-equivalence class can be represented by a PDAG, which can be constructed simply by applying the procedure Build-PDAG to $\mathcal{G}$.

Bayesian model averaging

A better solution, however, accounts for the fact that the highest-scoring network is not generally the only viable hypothesis. **In many cases, especially when data are scarce, there can be several *non-equivalent* network structures that all have a reasonably high posterior probability. Thus, a better approach for causal discovery is to use *Bayesian model averaging***, as described in section 18.5. The result is a distribution over different network structures that allows us to encode our uncertainty not only about the orientation of an edge, but also about the presence or absence of an edge. Furthermore, we obtain numerical confidence measures in these network features. Thus, whereas the PDAG representation prevents us from orienting an edge $X \rightarrow Y$ even if there is only a single member in the equivalence class that has the opposite orientation, the Bayesian model averaging approach allows us to quantify the overall probability mass of structures in which an edge exists and is oriented in a particular direction.

Although constraint-based methods and Bayesian methods are often viewed as competing solutions, one successful approach is to combine them, using a constraint-based method to initialize a graph structure, and then using Bayesian methods to refine it. This approach exploits the strengths of both methods. It uses the global nature of the constraint-based methods to avoid local maxima, and it uses the ability of the Bayesian methods to avoid making irreversible decisions about independencies that may be incorrect due to noise in the data.

21.7.2 Learning from Interventional Data

So far, we have focused on the task of learning causal models from observational data alone. In the causal setting, a natural question is to consider data that are obtained (at least partly) from interventional queries. That is, some of our data cases are obtained in a situation where we intervene in the model, sampled from the mutilated network corresponding to an intervention. One important situation where such data are often available is scientific discovery, where the data we obtain can be either a measurement of an existing system or a system that was subjected to perturbations.

Although both constraint-based and score-based approaches can be applied in this setting, constraint-based approaches are not as commonly used. Although it is straightforward to define the independencies associated with the mutilated network, we generally do not have enough data instances for any given type of intervention to measure reliably whether these independencies hold. Score-based approaches are much more flexible at combining data from diverse interventions. We therefore focus our discussion on that setting.

To formalize our analysis, we assume that each data instance in $\mathcal{D}$ is specified by an intervention $do(\boldsymbol{Z}[m] := \boldsymbol{z}[m])$; for each such data case, we have a fully observed data instance $\mathcal{X}[m] = \xi[m]$. As usual in a score-based setting, our first task is to define the likelihood function. Consider first the probability of a single instance $P(\xi \mid do(\boldsymbol{Z} := \boldsymbol{Z}), \mathcal{C})$. This term is defined in terms of the mutilated network $\mathcal{C}_{\boldsymbol{Z}=\boldsymbol{z}}$. In this network, the distribution of each variable $Z \in \boldsymbol{Z}$ is defined by the intervention, so that $Z = z$ with probability 1 and therefore is also the value we necessarily see in ξ. The variables not in $\boldsymbol{Z}$ are subject to their normal probabilistic model, as specified in $\mathcal{C}$. Letting $\boldsymbol{u}_i$ be the assignment to Pa_{X_i} in ξ, we obtain:

$$P(\xi \mid do(\boldsymbol{Z} := \boldsymbol{z}), \mathcal{C}) = \prod_{X_i \notin \boldsymbol{Z}} P(x_i \mid \boldsymbol{u}_i).$$

sufficient statistics

In the case of table-CPDs (see exercise 21.8 for another example), it follows that the *sufficient statistics* for this type of likelihood function are:

$$M[x_i; \boldsymbol{u}_i] = \sum_{m \,:\, X_i \notin \boldsymbol{Z}[m]} \boldsymbol{1}\{X_i[m] = x_i, \mathrm{Pa}_{X_i}[m] = \boldsymbol{u}_i\}, \tag{21.7}$$

for an assignment x_i to X_i and $\boldsymbol{u}_i$ to Pa_{X_i}. This sufficient statistic counts the number of occurrences of this event, *in data instances where there is no intervention at* X_i. Unlike our original sufficient statistic $M[x_i, \boldsymbol{u}_i]$, this definition of the sufficient statistic treats X_i differently from its parents, hence the change in notation.

It now follows that:

$$L(\mathcal{C} : \mathcal{D}) = \prod_{i=1}^{n} \prod_{x_i \in Val(X_i), \boldsymbol{u}_i \in Val(\mathrm{Pa}_{X_i})} \theta_{x_i|\boldsymbol{u}_i}^{M[x_i;\boldsymbol{u}_i]}. \tag{21.8}$$

Because this likelihood function has the same functional form as our original likelihood function, we can proceed to apply any of the likelihood-based approaches described in earlier chapters. We can perform maximum likelihood estimation or incorporate a prior to define a Bayesian posterior over the parameters. We can also define a Bayesian (or other likelihood-based) score in

order to perform model selection or model averaging. The formulas and derivations are exactly the same, only with the new sufficient statistics.

Importantly, **in this framework, we can now distinguish between I-equivalent models, which are indistinguishable given observational data alone.** In particular, consider a network over two variables X, Y, and assume that we have interventional data at either X, Y, or both. As we just mentioned, the sufficient statistics are asymmetrical in the parent and child, so that $M[X;Y]$, for the network $Y \to X$, is different from $M[Y;X]$, for the network $X \to Y$. Therefore, although the two networks are I-equivalent, the likelihood function can be different for the two networks, allowing us to select one over the other.

Example 21.27

Consider the task of learning a causal model over the variables X, Y. Assume that we have the samples with the sufficient statistics shown in the following table:

Intervention	x^1, y^1	x^1, y^0	x^0, y^1	x^0, y^0
None	4	1	1	4
$\text{do}(X := x^1)$	2	0	0	0
$\text{do}(Y := y^1)$	1	0	1	0

The observational data suggest that each of X and Y are (roughly) uniformly distributed, but that they are correlated with each other. The interventional data, although limited, suggest that, when we intervene at X, Y tends to follow, but when we intervene at Y, X is unaffected. These intuitions suggest that the causal model $X \to Y$ is more plausible. Indeed, computing the sufficient statistics for the model $X \to Y$ and these data instances, we obtain:

$$\begin{aligned}
M[x^1] &= M[x^1y^1 \mid \text{None}] + M[x^1y^0 \mid \text{None}] + M[x^1y^1 \mid \text{do}(y^1)] = 4 + 1 + 1 \\
M[x^0] &= M[x^0y^1 \mid \text{None}] + M[x^0y^0 \mid \text{None}] + M[x^0y^1 \mid \text{do}(y^1)] = 1 + 4 + 1 \\
M[y^1; x^1] &= M[x^1y^1 \mid \text{None}] + M[x^1y^1 \mid \text{do}(x^1)] = 4 + 2 \\
M[y^0; x^1] &= M[x^1y^0 \mid \text{None}] = 1 \\
M[y^1; x^0] &= M[x^0y^1 \mid \text{None}] = 1 \\
M[y^0; x^0] &= M[x^0y^0 \mid \text{None}] = 4.
\end{aligned}$$

Importantly, we note that, unlike the purely observational case, $M[y^1; x^1] + M[y^0; x^1] \neq M[x^1]$; this is because different data instances contribute to the different counts, depending on the variable at which the intervention takes place.

We can now compute the maximum likelihood parameters for this model as $\theta_{x^1} = 0.5$, $\theta_{y^1|x^1} = 6/7$, $\theta_{y^1|x^0} = 1/5$. The log-likelihood of the data is then:

$$\begin{aligned}
&M[x^1] \log \theta_{x^1} + M[x^0] \log \theta_{x^0} + \\
&M[y^1; x^1] \log \theta_{y^1|x^1} + M[y^0; x^1] \log \theta_{y^0|x^1} + M[y^1; x^0] \log \theta_{y^1|x^0} + M[y^0; x^0] \log \theta_{y^0|x^0},
\end{aligned}$$

which equals -19.75.

We can analogously execute the same steps for the causal model $Y \to X$, where our sufficient statistics would have the form $M[y]$ and $M[x; y]$, each utilizing a different set of instances. Overall, we would obtain a log-likelihood of -21.41, which is lower than for the causal model $X \to Y$. Thus, the log-likelihood of the two causal models is different, even though they are I-equivalent as probabilistic models. Moreover, the causal model that is consistent with our intuitions is the one that obtains the highest score. ■

We also note that **an intervention at X can help disambiguate parts of the network not directly adjacent to X.** For example, assume that the true network is a chain $X_1 \rightarrow X_2 \rightarrow \ldots \rightarrow X_n$. There are n I-equivalent directed graphs (where we root the graph at X_i for $i = 1, \ldots, n$). Interventions at any X_i can reveal that $X_{i+1}, \ldots, X_n$ all respond to an intervention at X_i, whereas $X_1, \ldots, X_{i-1}$ do not. Although these experiments would not fully disambiguate the causal structure, they would help direct all of the edges from X_i toward any of its descendants. Perhaps less intuitive is that they also help direct edges that are *not* downstream of our intervention. For example, if X_{i-1} does not respond to an intervention at X_i, but we are convinced that they are directly correlated, we now have more confidence that we can direct the edge $X_{i-1} \rightarrow X_i$. Importantly, directing some edges can have repercussions on others, as we saw in section 3.4.3. Indeed, in practice, a series of interventions at some subset of variables can significantly help disambiguate the directionality of many of the edges; see box 21.D.

Box 21.D — Case Study: Learning Cellular Networks from Intervention Data. *As we mentioned earlier, one of the important settings where interventional data arise naturally is scientific discovery. One application where causal network discovery has been applied is to the task of* cellular network reconstruction. *In the central paradigm of molecular biology, a gene in a cell (as encoded in DNA) is* expressed *to produce mRNA, which in turn is translated to produce protein, which performs its cellular function. The different steps of this process are carefully regulated. There are proteins whose task it is to regulate the expression of their* target genes*; others change the activity level of proteins by a physical change to the protein itself. Unraveling the structure of these networks is a key problem in cell biology. Causal network learning has been successfully applied to this task in a variety of ways.*

cellular network reconstruction

One important type of cellular network is a signaling network*, where a signaling protein physically modifies the structure of a target in a process called* phosphorylation*, thereby changing its activity level.* Fluorescence microscopy *can be used to measure the level of a* phosphoprotein *— a particular protein in a particular phosphorylation state. The phosphoprotein is fused to a fluorescent marker of a particular color. The fluorescence level of a cell, for a given color channel, indicates the level of the phosphoprotein fused with that color marker. Current technology allows us to measure simultaneously the levels of a small number of phosphoproteins, at the level of single cells. These data provide a unique opportunity to measure the activity levels of several proteins within individual cells, and thereby, we hope, to determine the causal network that underlies their interactions.*

Sachs et al. (2005) measured eleven phosphoproteins in a signaling pathway in human T-cells under nine different perturbation conditions. Of these conditions, two were general perturbations, but the remaining seven activated or inhibited particular phosphoproteins, and hence could be viewed as ideal interventions. Overall, 5,400 measurements were obtained over these nine conditions. The continuous measurements for each gene were discretized using a k-means algorithm, and the system was modeled as a Bayesian network where the variables are the levels of the phosphoproteins and the edges are the causal connections between them. The network was learned using standard score-based search, using a Bayesian score based on the interventional likelihood of equation (21.8). To obtain a measure of confidence in the edges of the learned structure, confidence estimation was performed using a bootstrap *method, where the same learning procedure was applied to different training sets, each sampled randomly, with replacement, from the original data set. This procedure*

bootstrap

gave rise to an ensemble of networks, each with its own structure. The confidence associated with an edge was then estimated as the fraction of the learned networks that contained the edge.

The result of this procedure gave rise to a network with seventeen high-confidence causal arcs between various components. A comparison to the known literature for this pathway revealed that fifteen of the seventeen edges were well established in the literature; the other two were novel but had supporting evidence in at least one literature citation. Only three well-established connections were missed by this analysis. Moreover, in all but one case, the direction of causal influence was correctly inferred. One of the two novel predictions was subsequently tested and validated in a wet-lab experiment. This finding suggested a new interaction between two pathways.

The use of a single global model for inferring the causal connections played an important role in the quality of the results. For example, because causal directions of arcs are often compelled by their interaction with other arcs within the overall structure, the learning algorithm was able to detect correctly causal influences from proteins that were not perturbed in the assay. In other cases, strong correlations did not lead to the inclusion of direct arcs in the model, since the correlations were well explained by indirect pathways. Importantly, although the data were modeled as fully observed, many relevant proteins were not actually measured. In such cases, the resulting indirect paths gave rise to direct edges in the learned network.

Various characteristics of this data set played an important role in the quality of the results. For example, the application of learning to a curtailed data set consisting solely of 1,200 observational data points gave rise to only ten arcs, all undirected, of which eight were expected or reported; ten of the established arcs were missing. Thus, the availability of interventional data played an important role in the accurate reconstruction of the network, especially the directionality of the edges. Another experiment showed the value of single-cell measurements, as compared to an equal-size data set each of whose instances is an average over a population of cells. This result suggests that the cell population is heterogeneous, so that averaging destroys much of the signal present in the data.

Nevertheless, the same techniques were also applied, with some success, to a data set that lacks these enabling properties. These data consist of gene expression measurements — measurements of mRNA levels for different genes, each collected from a population of cells. Here, data were collected from approximately 300 experiments, each acquired from a strain with a different gene deleted. Such perturbations are well modeled as ideal interventions, and hence they allow the use of the same techniques. This data set poses significant challenges: there are only 300 measurements (one for each perturbation experiment), but close to 6,000 variables (genes); the population averaging of each sample obscures much of the signal; and mRNA levels are a much weaker surrogate for activity level than direct protein measurements. Nevertheless, by focusing attention on subgraphs where many edges had high confidence, it was possible to reconstruct correctly some known pathways.

The main limitation of these techniques is the assumption of acyclicity, which does not hold for cellular networks. Nevertheless, these results suggest that causal-network learning, combined with appropriate data, can provide a viable approach for uncovering cellular pathways of different types.

21.7.3 Dealing with Latent Variables ⋆

So far, we have discussed the learning task in the setting where we have no confounding factors. The situation is more complicated if we have confounding effects such as latent variables or selection bias. In this case, the samples in our empirical distribution are generated from a "partial view" of P^*, where some variables have been marginalized out, and others perhaps instantiated to particular values. Because we do not know the set of confounding variables, there is an infinite set of networks that could have given rise to exactly the same set of dependencies over the observable variables. For example, $X \to Y$, $X \to H \to Y$, $X \to H \to H' \to Y$, and so on are completely indistinguishable in terms of their effect on X, Y (assuming that the hidden variables H, H' do not affect other variables). The task of determining a causal model in this case seems unreasonably daunting.

In the remainder of this section, we describe solutions to the problem of discovering causal structure in presence of confounding effects that induce noncausal correlations between the observed variables. These confounding effects include latent variables and selection bias. Although there are methods that cover both of these problems, the treatment of the latter is significantly more complex. Moreover, although selection bias clearly occurs, latent variables are almost ubiquitous, and they have therefore been the focus of more work. We therefore restrict our discussion to the case of learning models with latent variables.

21.7.3.1 Score-Based Approaches

A first thought is to try to learn a model with hidden variables using score-based methods such as those we discussed in chapter 19. The implementation of this idea, however, requires significant care. First, we generally do not know where in the model we need to introduce hidden variables. In fact, we can have an unbounded number of latent variables in the model. Moreover, as we saw, causal conclusions can be quite sensitive to local maxima or to design decisions such as the number of values of the hidden variable. We note, again, that probabilistic conclusions are also somewhat sensitive to these issues, but significantly less so, since models that achieve the same marginals over the observed variables are equivalent with respect to probabilistic queries, but not with respect to causal queries. Nevertheless, when we have some strong prior knowledge about the possible number and placement of hidden variables, and we apply our analysis with care, we can learn informative causal models.

21.7.3.2 Constraint-Based Approaches

An alternative solution is to use constraint-based approaches that try to use the independence properties of the distribution to learn the structure of the causal model, *including* the placement of the hidden variables. Here, as for the fully observable case, the independencies in a distribution only determine a structure up to equivalence. However, in the case of latent variables, we observe independence relationships only between the observable variables. We therefore need to introduce a notion of the independencies induced over the observable variables. We say that a directed acyclic graph $\mathcal{G}$ is a *latent variable network* over $\mathcal{X}$ if it is a causal network structure over $\mathcal{X} \cup \mathcal{H}$, where $\mathcal{H}$ is some arbitrarily large set of (latent) variables disjoint from $\mathcal{X}$.

latent variable network

Definition 21.5

Let $\mathcal{G}$ be a latent variable network over $\mathcal{X}$. We define $\mathcal{I}_{\mathcal{X}}(\mathcal{G})$ to be the set of independencies $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z}) \in \mathcal{I}(\mathcal{G})$ for $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \subset \mathcal{X}$. ■

We can now define a notion of I-equivalence over the observable variables.

Definition 21.6

$\mathrm{I}_{\mathcal{X}}$-equivalence

Let $\mathcal{G}_1, \mathcal{G}_2$ be two latent variable networks over $\mathcal{X}$ (not necessarily over the same set of latent variables). We say that $\mathcal{G}_1$ and $\mathcal{G}_2$ are $\mathrm{I}_{\mathcal{X}}$-equivalent *if $\mathcal{I}_{\mathcal{X}}(\mathcal{G}_1) = \mathcal{I}_{\mathcal{X}}(\mathcal{G}_2)$.* ■

Clearly, we cannot explicitly enumerate the $\mathrm{I}_{\mathcal{X}}$-equivalence class of a graph. Even in the fully observable case, this equivalence class can be quite large; when we allow latent variables, the equivalence class is generally infinite, since we can have an unbounded number of latent variables placed in a variety of configurations. The constraint-based methods sidestep this difficulty by searching over a space of graphs over the observable variables alone. Like PDAGs, an edge in these graphs can represent different types of relationships between its endpoints.

Definition 21.7

partial ancestral graph

Let $\boldsymbol{\mathcal{G}}$ be an $\mathrm{I}_{\mathcal{X}}$-equivalence class of latent variable networks over $\mathcal{X}$. A partial ancestral graph *(PAG) $\mathcal{P}$ over $\mathcal{X}$ is a graph whose nodes correspond to $\mathcal{X}$, and whose edges represent the dependencies in $\boldsymbol{\mathcal{G}}$. The presence of an edge between X and Y in $\mathcal{P}$ corresponds to the existence, in each $\mathcal{G} \in \boldsymbol{\mathcal{G}}$, of an active trail between X and Y that utilizes only latent variables. Edges have three types of endpoints: $-$, $>$, and $\circ$; these endpoints on the Y end of an edge between X and Y have the following meanings:*

- *An arrowhead $>$ implies that Y is not an ancestor of X in any graph in $\boldsymbol{\mathcal{G}}$.*
- *A straight end $-$ implies that Y is an ancestor of X in all graphs in $\boldsymbol{\mathcal{G}}$.*
- *A circle $\circ$ implies that neither of the two previous cases holds.* ■

The interpretation of the different edge types is as follows: An edge $X \rightarrow Y$ has (almost) the standard meaning: X is an ancestor of Y in all graphs in $\boldsymbol{\mathcal{G}}$, and Y is not an ancestor of X in any graph. Thus, each graph in $\boldsymbol{\mathcal{G}}$ contains a directed path from X to Y. However, some graphs may also contain trail where a latent variable is an ancestor of both. Thus, for example, the edge $S \rightarrow C$ would represent both of the networks in figure 21.A.1a,b when both G and T are latent.

An edge $X \leftrightarrow Y$ means that X is never an ancestor of Y, and Y is never an ancestor of X; thus, the edge must be due to the presence of a latent common cause. Note that an undirected edge X—Y is illegal relative to this definition, since it is inconsistent with the acyclicity of the graphs in $\boldsymbol{\mathcal{G}}$. An edge $X \circ\!\!\rightarrow Y$ means that Y is not an ancestor of X in any graph, but X is an ancestor of Y in some, but not all, graphs.

Figure 21.8 shows an example PAG, along with several members of the (infinite) equivalence class that it represents. All of the graphs in the equivalence class have one or more active trails between X and Y, none of which are directed from Y to X.

At first glance, it might appear that the presence of latent variables completely eliminates our ability to infer causal direction. After all, any edge can be ascribed to an indirect correlation via a latent variable. However, somewhat surprisingly, there are configurations where we can infer a causal orientation to an edge.

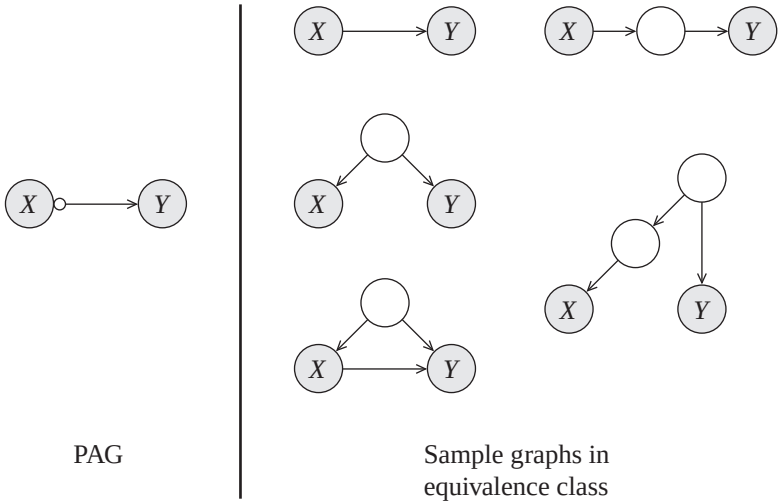


Figure 21.8 Example PAG (left), along with several members of the (infinite) equivalence class that it represents. All of the graphs in the equivalence class have one or more active trails between X and Y, none of which are directed from Y to X.

Example 21.28

Consider again the learning problem in example 21.26, but where we now allow for the presence of latent variables. Figure 21.7b shows the PAG reflecting the equivalence class of our original network of figure 3.3. Not surprisingly, we cannot reach any conclusions about the edge between I and S. The edge between D and G can arise both from a directed path from D to G and from the presence of a latent variable that is the parent of both. However, a directed path from G to D would not result in a marginal independence between D and I, and a dependence given G. Thus, we have an arrowhead $>$ on the G side of this edge. The same analysis holds for the edge between I and G.

Most interesting, however, is the directed edge $G \rightarrow J$, which asserts that, in any graph in $\mathcal{G}$, there is a directed path from G to J. To understand why, let us try to explain the correlation between G and J by introducing a common latent parent (see figure 21.7c). This model is not $I_{\mathcal{X}}$-equivalent to our original network, because it implies that J is marginally independent of I and D. More generally, we can conclude that J must be a descendant of I and D, because observing J renders them dependent. Because G renders J independent of I, it must block any directed path from I to J. It follows that there is a directed path from G to J in every member of $\mathcal{G}$. In fact, in this case, we can reach the stronger conclusions that all the trails between these two variables are directed paths from G to J. Thus, in this particular case, the causal influence of G on J is simply $P(J \mid G)$, which we can obtain directly from observational data alone. ■

This example gives intuition for how we might determine a PAG structure for a given distribution. The algorithm for constructing a PAG for a distribution P proceeds along similar lines to the algorithm for constructing PDAGs, described in section 3.4.3 and 18.2. The full algorithm for learning PAGs is quite intricate, and we do not provide a full description of it, but only

give some high-level intuition. The algorithm has two main phases. The first phase constructs an undirected graph over the observed variables, representing direct probabilistic interactions between them in P. In general, we want to connect X and Y with a direct edge if and only if there is no subset $\boldsymbol{Z}$ of $\mathcal{X} - \{X, Y\}$ such that $P \models (X \perp Y \mid \boldsymbol{Z})$. Of course, we cannot actually enumerate over the exponentially many possible subsets $\boldsymbol{Z} \subset \mathcal{X}$. As in Build-PMap-Skeleton, we both bound the size of the possible separating set and prune sets that cannot be separating sets given our current knowledge about the adjacency structure of the graph. The second phase of the algorithm orients as many edges as possible, using reasoning similar to the ideas used for PDAGs, but extended to deal with the confounding effect of latent variables.

The PAG-learning algorithm offers similar (albeit somewhat weaker) guarantees than the PDAG construction algorithm. In particular, one cannot show that the edge orientation rules are complete, that is, produce the strongest possible conclusion about edge orientation that is consistent with the equivalence class. However, one can show that all of the latent variable networks over $\mathcal{X}$ that are consistent with a PAG produced by this algorithm are $\mathrm{I}_{\mathcal{X}}$-equivalent.

Importantly, we note that a PAG is only a partial graph structure, and not a full model; thus, it cannot be used directly for answering causal queries. One possible solution is to use the score-based techniques we described to parameterize the causal model. This approach, however, is fraught with difficulties: First, we have the standard difficulties of using EM to learn parameters for hidden variables; an even bigger problem is that the PAG provides no guidance about the number of latent variables, their domain, or the edges between them.

Another alternative is to use the methods of section 21.3 and 21.5, which use a learned causal structure with latent variables, in conjunction with statistics over the observable data, to answer causal queries. We note, however, that these methods require a known connectivity structure among the hidden variables, whereas the learned PAG does not specify this structure. Nevertheless, if we are willing to introduce some assumptions about this structure, these algorithms may be usable. We return to this option in section 21.7.4, where we discuss more robust ways of estimating the answers to such queries.

21.7.4 Learning Functional Causal Models ⋆

Finally, we turn to the question of learning a much richer class of models: functional causal models, where we have a set of response variables with their associated parameters. As we discussed, these models have two distinct uses. The first is to answer a broader range of causal queries, such as counterfactual queries or queries regarding the average causal effect. The second is to avoid, to some extent, the infinite space of possible configurations of latent variables. As we discussed in section 21.4, a fully specified functional causal model summarizes the effect of all of the exogenous variables on the variables in our model, and thereby, within a finite description, specifies the causal behavior of our endogenous variables. Thus, rather than select a set of concrete latent variables with a particular domain and parameterization for each one, we use response variables to summarize all of the possibilities. Our conclusions in this case are robust, and they apply for any true underlying model of the latent variables.

The difficulty, of course, is that a functional causal model is a very complex object. The parameterization of a response variable is generally exponentially larger than the parameterization of a CPD for the corresponding endogenous variable. Moreover, the data we are given provide the outcome in only one of the exponentially many counterfactual cases given by the response

variable. In this section, we describe one approach for learning with these issues.

Recall that a functional causal model is parameterized by a joint distribution $P(\mathcal{U})$ over the response variables. The local models of the endogenous variables are, by definition, deterministic functions of the response variables. A response variable U^X for a variable X with parents $\boldsymbol{Y}$ is a discrete random variable, whose domain is the space of all functions $\mu(\boldsymbol{Y})$ from $Val(\boldsymbol{Y})$ to $Val(X)$. The joint distribution $P(\mathcal{U})$ is encoded by a Bayesian network. We first focus on the case where the structure of the network is known, and our task is only to learn the parameterization. We then briefly discuss the issue of structure learning.

Consider first the simple case, where U^X has no parents. In this case, we can parameterize U^X using a multinomial distribution $\boldsymbol{\nu}^X = (\nu_1^X, \ldots, \nu_m^X)$, where $m = |Val(U^X)|$. In the Bayesian approach, we would take this parameter to be itself a random variable and introduce an appropriate prior, such as a Dirichlet distribution, over $\boldsymbol{\nu}^X$. More generally, we can use the techniques of section 17.3 to parameterize the entire Bayesian network over $\mathcal{U}$.

Our goal is then to compute the posterior distribution $P(\mathcal{U} \mid \mathcal{D})$; this posterior defines an answer to both intervention queries and counterfactual queries. Consider a general causal query $P(\phi \mid \psi)$, where ϕ and ψ may contain both real and counterfactual variables, and ψ may also contain interventions. We have that:

$$P(\phi \mid \psi, \mathcal{D}) = \int P(\phi \mid \psi, \boldsymbol{\nu}) P(\boldsymbol{\nu} \mid \psi, \mathcal{D}) d\boldsymbol{\nu}.$$

Assuming that $\mathcal{D}$ is reasonably large, we can approximate $P(\boldsymbol{\nu} \mid \psi, \mathcal{D})$ as $P(\boldsymbol{\nu} \mid \mathcal{D})$. Thus, to answer a causal query, we simply take the expectation of the answer to the query over the posterior parameter distribution $P(\boldsymbol{\nu} \mid \mathcal{D})$.

The main difficulty in this procedure is that the data set $\mathcal{D}$ is only partly observable: even if we fully observe the endogenous variables $\mathcal{X}$, the response variables $\mathcal{U}$ are not directly observed. As we saw, an observed assignment ξ to $\mathcal{X}$ limits the set of possible values to $\mathcal{U}$ to the subset of functions consistent with ξ. In particular, if $x, \boldsymbol{y}$ is the assignment to $X, \boldsymbol{Y}$ in ξ, then U^X is restricted to the set of possible functions μ for which $\mu(\boldsymbol{y}) = x$, a set that is exponentially large. Thus, to apply Bayesian learning, we must use techniques that approximate the posterior parameter distribution $P(\boldsymbol{\nu} \mid \mathcal{D})$. In section 19.3, we discussed several approaches to approximating this posterior, including variational Bayesian learning and MCMC methods. Both can be applied in this setting as well.

Thus, in principle, this approach is a straightforward application of techniques we have already discussed. However, because of the size of the space, the use of functional causal models in general and in this case in particular is feasible only for fairly small models.

When we also need to learn the structure of the functional causal model, the situation becomes even more complex, since the problem is one of structure learning in the presence of hidden variables. One approach is to use the constraint-based approach of section 21.7.3.2 to learn a structure involving latent variables, and then the approach described here for filling in the parameters. A second approach is to use one of the methods of section 19.4. However, there is an important issue that arises in this approach: Recall that a response variable for a variable X specifies the value of X for each configuration of its endogenous parents $\boldsymbol{U}$. Thus, as our structure learning algorithm adapts the structure of the network, the *domain* of the response variables changes; for example, if our search adds a parent to X, the domain of U^X changes. Thus, when performing the search, we would need to recompute the posterior parameter dis-

tribution and thereby the score after every structure change to the model. However, under certain independence assumptions, we can use score decomposability to reduce significantly the amount of recomputation required; see exercise 21.10.

21.8 Summary

In this chapter, we addressed the issue of ascribing a causal interpretation to a Bayesian network. While a causal interpretation does not provide any additional capabilities in terms of answering standard probabilistic queries, it provides the basic framework for answering *causal queries* — queries involving interventions in the world. We provided semantics for causal models in terms of the causal mechanism by which a variable's value is generated. An intervention query can then be viewed as a substitution of the existing causal mechanism with one that simply forces the intervened variable to take on a particular value.

We discussed the greater sensitivity of causal queries to the specifics of the model, including the specific orientations of the arcs and the presence of latent variables. Latent variables are particularly tricky, since they can induce correlations between the variables in the model that are hard to distinguish from causal relationships. These issues make the identification of a causal model much more difficult than the selection of an adequate probabilistic model.

We presented a class of situations in which a causal query can be answered exactly, using only a distribution over the observable variables, even when the model as a whole is not identifiable. In other cases, even if the query is not fully identifiable, we can often provide surprisingly strong bounds over the answer to a causal query.

Besides intervention queries, causal models can also be used to answer counterfactual queries — queries about a sequence of events that we know to be different from the sequence that actually took place in the world. To answer such queries, we need to make explicit the random choices made in selecting the values of variables in the model; these random choices need to be preserved between the real and counterfactual worlds in order to maintain the correct semantics for the idea of a counterfactual. Functional causal models allow us to represent these random choices in a finite way, regardless of the (potentially unbounded) number of latent variables in the domain. We showed how to use functional causal models to answer counterfactual queries. While these models are even harder to identify than standard causal models, the techniques for partially identifying causal queries can also be used in this case.

Finally, we discussed the controversial and challenging problem of learning causal models from data. Much of the work in this area has been devoted to the problem of inferring causal models from observational data alone. This problem is very challenging, especially when we allow for the possible presence of latent variables. We described both constraint-based and Bayesian methods for learning causal models from data, and we discussed their advantages and disadvantages.

Causality is a fundamental concept when reasoning about many topics, ranging from specific scientific applications to commonsense reasoning. Causal networks provide a framework for performing this type of reasoning in a systematic and principled way. On the other side, the learning algorithms we described, by combining prior knowledge about domain structure with empirical data, can help us identify a more accurate causal structure, and perhaps obtain a better understanding of the domain. There are many possible applications of this framework in

the realm of scientific discovery, both in the physical and life sciences and in the social sciences.

21.9 Relevant Literature

The use of functional equations to encode causal processes dates back at least as far as the work of Wright (1921), who used them to model genetic inheritance. Wright (1934) also used directed graphs to represent causal structures.

The view of Bayesian networks as encoding causal processes was present throughout much of their history, and certainly played a significant role in early work on constraint-based methods for learning network structure from data (Verma and Pearl 1990; Spirtes et al. 1991, 1993). The formal framework for viewing a Bayesian network as a causal graph was developed in the early and mid 1990s, primarily by two groups: by Spirtes, Glymour, and Scheines, and by Pearl and his students Balke and Galles. Much of this work is summarized in two seminal books: the early book of Spirtes, Glymour, and Scheines (1993) and the more recent book by Pearl (2000), on which much of the content of this chapter is based. The edited collection of Glymour and Cooper (1999) also reviews other important developments.

The use of a causal model for analyzing the effect of interventions was introduced by Pearl and Verma (1991) and Spirtes, Glymour, and Scheines (1993). The formalization of the causal calculus, which allows the simplification of intervention queries and their reformulation in terms of purely observable queries, was first presented in detail in Pearl (1995). The example on smoking and cancer was also presented there. Based on these ideas, Galles and Pearl (1995) provide an algorithm for determining the identifiability of an intervention query. Dawid (2002, 2007) provides an alternative formulation of causal intervention that makes explicit use of decision variables. This perspective, which we used in section 21.3, significantly simplifies certain aspects of causal reasoning.

The idea of making mechanisms explicit via response variables is based on ideas proposed in Rubin's theory of counterfactuals (Rubin 1974). It was introduced into the framework of causal networks by Balke and Pearl (1994b,a), and in parallel by Heckerman and Shachter (1994), who use a somewhat different framework based on influence diagrams. Balke and Pearl (1994a) describe a method that uses the distribution over the observed variables to constrain the distribution of the response variables. The PeptAid example (example 21.25) is due to Balke and Pearl (1994a), who also performed the analysis of the cholesterolymine example (box 21.B). Chickering and Pearl (1997) present a Gibbs sampling approach to Bayesian parameter estimation in causal settings.

The work on constraint-based structure learning (described in section 18.2) was first presented as an approach for learning causal networks. It was proposed and developed in the work of Verma and Pearl (1990) and in the work of Spirtes et al. (1993). Even this very early work was able to deal with latent variables. Since then, there has been significant work on extending and improving these early algorithms. Spirtes, Meek, and Richardson (1999) present a state-of-the-art algorithm for identifying a PAG from data and show that it can accommodate both latent variables and selection bias.

Heckerman, Meek, and Cooper (1999) proposed a Bayesian approach to causal discovery and the use of a Markov chain Monte Carlo algorithm for sampling structures in order to obtain probabilities of causal features.

The extension of Bayesian structure learning to a combination of observational and inter-

ventional data was first developed by Cooper and Yoo (1999). These ideas were extended and applied by Pe'er et al. (2001) to the problem of identifying regulatory networks from gene expression data, and by Sachs et al. (2005) to the problem of identifying signaling networks from fluorescent microscopy data, as described in box 21.D. Tong and Koller (2001a,b) build on these ideas in addressing the problem of *active learning* — choosing a set of interventions so as best to learn a causal network model.

active learning

21.10 Exercises

Exercise 21.1⋆

a. Prove proposition 21.2, which allows us to convert causal interventions in a query into observations.
b. An alternative condition for this proposition works in terms of the original graph $\mathcal{G}$ rather than the graph $\mathcal{G}^\dagger$. Let $\mathcal{G}_{\underline{\boldsymbol{X}}}$ denote the graph $\mathcal{G}$, minus all edges going out of nodes in $\boldsymbol{X}$. Show that the d-separation criterion used in the proposition is equivalent to requiring that $\boldsymbol{Y}$ is d-separated from $\boldsymbol{X}$ given $\boldsymbol{Z}, \boldsymbol{W}$ in the graph $\mathcal{G}_{\overline{\boldsymbol{Z}}\underline{\boldsymbol{X}}}$.

Exercise 21.2⋆

Prove proposition 21.3, which allows us to drop causal interventions from a query entirely.

Exercise 21.3⋆

For probabilistic queries, we have that

$$\min_x P(y \mid x) \leq P(y) \leq \max_x P(y \mid x).$$

Show that the same property does not hold for intervention queries. Specifically, provide an example where it is not the case that:

$$\min_x P(y \mid do(x)) \leq P(y) \leq \max_x P(y \mid do(x)).$$

Exercise 21.4⋆⋆

Show that every one of the diagrams in figure 21.3 is identifiable via the repeated application of proposition 21.1, 21.2, and 21.3.

Exercise 21.5⋆

a. Show that, in the causal model of figure 21.4g, each of the queries $P(Z_1 \mid do(X))$, $P(Z_2 \mid do(X))$, $P(Y \mid do(Z_1))$, and $P(Y \mid do(Z_2))$ are identifiable.
b. Explain why the effect of X on Y cannot be identifiable in this model.
c. Show that we can identify both $P(Y \mid do(X), do(Z_1))$ and $P(Y \mid do(X), do(Z_2))$. This example illustrates that the effect of a joint intervention may be more easily identified than the effect of each of its components.

Exercise 21.6

As we discussed in box 21.C, under certain assumptions, we can reduce the cost of performing counterfactual inference to that of a standard probabilistic query. In particular, assume that we have a system status variable X that is a noisy-or of the failure variables $X_1, \ldots, X_k$, and that there is no leak probability, so that $X = x^0$ when all $X_i = x_i^0$ (that is, X is normal when all its components are normal). Furthermore, assume that only a single X_i is in the failure mode ($X_i = x_i^1$). Show that

$$P(x'^0 \mid x^1, do(x_i^0), \boldsymbol{e}) = P(d_i^1 \mid x^1, \boldsymbol{e}),$$

where Z_i is the noisy version of X_i, as in definition 5.11.

Exercise 21.7

This exercise demonstrates computation of sufficient statistics with interventional data. The following table shows counts for different interventions.

Intervention	$x^0y^0z^0$	$x^0y^0z^1$	$x^0y^1z^0$	$x^0y^1z^1$	$x^1y^0z^0$	$x^1y^0z^1$	$x^1y^1z^0$	$x^1y^1z^1$
None	4	2	1	0	3	2	1	4
$do(X := x^0)$	3	1	2	1	0	0	0	0
$do(Y := y^0)$	7	1	0	0	2	1	0	0
$do(Z := z^0)$	1	0	1	0	1	0	1	0

Calculate $M[x^0; y^0z^0]$, $M[y^0; x^0z^0]$, and $M[x^0]$.

Exercise 21.8

Consider the problem of learning a Gaussian Bayesian network from interventional data $\mathcal{D}$. As in section 21.7.2, assume that each data instance in $\mathcal{D}$ is specified by an intervention $do(\boldsymbol{Z}[m] := \boldsymbol{z}[m])$; for each such data case, we have a fully observed data instance $\mathcal{X}[m] = \xi[m]$. Write down the sufficient statistics that would be used to score a network structure $\mathcal{G}$ from this data set.

Exercise 21.9⋆

Consider the problem of Bayesian learning for a functional causal model $\mathcal{C}$ over a set of endogenous variables $\mathcal{X}$. Assume we have a data set $\mathcal{D}$ where the endogenous variables $\mathcal{X}$ are fully observed. Describe a way for approximating the parameter posterior $P(\boldsymbol{\nu} \mid \mathcal{X})$ using collapsed Gibbs sampling. Specifically, your algorithm should sample the response variables $\mathcal{U}$ and compute a closed-form distribution over the parameters $\boldsymbol{\nu}$.

Exercise 21.10⋆⋆

Consider the problem of learning the structure of a functional causal model $\mathcal{C}$ over a set of endogenous variables $\mathcal{X}$.

a. Using your answer from exercise 21.9, construct an algorithm for learning the structure of a causal model. Describe precisely the key steps used in the algorithm, including the search steps and the use of the Gibbs sampling algorithm to evaluate the score at each step.

b. Now, assume that we are willing to stipulate that the response variables U^X for each variable X are independent. (This assumption is a very strong one, but it may be a reasonable approximation in some cases.) How can you significantly improve the learning algorithm in this case? Provide a new pseudo-code description of the algorithm, and quantify the computational gains.

Exercise 21.11⋆

causal independence

As for probabilistic independence, we can define a notion of *causal independence*: $(\boldsymbol{X} \perp_C \boldsymbol{Y} \mid \boldsymbol{Z})$ if, for any values $\boldsymbol{x}, \boldsymbol{x}' \in Val(\boldsymbol{X})$, we have that $P(\boldsymbol{Y} \mid do(\boldsymbol{Z}), do(\boldsymbol{x})) = P(\boldsymbol{Y} \mid do(\boldsymbol{Z}), do(\boldsymbol{x}'))$. (Note that, unlike probabilistic independence — $(\boldsymbol{X} \perp \boldsymbol{Y} \mid \boldsymbol{Z})$ — causal independence is not symmetric over $\boldsymbol{X}, \boldsymbol{Y}$.)

a. Is causal independence equivalent to the statement: "For any value $\boldsymbol{x} \in Val(\boldsymbol{X})$, we have that $P(\boldsymbol{Y} \mid do(\boldsymbol{Z}), do(\boldsymbol{x})) = P(\boldsymbol{Y} \mid do(\boldsymbol{Z}))$." (Hint: Use your result from exercise 21.3.)

b. Prove that $(\boldsymbol{X} \perp_C \boldsymbol{Y} \mid \boldsymbol{Z}, \boldsymbol{W})$ and $(\boldsymbol{W} \perp_C \boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{Z})$ implies that $(\boldsymbol{X}, \boldsymbol{W} \perp_C \boldsymbol{Y} \mid \boldsymbol{Z})$. Intuitively, this property states that if changing $\boldsymbol{X}$ cannot affect $P(\boldsymbol{Y})$ when $\boldsymbol{W}$ is fixed, and changing $\boldsymbol{W}$ cannot affect $P(\boldsymbol{Y})$ when $\boldsymbol{X}$ is fixed, then changing $\boldsymbol{X}$ and $\boldsymbol{Y}$ together cannot affect $P(\boldsymbol{Y})$.

Exercise 21.12⋆

We discussed the issue of trying to use data to extract causal knowledge, that is, the directionality of an influence. In this problem, we will consider the interaction between this problem and both hidden variables and selection bias.

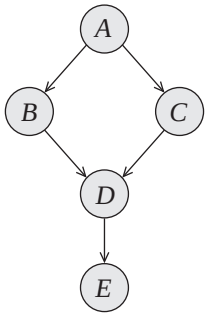


Figure 21.9 Learned causal network for exercise 21.12

Assume that our learning algorithm came up with the network in figure 21.9, which we are willing to assume is a perfect map for the distribution over the variables A, B, C, D, E. Under this assumption, among which pairs of variables between which a causal path exists in this model does there also necessarily exist a causal path ...

a. ... if we assume there are no hidden variables?

b. ... if we allow the possibility of one or more hidden variables?

c. ... if we allow for the possibility of selection bias?

For each of these options, specify the pairs for which a causal path exists, and explain why it exists in every $\mathrm{I}_{\mathcal{X}}$-equivalent structure. For the other pairs, provide an example of an $\mathrm{I}_{\mathcal{X}}$-equivalent structure for which no causal path exists.

22 *Utilities and Decisions*

We now move from the task of simply reasoning under uncertainty — reaching conclusions about the current situation from partial evidence — to the task of deciding how to act in the world. In a decision-making setting, an agent has a set of possible actions and has to choose between them. Each action can lead to one of several outcomes, which the agent can prefer to different degrees.

Most simply, the outcome of each action is known with certainty. In this case, the agent must simply select the action that leads to the outcome that is most preferred. Even this problem is far from trivial, since the set of outcomes can be large and complex and the agent must weigh different factors in determining which of the possible outcomes is most preferred. For example, when deciding which computer to buy, the agent must take into consideration the CPU speed, the amount of memory, the cost, the screen size, and many other factors. Deciding which of the possible configurations he most prefers can be quite difficult.

Even more difficult is the decision-making task in situations where the outcome of an action is not fully determined. In this case, we must take into account both the probabilities of various outcomes and the preferences of the agent between these outcomes. Here, it is not enough to determine a preference ordering between the different outcomes. We must be able to ascribe preferences to complex scenarios involving probability distributions over possible outcomes.

decision theory

The framework of *decision theory* provides a formal foundation for this type of reasoning. This framework requires that we assign numerical *utilities* to the various possible outcome, encoding the agent's preferences. In this chapter, we focus on a discussion of utilities functions and the principle of *maximum expected utility*, which is the foundation for decision making under uncertainty. In the next chapter, we discuss computationally tractable representations of an agent's decision problem and the algorithmic task of finding an optimal strategy.

22.1 Foundations: Maximizing Expected Utility

In this section, we formally describe the basic decision-making task and define the principle of maximum expected utility. We also provide a formal justification for this principle from basic axioms of rationality.

22.1.1 Decision Making Under Uncertainty

We begin with a simple motivating example.

Example 22.1

Consider a decision maker who encounters the following situation. She can invest in a high-tech company (A), where she can make a profit of \$4 million with 20 percent probability and \$0 with 80 percent probability; or she can invest in pork belly futures (B), where she can make \$3 million with 25 percent probability and \$0 with 75 percent probability. (That is, the pork belly investment is less profitable but also less risky.) In order to choose between these two investment opportunities, the investor must compare her preferences between two scenarios, each of which encodes a probability distribution over outcomes: the first scenario, which we denote π_A, can be written as $[\$4\mathit{million} : 0.2; \$0 : 0.8]$*; the second scenario, denoted π_B, has the form* $[\$3\mathit{million} : 0.25; \$0 : 0.75]$. ■

In order to ascertain which of these scenarios we prefer, it is not enough to determine that we prefer \$4 million to \$3 million to \$0. We need some way to aggregate our preferences for these outcomes with the probabilities with which we will get each of them. One approach for doing this aggregation is to assign each outcome a numerical *utility*, where a higher utility value associated with an outcome indicates that this outcome is more preferred. Importantly, however, utility values indicate more than just an ordinal preference ranking between outcomes; their numerical value is significant by itself, so that the relative values of different states tells us the strength of our preferences between them. This property allows us to combine the utility values of different states, allowing us to ascribe an *expected utility* to situations where we are uncertain about the outcome of an action. Thus, we can compare two possible actions using their expected utility, an ability critical for decision making under uncertainty.

utility

expected utility

We now formalize these intuitions.

Definition 22.1

lottery

preference over lotteries

A lottery π *over an outcome space $\mathcal{O}$ is a set $[\pi_1 : \alpha_1; \ldots; \pi_k : \alpha_k]$ such that $\alpha_1, \ldots, \alpha_k \in [0, 1]$, $\sum_i \alpha_i = 1$, and each π_i is an outcome in $\mathcal{O}$. For two lotteries π_1, π_2, if the agent* prefers π_1*, we say that $\pi_1 \succ \pi_2$. If the agent is indifferent between the two lotteries, we say that $\pi_1 \sim \pi_2$.* ■

A comparison between two different scenarios involving uncertainty over the outcomes is quite difficult for most people. At first glance, one might think that the "right" decision is the one that optimizes a person's monetary gain. However, that approach rarely reflects the preferences of the decision maker.

Example 22.2

Consider a slightly different decision-making situation. Here, the investor must decide between company C, where she earns \$3 million with certainty, and company D, where she can earn \$4 million with probability 0.8 and \$0 with probability 0.2. In other words, she is now comparing two lotteries $\pi_C = [\$3\mathit{million} : 1]$ and $\pi_D = [\$4\mathit{million} : 0.8; \$0 : 0.2]$. The expected profit of lottery D is \$3.2 million, which is larger than the profit of \$3 million from lottery C. However, a vast majority of people prefer the option of lottery C to that of lottery D. ■

The problem becomes far more complicated when one accounts for the fact that many decision-making situations involve aspects other than financial gain.

A general framework that allows us to make decisions such as these ascribes a numerical *utility* to different outcomes. An agent's utilities describe her overall preferences, which can depend not only on monetary gains and losses, but also on all other relevant aspects. Each outcome o is associated with a numerical value $U(o)$, which is a numerical encoding of the agent's "happiness" for this outcome. Importantly, utilities are not just ordinal

values, denoting the agent's preferences between the outcomes, but are actual numbers whose magnitude is meaningful. Thus, we can probabilistically aggregate utilities and compute their expectations over the different possible outcomes.

We now make these intuitions more formal.

Definition 22.2

decision-making situation

outcome

action

utility function

A decision-making situation $\mathcal{D}$ *is defined by the following elements:*

- *a set of* outcomes $\mathcal{O} = \{o_1, \ldots, o_N\}$*;*
- *a set of possible* actions *that the agent can take,* $\mathcal{A} = \{a_1, \ldots, a_K\}$*;*
- *a probabilistic outcome model* $P : \mathcal{A} \mapsto \Delta_{\mathcal{O}}$*, which defines a lottery* π_a*, which specifies a probability distribution over outcomes given that the action* a *was taken;*
- *a* utility function $U : \mathcal{O} \mapsto \mathbb{R}$*, where* $U(o)$ *is the agent's preferences for the outcome* o*.* ■

Note that the definition of an outcome can also include the action taken; outcomes that involve one action a would then get probability 0 in the lottery induced by another action a'.

Definition 22.3

MEU principle

expected utility

The principle of maximum expected utility (MEU principle) *asserts that, in a decision-making situation* $\mathcal{D}$*, we should choose the action* a *that maximizes the* expected utility*:*

$$\mathrm{EU}[\mathcal{D}[a]] = \sum_{o \in \mathcal{O}} \pi_a(o) U(o).$$

■

Example 22.3

Consider a decision situation $\mathcal{I}_F$ *where a college graduate is trying to decide whether to start up a company that builds widgets. The potential entrepreneur does not know how large the market demand for widgets really is, but he has a distribution: the demand is either* m^0*—nonexistent,* m^1*—low, or* m^2*—high, with probabilities* 0.5, 0.3, *and* 0.2 *respectively. The entrepreneur's profit, if he founds the startup, depends on the situation. If the demand is nonexistent, he loses a significant amount of money (outcome* o_1*); if it is low, he sells the company and makes a small profit (outcome* o_2*); if it is high, he goes public and makes a fortune (outcome* o_3*). If he does not found the startup, he loses nothing and earns nothing (outcome* o_0*). These outcomes might involve attributes other than money. For example, if he loses a significant amount of money, he also loses his credibility and his ability to start another company later on. Let us assume that the agent's utilities for the four outcomes are:* $U(o_0) = 0$*;* $U(o_1) = -7$*;* $U(o_2) = 5$*;* $U(o_3) = 20$*. The agent's expected utility for the action of founding the company (denoted* f^1*) is*

$$\mathrm{EU}[\mathcal{D}[f^1]] = 0.5 \cdot (-7) + 0.3 \cdot 5 + 0.2 \cdot 20 = 2.$$

His expected utility for the action of not founding the company (denoted f^0*) is* 0*. The action choice maximizing the expected utility is therefore* f^1*.* ■

Our definition of a decision-making situation is very abstract, resulting in the impression that the setting is one where an agent takes a single simple action, resulting in a single simple outcome. In fact, both actions and outcomes can be quite complex. Actions can be complete strategies involving sequences of decisions, and outcomes (as in box 22.A) can also involve multiple aspects. We will return to these issues later on.

22.1.2 Theoretical Justification ⋆

What justifies the principle of maximizing expected utility, with its associated assumption regarding the existence of a numerical utility function, as a definition of rational behavior? It turns out that there are several theoretical analyses that can be used to *prove* the existence of such a function. At a high level, these analyses postulate some set of axioms that characterize the behavior of a rational decision maker. They then show that, for any agent whose decisions abide by these postulates, there exists some utility function U such that the agent's decisions are equivalent to maximizing the expected utility relative to U.

The analysis in this chapter is based on the premise that a decision maker under uncertainty must be able to decide between different lotteries. We then make a set of assumptions about the nature of the agent's preferences over lotteries; these assumptions arguably should hold for the preferences of any rational agent. **For an agent whose preferences satisfy these axioms, we prove that there exists a utility function U such that the agent's preferences are equivalent to those obtained by maximizing the expected utility relative to U.**

We first extend the concept of a lottery.

Definition 22.4 compound lottery

A compound lottery π *over an outcome space* $\mathcal{O}$ *is a set* $[\pi_1 : \alpha_1; \ldots; \pi_k : \alpha_k]$ *such that* $\alpha_1, \ldots, \alpha_k \in [0, 1]$, $\sum_i \alpha_i = 1$, *and each* π_i *is either an outcome in* $\mathcal{O}$ *or another lottery.* ■

Example 22.4

One example of a compound lottery is a game where we first toss a coin; if it comes up heads, we get \$3 (o_1); if it comes up tails, we participate in another subgame where we draw a random card from a deck, and if it comes out spades, we get \$50 (o_2); otherwise we get nothing (o_3). This lottery would be represented as $[o_1 : 0.5; [o_2 : 0.25; o_3 : 0.75] : 0.5]$. ■

rationality postulates

We can now state the *postulates of rationality* regarding the agent's preferences over lotteries. At first glance, each these postulates seems fairly reasonable, but each of them has been subject to significant criticism and discussion in the literature.

- **(A1) Orderability:** For all lotteries π_1, π_2, either

$$(\pi_1 \prec \pi_2) \text{ or } (\pi_1 \succ \pi_2) \text{ or } (\pi_1 \sim \pi_2) \tag{22.1}$$

  This postulate asserts that an agent must know what he wants; that is, for any pair of lotteries, he must prefer one, prefer the other, or consider them to be equivalent. Note that this assumption is not a trivial one; as we discussed, it is hard for people to come up with preferences over lotteries.

- **(A2) Transitivity:** For all lotteries π_1, π_2, π_3, we have that:

$$\text{If } (\pi_1 \prec \pi_2) \text{ and } (\pi_2 \prec \pi_3) \text{ then } (\pi_1 \prec \pi_3). \tag{22.2}$$

  The transitivity postulate asserts that preferences are transitive, so that if the agent prefers lottery 1 to lottery 2 and lottery 2 to lottery 3, he also prefers lottery 1 to lottery 3. Although transitivity seems very compelling on normative grounds, it is the most frequently violated axiom in practice. One hypothesis is that these "mistakes" arise when a person is forced to make choices between inherently incomparable alternatives. The idea is that each pairwise

comparison invokes a preference response on a different "attribute" (for instance, money, time, health). Although each scale itself may be transitive, their combination need not be. A similar situation arises when the overall preference arises as an aggregate of the preferences of several individuals.

- **(A3) Continuity:** For all lotteries π_1, π_2, π_3,

  If $(\pi_1 \prec \pi_2 \prec \pi_3)$ then there exists $\alpha \in (0,1)$ such that $(\pi_2 \sim [\pi_1 : \alpha; \pi_3 : (1-\alpha)])$. (22.3)

  This postulate asserts that if π_2 is somewhere between π_1 and π_3, then there should be some lottery between π_1 and π_3, which is equivalent to π_2. For our simple Entrepreneur example, we might have that $o_0 \sim [o_1 : 0.8; o_3 : 0.2]$. This axiom excludes the possibility that one alternative is "infinitely better" than another one, in the sense that any probability mixture involving the former is preferable to the latter. It therefore captures the relationship between probabilities and preferences and the form in which they compensate for each other.

- **(A4) Monotonicity:** For all lotteries π_1, π_2, and probabilities α, β,

  $$(\pi_1 \succ \pi_2), (\alpha \geq \beta) \Rightarrow ([\pi_1 : \alpha; \pi_2 : (1-\alpha)] \succ [\pi_1 : \beta; \pi_2 : (1-\beta)]). \quad (22.4)$$

  This postulate asserts that an agent prefers that better things happen with higher probability. Again, although this attribute seems unobjectionable, it has been argued that risky behavior such as Russian roulette violates this axiom. People who choose to engage in such behavior seem to prefer a probability mixture of "life" and "death" to "life," even though they (presumably) prefer "life" to "death." This argument can be resolved by revising the outcome descriptions, incorporating the aspect of the thrill obtained by playing the game.

- **(A5) Substitutability:** For all lotteries π_1, π_2, π_3, and probabilities α,

  $$(\pi_1 \sim \pi_2) \Rightarrow ([\pi_1 : \alpha; \pi_3 : (1-\alpha)] \sim [\pi_2 : \alpha; \pi_3 : (1-\alpha)]). \quad (22.5)$$

  This axiom states that if π_1 and π_2 are equally preferred, we can substitute one for the other without changing our preferences.

- **(A6) Decomposability:** For all lotteries π_1, π_2, and probabilities α, β,

  $$[\pi_1 : \alpha, [\pi_2 : \beta, \pi_3 : (1-\beta)] : (1-\alpha)] \sim [\pi_1 : \alpha, \pi_2 : (1-\alpha)\beta, \pi_3 : (1-\alpha)(1-\beta)]. \quad (22.6)$$

  This postulate says that compound lotteries are equivalent to flat ones. For example, our lottery in example 22.4 would be equivalent to the lottery

  $$[o_1 : 0.5; o_2 : 0.125; o_3 : 0.375].$$

  Intuitively, this axiom implies that the preferences depend only on outcomes, not the process in which they are obtained. It implies that a person does not derive any additional pleasure (or displeasure) from suspense or participation in the game.

If we are willing to accept these postulates, we can derive the following result:

Theorem 22.1 *Assume that we have an agent whose preferences over lotteries satisfy the axioms (A1)–(A6). Then there exists a function $U : \mathcal{O} \mapsto \mathbb{R}$, such that, for any pair of lotteries π, π', we have that $\pi \prec \pi'$ if and only if $U(\pi) < U(\pi')$, where we define (recursively) the expected utility of any lottery as:*

$$U([\pi_1 : \alpha_1, \ldots, \pi_k : \alpha_k]) = \sum_{i=1}^{k} \alpha_i U(\pi_i).$$

That is, the utility of a lottery is simply the expectation of the utilities of its components.

PROOF Our goal is to take a preference relation $\prec$ that satisfies these axioms, and to construct a utility function U over consequences such that $\prec$ is equivalent to implementing the MEU principle over the utility function U. We take the least and most preferred outcomes $o_{\min}$ and $o_{\max}$; these outcomes are typically known as *anchor outcomes*. By orderability (Al) and transitivity (A2), such outcomes must exist. We assign $U(o_{\min}) := 0$ and $U(o_{\max}) := 1$. By orderability, we have that for any other outcome o:

anchor outcome

$$o_{\min} \preceq o \preceq o_{\max}.$$

By continuity (A3), there must exist a probability α such that

$$[o : 1] \sim [o_{\min} : (1 - \alpha); o_{\max} : \alpha] \tag{22.7}$$

We assign $U(o) := \alpha$. The axioms can then be used to show that the assignment of utilities to lotteries resulting from applying the expected utility-principle results in an ordering that is consistent with our preferences. We leave the completion of this proof as an exercise (exercise 22.1). ■

From an operational perspective, this discussion gives us a formal justification for the principle of maximum expected utility. When we have a set of outcomes, we ascribe a numerical *utility* to each one. If we have a set of actions that induce different lotteries over outcomes, we should choose the action whose *expected utility* is largest; as shown by theorem 22.1, this choice is equivalent to choosing the action that induces the lottery we most prefer.

22.2 Utility Curves

The preceding analysis shows that, under certain assumptions, a utility function must exist. However, it does not provide us with an understanding of utility functions. In this section, we take a more detailed look at the form of a utility functions and its connection to the utility function properties.

A utility function assigns numeric values to various possible outcomes. These outcomes can vary along multiple dimensions. Most obvious is monetary gain, but most settings involve other attributes as well. We begin in this section by considering the utility of simple outcomes, involving only a single attribute. We discuss the form of a utility function over a single attribute and the effects of the utility function on the agent's behavior. We focus on monetary outcomes, which are the most common and easy to understand. However, many of the issues we discuss in this section — those relating to risk attitudes and rationality — are general in their scope, and they apply also to other types of outcomes.

22.2.1 Utility of Money

Consider a decision-making situation where the outcomes are simply monetary gains or losses. In this simple setting, it is tempting to assume that the utility of an outcome is simply the amount of money gained in that outcome (with losses corresponding to negative utilities). However, as we discussed in example 22.2, most people do not always choose the outcome that maximizes their expected monetary gain. Making such a decision is not irrational; it simply implies that, for most people, their utility for an outcome is not simply the amount of money they have in that outcome.

Consider a graph whose X-axis is the monetary gain a person obtains in an outcome (with losses corresponding to negative amounts), and whose Y-axis is the person's utility for that outcome. In general, most people's utility is monotonic in money, so that they prefer outcomes with more money to outcomes with less. However, if we draw a *curve* representing a person's utility as a function of the amount of money he or she gains in an outcome, that curve is rarely a straight line. This nonlinearity is the "justification" for the rationality of the preferences we observe in practice in example 22.2.

utility curve

Example 22.5

Let c_0 represent the agent's current financial status, and assume for simplicity that he assigns a utility of 0 to c_0. If he assigns a utility of 10 to the consequence $c_0 + 3$million and 12 to the consequence $c_0 + 4$million, then the expected utility of the gamble in example 22.2 is $0.2 \cdot 0 + 0.8 \cdot 12 = 9.6 < 10$. Therefore, with this utility function, the agent's decision is completely rational. ■

Saint Petersburg paradox

A famous example of the nonlinearity of the utility of money is the *Saint Petersburg paradox*:

Example 22.6

Suppose you are offered a chance to play a game where a fair coin is tossed repeatedly until it comes up heads. If the first head appears on the nth toss, you get $\$2^n$. How much would you be willing to pay in order to play this game?

The probability of the event H_n—the first head showing up on the nth toss—is $1/2^n$. Therefore, the expected winnings from playing this game are:

$$\sum_{n=1}^{\infty} P(H_n)\text{Payoff}(H_n) = \sum_{n=1}^{\infty} \frac{1}{2^n} 2^n = 1 + 1 + 1 + \ldots = \infty.$$

Therefore, you should be willing to pay any amount to play this game. However, most people are willing to pay only about \$2. ■

Empirical psychological studies show that people's utility functions in a certain range often grow logarithmically in the amount of monetary gain. That is, the utility of the outcome c_k, corresponding to an agent's current financial status plus $\$k$, looks like $\alpha + \beta \log(k + \gamma)$. In the Saint Petersburg example, if we take $U(c_k) = \log_2 k$, we get:

$$\sum_{n=1}^{\infty} P(H_n)U(\textit{Payoff}(H_n)) = \sum_{n=1}^{\infty} \frac{1}{2^n} U(c_{2^n}) = \sum_{n=1}^{\infty} \frac{n}{2^n} = 2,$$

which is precisely the amount that most people are willing to pay in order to play this game.

In general, most people's utility function tends to be concave for positive amount of money, so that the incremental value of additional money decreases as the amount of wealth grows.

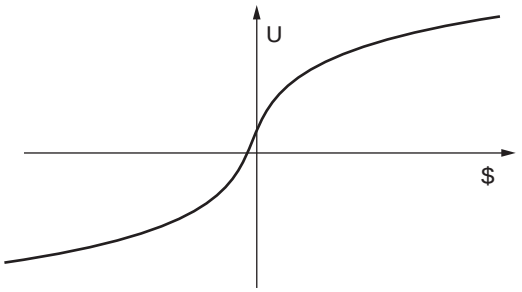


Figure 22.1 Example curve for the utility of money

Conversely, for negative amounts of money (debts), the shape of the curve often has the opposite shape, as shown in figure 22.1. Thus, for many people, going into debt of $1 million has significant negative utility, but the additional negative utility incurred by an extra $1 million of debt is a lot lower. Formally, $|U(-\$2,000,000) - U(-\$1,000,000)|$ is often significantly less than $|U(-\$1,000,000) - U(\$0)|$.

22.2.2 Attitudes Toward Risk

There is a tight connection between the form of a person's utility curve and his behavior in different decision-making situations. In particular, the shape of this curve determines the person's attitude toward *risk*. A concave function, as in figure 22.2, indicates that the agent is *risk-averse*: he prefers a sure thing to a gamble with the same payoff. Consider in more detail the risk-averse curve of figure 22.2. We see that the utility of a lottery such as $\pi = [\$1000 : 0.5, \$0 : 0.5]$ is lower than the utility of getting $500 with certainty. Indeed, risk-averse preferences are characteristic of most people, especially when large sums of money are involved. In particular, recall example 22.2, where we compared a lottery where we win $3 million with certainty to one where we win $4 million with probability 0.8. As we discussed, most people prefer the first lottery to the second, despite the fact that the expected monetary gain in the first lottery is lower. This behavior can be explained by a risk-averse utility function in that region.

risk

risk-averse

Returning to the lottery π, empirical research shows that many people are indifferent between playing π and the outcome where they get (around) $400 with certainty; that is, the utilities of the lottery and the outcome are similar. The amount $400 is called the *certainty equivalent* of the lottery. It is the amount of "sure thing" money that people are willing to trade for a lottery. The difference between the expected monetary reward of $500 and the certainty equivalent of $500 is called the *insurance premium*, and for good reason. The premium people pay to the insurance company is precisely to guarantee a sure thing (a sure small loss) as opposed to a lottery where one of the consequences involves a large negative utility (for example, the price of rebuilding the house if it burns down).

certainty equivalent

insurance premium

As we discussed, people are typically risk-averse. However, they often seek risk when the certain loss is small (relative to their financial situation). Indeed, lotteries and other forms of gambling exploit precisely this phenomenon. When the agent prefers the lottery to the certainty

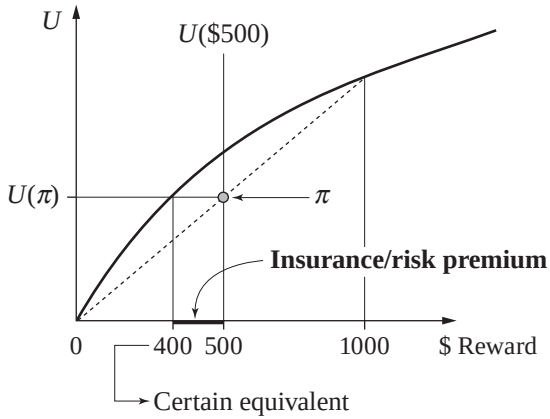


Figure 22.2 Utility curve and its consequences to an agent's attitude toward risk

risk-seeking

risk-neutral

equivalent, he is said to be *risk-seeking*, a behavior that corresponds to a curve whose shape is convex. Finally, if the agent's utility curve is linear, he is said to be *risk-neutral*. Most utility curves are locally linear, which means we can assume risk neutrality for small risks and rewards. Finally, as we noted, people are rarely consistent about risk throughout the entire monetary range: They are often risk-averse for positive gains, but can be risk-seeking for large negative amounts (going into debt). Thus, in our example, someone who is already \$10 million in debt might choose to accept a gamble on a fair coin with \$10 million payoff on heads and a \$20 million loss on tails.

22.2.3 Rationality

The framework of utility curves provides a rich language for describing complex behaviors, including risk-averse, risk-seeking, or risk-neutral behaviors. They can even change their risk preferences over the range. One may thus be tempted to conclude that, for any behavior profile, there is some utility function for which that behavior is rational. However, that conclusion turns out to be false; indeed, empirical evidence shows that people's preferences are rarely rational under our definitions.

Example 22.7

Consider again the two simple lotteries in example 22.2 and example 22.1. In the two examples, we had four lotteries:

$$\pi_A : [\$4\mathit{million} : 0.2; \$0 : 0.8]$$
$$\pi_B : [\$3\mathit{million} : 0.25; \$0 : 0.75]$$
$$\pi_C : [\$3\mathit{million} : 1]$$
$$\pi_D : [\$4\mathit{million} : 0.8; \$0 : 0.2].$$

Most people, by an overwhelming majority, prefer π_C to π_D. The opinions on π_A versus π_B are more divided, but quite a number of people prefer π_A to π_B. Each of these two preferences — $\pi_D \succ \pi_C$ and $\pi_A \succ \pi_B$ — is rational relative to some utility functions. However, their combination is not — there is no utility function that is consistent with both of these preferences. To understand why, assume (purely to simplify the presentation) that $U(\$0) = 0$. In this case, preferring π_C to π_D is equivalent to saying that

$$U(c_{3,000,000}) > 0.8 \cdot U(c_{4,000,000}).$$

On the other hand, preferring π_A to π_B is equivalent to:

$$\begin{aligned} 0.2 \cdot U(c_{4,000,000}) &> 0.25 \cdot U(c_{3,000,000}) \\ 0.8 \cdot U(c_{4,000,000}) &> U(c_{3,000,000}). \end{aligned}$$

Multiplying both sides of the first inequality by 4, we see that these two statements are directly contradictory, so that these preferences are inconsistent with decision-theoretic foundations, for any utility function. ■

Thus, people are often irrational, in that their choices do not satisfy the principle of maximum expected utility relative to any utility function. When confronted with their "irrationality," the responses of people vary. Some feel that they have learned an important lesson, which often affects other decisions that they make. For example, some subjects have been observed to cancel their automobile collision insurance and take out more life insurance. In other cases, people stick to their preferences even after seeing the expected utility analysis. These latter cases indicate that the principle of maximizing expected utility is not, in general, an adequate *descriptive model* of human behavior. As a consequence, there have been many proposals for alternative definitions of rationality that attempt to provide a better fit to the behavior of human decision makers. Although of great interest from a psychological perspective, there is no reason to believe that these frameworks will provide a better basis for building automated decision-making systems. Alternatively, we can view decision theory as a *normative model* that provides the "right" formal basis for rational behavior, regardless of human behavior. One can then argue that we should design automated decision-making systems based on these foundations; indeed, so far, most such systems have been based on the precepts of decision theory.

22.3 Utility Elicitation

22.3.1 Utility Elicitation Procedures

How do we acquire an appropriate utility function to use in a given setting? In many ways, this problem is much harder than acquiring a probabilistic model. In general, we can reasonably assume that the probabilities of chance events apply to an entire population and acquire a single probabilistic model for the whole population. For example, when constructing a medical diagnosis network, the probabilities will usually be learned from data or acquired from a human expert who understands the statistics of the domain. By contrast, utilities are inherently personal, and people often have very different preference orderings in the same situation. Thus, the utility function we use needs to be acquired for the individual person or entity for whom the decision

is being made. Moreover, as we discussed, probability values can be learned from data by observing empirical frequencies in the population. The individuality of utility values, and the fact that they are never observed directly, makes it difficult to apply similar learning methods to the utility acquisition task.

utility elicitation

There have been several methods proposed for *eliciting* utilities from people. The most classical method is the *standard gamble* method, which is based directly on the axioms of utility theory. In the proof of theorem 22.1, we selected two anchor states — our least preferred and most preferred states $s_\perp$ and $s_\top$. We then used the continuity axiom (equation (22.3)) to place each state on a continuous spectrum between these two anchor states, by finding the *indifference point* α — a probability value $\alpha \in [0, 1]$ such that $s \sim [s_\perp : (1 - \alpha); s_\top : \alpha]$.

standard gamble

indifference point

We can convert this idea to a utility elicitation procedure as follows. We select a pair of anchor states. In most cases, these are determined in advance, independently of the user. For example, in a medical decision-making situation, $s_\perp$ is often "death," whereas $s_\top$ is an immediate and complete cure. For any outcome s, we can now try to find the indifference point. It is generally assumed that we cannot ask a user to assess the value of α directly. We therefore use some procedure that searches over the space of possible α's. If $s \prec [s_\perp : (1-\alpha); s_\top : \alpha]$, we consider lower values of α, and if $s \succ [s_\perp : (1 - \alpha); s_\top : \alpha]$, we consider higher values, until we find the indifference point. Taking $U(s_\perp) = 0$ and $U(s_\top) = 1$, we simply take $U(s) = \alpha$.

The standard gamble procedure is satisfying because of its sound theoretical foundations. However, it is very difficult for people to apply in practice, especially in situations involving large numbers of outcomes. Moreover, many independent studies have shown that the final values obtained in the process of standard gamble elicitation are sensitive to the choice of anchors and to the choice of the search procedure.

Several other methods for utility elicitation have been proposed to address these limitations. For example, *time trade-off* tries to compare two outcomes: (1) t years (where t is the patient's life expectancy) in the current state of health (state s), and (2) t' years (where $t' < t$) in perfect health (the outcome $s_\top$). As in standard gamble, t' is varied until the indifferent point is reached, and the utility of the state s is taken to be proportional to t' at that point. Another method, the *visual-analog scale*, simply asks users to point out their utilities on some scale.

time trade-off

visual-analog scale

Overall, each of the methods proposed has significant limitations in practice. Moreover, the results obtained for the same individual using different methods are usually quite different, putting into question the results obtained by any method. Indeed, one might wonder whether there even exists such an object as a person's "true utility value." Nevertheless, one can still argue that decisions made for an individual using his or her own utility function (even with the imprecisions involved in the process) are generally better for that individual than decisions made using some "global" utility function determined for the entire population.

22.3.2 Utility of Human Life

Attributes whose utility function is particularly difficult to acquire are those involving human life. Clearly, such factors play a key role in medical decision-making situations. However, they also appear in a wide variety of other settings. For example, even a simple decision such as whether to replace worn-out tires for a car involves the reduced risk of death or serious injury in a car with new tires.

Because utility theory requires that we reduce all outcomes to a single numerical value, we

are forced to place a utility value on human life, placing it on the same scale as other factors, such as money. Many people find this notion morally repugnant, and some simply refuse to do so. However, the fact of the matter is that, in making decisions, one makes these trade-offs, whether consciously or unconsciously. For example, airplanes are not overhauled after each trip, even though that would clearly improve safety. Not all cars are made with airbags, even though they are known to save lives. Many people accept an extra stopover on a flight in order to save money, even though most airplane accidents happen on takeoff and landing.

Placing a utility on human life raises severe psychological and philosophical difficulties. One such difficulty relates to actions involving some probability of death. The naive approach would be to elicit the utility of the outcome *death* and then estimate the utility of an outcome involving some probability p of death as $p \cdot U(\textit{death})$. However, this approach implies that people's utility is linear in their probability of death, an assumption which is generally false. In other words, even if a person is willing to accept \$50 for an outcome involving a one-in-a-million chance of death, it does not mean that he would be willing to accept \$50 million for the outcome of *death* with certainty. Note that this example shows that, at least for this case, people violate the basic assumption of decision theory: that a person's preference for an uncertain outcome can be evaluated using expected utility, which is linear in the probabilities.

micromort

A more appropriate approach is to encode explicitly the chance of death. Thus, a key metric used to measure utilities for outcomes involving risk to human life is the *micromort* — a one-in-a-million chance of death. Several studies across a range of people have shown that a micromort is worth about \$20 in 1980 dollars, or under \$50 in today's dollars. We can consider a utility curve whose X-axis is micromorts. As for monetary utility, this curve behaves differently for positive and negative values. For example, many people are not willing to pay very much to remove a risk of death, but require significant payment in order to assume additional risk.

Micromorts are useful for evaluating situations where the primary consideration is the probability that death will occur. However, in many situations, particularly in medical decision making, the issue is not the chance of immediate death, but rather the amount of life that a person has remaining. In one approach, we can evaluate outcomes using life expectancy, where we would construct a utility curve whose X axis was the number of expected years of life. However, our preferences for outcomes are generally much more complex, since they involve not only the quantity but also the *quality of life*. In some cases, a person may prefer to live for fewer years, but in better health, than to live longer in a state where he is in pain or is unable to perform certain activities. The trade-offs here are quite complex, and highly personal.

QALY

One approach to simplifying this complex problem is by measuring outcomes using units called *QALY*s — *quality-adjusted life years*. A year of life in good health with no infirmities is worth 1 QALY. A year of life in poor health is discounted, and it is worth some fraction (less than 1) of a QALY. (In fact, some health states — for example, those involving significant pain and loss of function — may even be worth negative QALYs.) Using QALYs, we can assign a single numerical score to complex outcomes, where a person's state of health can evolve over time. QALYs are much more widely used than micromorts as a measure of utility in medical and social-policy decision making.

22.4 Utilities of Complex Outcomes

So far, we have largely focused on outcomes involving only a single attribute. In this case, we can write down our utility function as a simple table in the case of discrete outcomes, or as a curve in the case of continuous-valued outcomes (such as money). In practice, however, outcomes often involve multiple facets. In a medical decision-making situation, outcomes might involve pain and suffering, long-term quality of life, risk of death, financial burdens, and more. Even in a much "simpler" setting such as travel planning, outcomes involve money, comfort of accommodations, availability of desired activities, and more. A utility function must incorporate the importance of these different attributes, and the preferences for various values that they might take, in order to produce a single numeric value for each outcome.

Our utility function in domains such as this has to construct a single number for each outcome that depends on the values of all of the relevant variables. More precisely, assume that an outcome is described by an assignment of values to some set of variables $\boldsymbol{V} = \{V_1, \ldots, V_k\}$; we then have to define a utility function $U : Val(\boldsymbol{V}) \mapsto \mathbb{R}$. As usual, the size of this representation is exponential in k.

In the case of probabilities, we addressed the issue of exponential blowup by exploiting structure in the distribution. We showed a direct connection between independence properties of the distribution and our ability to represent it compactly as a product of smaller factors. As we now discuss, very similar ideas apply in the setting of utility functions.

Specifically, we can show a correspondence between "independence properties" among utility attributes of an agent and our ability to factor his utility function into a combination of *subutility functions*, each defined over a subset of utility attributes. A *subutility function* is a function $f : Val(\boldsymbol{Y}) \mapsto \mathbb{R}$, for some $\boldsymbol{Y} \subseteq \boldsymbol{V}$, where $\boldsymbol{Y}$ is the scope of f.

subutility function

However, the notion of independence in this setting is somewhat subtle. A utility function on its own does not induce behavior; it is a meaningful entity only in the context of a decision-making setting. Thus, our independence properties must be defined in that context as well. As we will see, there is not a single definition of independence that is obviously the right choice; several definitions are plausible, each with its own properties.

22.4.1 Preference and Utility Independence ⋆

To understand the notion of independence in decision making, we begin by considering the simpler setting, where we are making decisions in the absence of uncertainty. Here, we need only consider preferences on outcomes. Let $\boldsymbol{X}, \boldsymbol{Y}$ be a disjoint partition of our set of utility attributes $\boldsymbol{V}$. We thus have a preference ordering $\prec$ over pairs $(\boldsymbol{x}, \boldsymbol{y})$.

When can we say that $\boldsymbol{X}$ is "independent" of $\boldsymbol{Y}$? Intuitively, if we are given $\boldsymbol{Y} = \boldsymbol{y}$, we can now consider our preferences over the possible values $\boldsymbol{x}$, given that $\boldsymbol{y}$ holds. Thus, we have an induced preference ordering $\prec_{\boldsymbol{y}}$ over values $\boldsymbol{x} \in Val(\boldsymbol{X})$, where we write $\boldsymbol{x}_1 \prec_{\boldsymbol{y}} \boldsymbol{x}_2$ if $(\boldsymbol{x}_1, \boldsymbol{y}) \prec (\boldsymbol{x}_2, \boldsymbol{y})$. In general, the ordering induced by one value $\boldsymbol{y}$ is different from the ordering induced by another. We say that $\boldsymbol{X}$ is *preferentially independent* of $\boldsymbol{Y}$ if all values $\boldsymbol{y}$ induce the same ordering over $Val(\boldsymbol{X})$. More precisely:

Definition 22.5 preference independence

The set of attributes $\boldsymbol{X}$ is preferentially independent *of $\boldsymbol{Y} = \boldsymbol{V} - \boldsymbol{X}$ in $\prec$ if, for all $\boldsymbol{y}, \boldsymbol{y}' \in$*

$Val(\boldsymbol{Y})$, and for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in Val(\boldsymbol{X})$, we have that

$$\boldsymbol{x}_1 \prec_{\boldsymbol{y}} \boldsymbol{x}_2 \quad \Leftrightarrow \quad \boldsymbol{x}_1 \prec_{\boldsymbol{y}'} \boldsymbol{x}_2.$$

■

Note that preferential independence is not a symmetric relation:

Example 22.8

Consider an entrepreneur whose utility function $U(S, F)$ involves two binary-valued attributes: the success of his company (S) and the fame he gets (F). One reasonable preference ordering over outcomes might be:

$$(s^0, f^1) \prec (s^0, f^0) \prec (s^1, f^0) \prec (s^1, f^1).$$

That is, the most-preferred state is where he is successful and famous; the next-preferred state is where he is successful but not famous; then the second-next-preferred state is where he is unsuccessful but unknown; and the least-preferred state is where he is unsuccessful and (in)famous. In this preference ordering, we have that S is preferentially independent of F, since the entrepreneur prefers to be successful whether he is famous or not ($(s^0, f^1) \prec (s^1, f^1)$ and $(s^0, f^0) \prec (s^1, f^0)$). On the other hand, F is not preferentially independent of S, since the entrepreneur prefers to be famous if successful but unknown if he is unsuccessful. ■

When we move to the more complex case of reasoning under uncertainty, we compare decisions that induce lotteries over outcomes. Thus, our notion of independence must be defined relative to this more complex setting. From now on, let $\prec, U$ be a pair, where U is a utility function over $Val(\boldsymbol{V})$, and $\prec$ is the associated preference ordering for lotteries over $Val(\boldsymbol{V})$. We define independence properties for U in terms of $\prec$.

Our first task is to define the notion of a conditional preference structure, where we "fix" the value of some subset of variables $\boldsymbol{Y}$. This structure defines a preference ordering $\prec_{\boldsymbol{y}}$ for lotteries over $Val(\boldsymbol{X})$, given some particular instantiation $\boldsymbol{y}$ to $\boldsymbol{Y}$. The definition is a straightforward generalization of the one we used for preferences over outcomes:

Definition 22.6

conditional preference structure

Let $\pi_1^{\boldsymbol{X}}$ and $\pi_2^{\boldsymbol{X}}$ be two distributions over $Val(\boldsymbol{X})$. We define the conditional preference structure *$\prec_{\boldsymbol{y}}$ as follows:*

$$\pi_1^{\boldsymbol{X}} \prec_{\boldsymbol{y}} \pi_2^{\boldsymbol{X}} \quad \text{if} \quad (\pi_1^{\boldsymbol{X}}, \mathbf{1}_{\boldsymbol{y}}) \prec (\pi_2^{\boldsymbol{X}}, \mathbf{1}_{\boldsymbol{y}}),$$

where $(\pi^{\boldsymbol{X}}, \mathbf{1}_{\boldsymbol{y}})$ assigns probability $\pi^{\boldsymbol{X}}(\boldsymbol{x})$ to any assignment $(\boldsymbol{x}, \boldsymbol{y})$ and probability 0 to any assignment $(\boldsymbol{x}, \boldsymbol{y}')$ for $\boldsymbol{y}' \neq \boldsymbol{y}$. ■

In other words, the preference ordering $\prec_{\boldsymbol{y}}$ "expands" lotteries over $Val(\boldsymbol{X})$ by having $\boldsymbol{Y} = \boldsymbol{y}$ with probability 1, and then using $\prec$.

With this definition, we can now generalize preferential independence in the obvious way: $\boldsymbol{X}$ is utility independent of $\boldsymbol{Y} = \boldsymbol{V} - \boldsymbol{X}$ when conditional preferences for lotteries over $\boldsymbol{X}$ do not depend on the particular value $\boldsymbol{y}$ given to $\boldsymbol{Y}$.

Definition 22.7

utility independence

We say that $\boldsymbol{X}$ is utility independent *of $\boldsymbol{Y} = \boldsymbol{V} - \boldsymbol{X}$ if, for all $\boldsymbol{y}, \boldsymbol{y}' \in Val(\boldsymbol{Y})$, and for any pair of lotteries $\pi_1^{\boldsymbol{X}}, \pi_2^{\boldsymbol{X}}$ over $Val(\boldsymbol{X})$, we have that:*

$$\pi_1^{\boldsymbol{X}} \prec_{\boldsymbol{y}} \pi_2^{\boldsymbol{X}} \quad \Leftrightarrow \quad \pi_1^{\boldsymbol{X}} \prec_{\boldsymbol{y}'} \pi_2^{\boldsymbol{X}}.$$

■

Because utility independence is a straight generalization of preference independence, it, too, is not symmetric.

Note that utility independence is only defined for a set of variables and its complement. This limitation is inevitable in the context of decision making, since we can define preferences only over entire outcomes, and therefore every variable must be assigned a value somehow.

Different sets of utility independence assumptions give rise to different decompositions of the utility function. Most basically, for a pair $(\prec, U)$ as before, we have that:

Proposition 22.1 *A set $\boldsymbol{X}$ is utility independent of $\boldsymbol{Y} = \boldsymbol{V} - \boldsymbol{X}$ in $\prec$ if and only if U has the form:*

$$U(\boldsymbol{V}) = f(\boldsymbol{Y}) + g(\boldsymbol{Y})h(\boldsymbol{X}).$$

Note that each of the functions f, g, h has a smaller scope than our original U, and hence this representation requires (in general) fewer parameters.

From this basic theorem, we can obtain two conclusions.

Proposition 22.2 *Every subset of variables $\boldsymbol{X} \subset \boldsymbol{V}$ is utility independent of its complement if and only if there exist k functions $U_i(V_i)$ and a constant c such that*

$$U(\boldsymbol{V}) = \prod_{i=1}^{k} U_i(V_i) + c,$$

or k functions $U_i(V_i)$ such that

$$U(\boldsymbol{V}) = \sum_{i=1}^{k} U_i(V_i).$$

utility decomposition

In other words, when every subset is utility independent of its complement, the utility function *decomposes* either as a sum or as a product of subutility functions over individual variables. In this case, we need only elicit a linear number of parameters, exponentially fewer than in the general case.

If we weaken our assumption, requiring only that each variable in isolation is utility independent of its complement, we obtain a much weaker result:

Proposition 22.3 *If, for every variable $V_i \in \boldsymbol{V}$, V_i is utility independent of $\boldsymbol{V} - \{V_i\}$, then there exist k functions $U_i(V_i)$ $(i = 1, \ldots, k)$ such that U is a multilinear function (a sum of products) of the U_i's.*

For example, if $\boldsymbol{V} = \{V_1, V_2, V_3\}$, then this theorem would imply only that $U(V_1, V_2, V_3)$ can be written as

$$c_1 U_1(V_1)U_2(V_2)U_3(V_3) + c_2 U_1(V_1)U_2(V_2) + c_3 U_1(V_1)U_3(V_3) + c_4 U_2(V_2)U_3(V_3) + c_5 U_1(V_1) + c_6 U_2(V_2) + c_7 U_3(V_3).$$

In this case, the number of subutility functions is linear, but we must elicit (in the worst case) exponentially many coefficients. Note that, if the domains of the variables are large, this might still result in an overall savings in the number of parameters.

22.4.2 Additive Independence Properties

Utility independence is an elegant assumption, but the resulting decomposition of the utility function can be difficult to work with. The case of a purely additive or purely multiplicative decomposition is generally too limited, since it does not allow us to express preferences that relate to combinations of values for the variables. For example, a person might prefer to take a vacation at a beach destination, but only if the weather is good; such a preference does not easily decompose as a sum or a product of subutilities involving only individual variables.

In this section, we explore progressively richer families of utility factorizations, where the utility is encoded as a sum of subutility functions:

$$U(\boldsymbol{V}) = \sum_{i=1}^{k} U_i(\boldsymbol{Z}_i). \tag{22.8}$$

We also study how these decompositions correspond to a form of independence assumption about the utility function.

22.4.2.1 Additive Independence

additive independence

In our first decomposition, we restrict attention to decomposition as in equation (22.8), where $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ is a *disjoint* partition of $\boldsymbol{V}$. This decomposition is more restrictive than the one allowed by utility independence, since we allow a decomposition only as a sum, and not as a product. This decomposition turns out to be equivalent to a notion called *additive independence*, which has much closer ties to probabilistic independence. Roughly speaking, $\boldsymbol{X}$ and $\boldsymbol{Y}$ are additively independent if our preference function for lotteries over $\boldsymbol{V}$ depends only on the marginals over $\boldsymbol{X}$ and $\boldsymbol{Y}$. More generally, we define:

Definition 22.8 *Let $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ be a disjoint partition of $\boldsymbol{V}$. We say that $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ are* additively independent *in $\prec$ if, for any lotteries π_1, π_2 that have the same marginals on all $\boldsymbol{Z}_i$, we have that π_1 and π_2 are indifferent under $\prec$.* ■

Additive independence is strictly stronger than utility independence: For two subsets $\boldsymbol{X} \cup \boldsymbol{Y} = \boldsymbol{V}$ that are additively independent, we have both that $\boldsymbol{X}$ is utility independent of $\boldsymbol{Y}$ and $\boldsymbol{Y}$ is utility independent of $\boldsymbol{X}$. It then follows, for example, that the preference ordering in example 22.8 does not have a corresponding additively independent utility function. Additive independence is equivalent to the decomposition of U as a sum of subutilities over the $\boldsymbol{Z}_i$'s:

Theorem 22.2 *Let $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ be a disjoint partition of $\boldsymbol{V}$, and let $\prec, U$ be a corresponding pair of a preference ordering and a utility function. Then $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ are additively independent in $\prec$ if and only if U can be written as: $U(\boldsymbol{V}) = \sum_{i=1}^{k} U_i(\boldsymbol{Z}_i)$.*

Proof The "if" direction is straightforward. For the "only if" direction, consider first the case where $\boldsymbol{X}, \boldsymbol{Y}$ is a disjoint partition of $\boldsymbol{V}$, and $\boldsymbol{X}, \boldsymbol{Y}$ are additively independent in $\prec$. Let $\boldsymbol{x}, \boldsymbol{y}$ be some arbitrary fixed assignment to $\boldsymbol{X}, \boldsymbol{Y}$. Let $\boldsymbol{x}', \boldsymbol{y}'$ be any other assignment to $\boldsymbol{X}, \boldsymbol{Y}$. Let π_1 be the distribution that assigns probability 0.5 to each of $\boldsymbol{x}, \boldsymbol{y}$ and $\boldsymbol{x}', \boldsymbol{y}'$, and π_2 be the distribution that assigns probability 0.5 to each of $\boldsymbol{x}, \boldsymbol{y}'$ and $\boldsymbol{x}', \boldsymbol{y}$. These two distributions have

the same marginals over $\boldsymbol{X}$ and $\boldsymbol{Y}$. Therefore, by the assumption of additive independence, $\pi_1 \sim \pi_2$, so that

$$\begin{aligned} 0.5U(\boldsymbol{x},\boldsymbol{y}) + 0.5U(\boldsymbol{x}',\boldsymbol{y}') &= 0.5U(\boldsymbol{x},\boldsymbol{y}') + 0.5U(\boldsymbol{x}',\boldsymbol{y}) \\ U(\boldsymbol{x}',\boldsymbol{y}') &= U(\boldsymbol{x},\boldsymbol{y}') - U(\boldsymbol{x},\boldsymbol{y}) + U(\boldsymbol{x}',\boldsymbol{y}). \end{aligned} \tag{22.9}$$

Now, define $U_1(\boldsymbol{X}) = U(\boldsymbol{X},\boldsymbol{y})$ and $U_2(\boldsymbol{Y}) = U(\boldsymbol{x},\boldsymbol{Y}) - U(\boldsymbol{x},\boldsymbol{y})$. It follows directly from equation (22.9) that for any $\boldsymbol{x}', \boldsymbol{y}'$, $U(\boldsymbol{x}',\boldsymbol{y}') = U_1(\boldsymbol{x}') + U_2(\boldsymbol{y}')$, as desired. The case of a decomposition $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ follows by a simple induction on k. ■

Example 22.9

Consider a student who is deciding whether to take a difficult course. Taking the course will require a significant time investment during the semester, so it has a cost. On the other hand, taking the course will result in a more impressive résumé, making the student more likely to get a good job with a high salary after she graduates. The student's utility might depend on the two attributes T (taking the course) and J (the quality of the job obtained). The two attributes are plausibly additively independent, so that we can express the student's utility as $U_1(T) + U_2(J)$. Note that this independence of the utility function is completely unrelated to any possible probabilistic (in)dependencies. For example, taking the class is definitely correlated probabilistically with the student's job prospects, so T and J are dependent as probabilistic attributes but additively independent as utility attributes. ■

In general, however, additive independence is a strong notion that rarely holds in practice.

Example 22.10

Consider a student planning his course load for the next semester. His utility might depend on two attributes — how interesting the courses are (I), and how much time he has to devote to class work versus social activities (T). It is quite plausible that these two attributes are not utility independent, because the student might be more willing to spend significant time on class work if the material is interesting. ■

Example 22.11

Consider the task of making travel reservations, and the two attributes H — the quality of one's hotel — and W — the weather. Even these two seemingly unrelated attributes might not be additively independent, because the pleasantness of one's hotel room is (perhaps) more important when one has to spend more time in it on account of bad weather. ■

22.4.2.2 Conditional Additive Independence

The preceding discussion provides a strong argument for extending additive independence to the case of nondisjoint subsets. For this extension, we turn to probability distributions for intuition: In a sense, additive independence is analogous to marginal independence. We therefore wish to construct a notion analogous to conditional independence:

Definition 22.9

CA-independence

Let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be a disjoint partition of $\boldsymbol{V}$. We say that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally additively independent (CA-independent) *given $\boldsymbol{Z}$ in $\succ$ if, for every assignment $\boldsymbol{z}$ to $\boldsymbol{Z}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$ are additively independent in the conditional preference structure $\succ_{\boldsymbol{z}}$.* ■

The CA-independence condition is equivalent to an assumption that the utility decomposes with overlapping subsets:

Proposition 22.4 *Let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be a disjoint partition of $\boldsymbol{V}$, and let $\prec, U$ be a corresponding pair of a preference ordering and a utility function. Then $\boldsymbol{X}$ and $\boldsymbol{Y}$ are CA-independent given $\boldsymbol{Z}$ in $\prec$ if and only if U can be written as:*

$$U(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) = U_1(\boldsymbol{X}, \boldsymbol{Z}) + U_2(\boldsymbol{Y}, \boldsymbol{Z}).$$

The proof is straightforward and is left as an exercise (exercise 22.2).

Example 22.12 *Consider again example 22.10, but now we add an attribute F representing how much fun the student has in his free time (for example, does he have a lot of friends and hobbies that he enjoys?). Given an assignment to T, which determines how much time the student has to devote to work versus social activities, it is quite reasonable to assume that I and F are additively independent. Thus, we can write $U(I, T, F)$ as $U_1(I, T) + U_2(T, F)$.* ■

Based on this result, we can prove an important theorem that allows us to view a utility function in terms of a graphical model. **Specifically, we associate a utility function with an undirected graph, like a Markov network. As in probabilistic graphical models, the separation properties in the graph encode the CA-independencies in the utility function. Conversely, the utility function decomposes additively along the maximal cliques in the network.** Formally, we define the two types of relationships between a pair $(\prec, U)$ and an undirected graph:

Definition 22.10 CAI-map *We say that $\mathcal{H}$ is an* CAI-map *for $\prec$ if, for any disjoint partition $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ of $\boldsymbol{V}$, if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are separated in $\mathcal{H}$ given $\boldsymbol{Z}$, we have that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are CA-independent in $\prec$ given $\boldsymbol{Z}$.* ■

Definition 22.11 utility factorization *We say that a utility function U* factorizes *according to $\mathcal{H}$ if we can write U as a sum*

$$U(\boldsymbol{V}) = \sum_{c=1}^{k} U_c(\boldsymbol{C}_c),$$

where $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_k$ are the maximal cliques in $\mathcal{H}$. ■

We can now show the same type of equivalence between these two definitions as we did for probability distributions. The first theorem goes from factorization to independencies, showing that a factorization of the utility function according to a network $\mathcal{H}$ implies that it satisfies the independence properties implied by the network. It is analogous to theorem 3.2 for Bayesian networks and theorem 4.1 for Markov networks.

Theorem 22.3 *Let $(\prec, U)$ be a corresponding pair of a preference function and a utility function. If U factorizes according to $\mathcal{H}$, then $\mathcal{H}$ is a CAI-map for $\prec$.*

Proof The proof of this result follows immediately from proposition 22.4. Assume that U factorizes according to $\mathcal{H}$, so that $U = \sum_c U_c(\boldsymbol{C}_c)$. Any $\boldsymbol{C}_c$ cannot involve variables from both $\boldsymbol{X}$ and $\boldsymbol{Y}$. Thus, we can divide the cliques into two subsets: $\mathcal{C}_1$, which involve only variables in

$\boldsymbol{X}, \boldsymbol{Z}$, and $\mathcal{C}_2$, which involve only variables in $\boldsymbol{Y}, \boldsymbol{Z}$. Letting $U_i = \sum_{c \in \mathcal{C}_i} U_c(\boldsymbol{C}_c)$, for $i = 1, 2$, we have that $U(\boldsymbol{V}) = U_1(\boldsymbol{X}, \boldsymbol{Z}) + U_2(\boldsymbol{Y}, \boldsymbol{Z})$, precisely the condition in proposition 22.4. The desired CA-independence follows. ■

The converse result asserts that any utility function that satisfies the CA-independence properties associated with the network can be factorized over the network's cliques. It is analogous to theorem 3.1 for Bayesian networks and theorem 4.2 for Markov networks.

Theorem 22.4 *Let $(\prec, U)$ be a corresponding pair of a preference function and a utility function. If $\mathcal{H}$ is a CAI-map for $\prec$, then U factorizes according to $\mathcal{H}$.*

Hammersley-Clifford theorem

Although it is possible to prove this result directly, it also follows from the analogous result for probability distributions (the *Hammersley-Clifford theorem* — theorem 4.2). The basic idea is to construct a probability distribution by exponentiating U and then show that CA-independence properties for U imply corresponding probabilistic conditional independence properties for P:

Lemma 22.1 *Let U be a utility function, and define $P(\boldsymbol{V}) \propto \exp(U(\boldsymbol{V}))$. For a disjoint partition $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ of $\boldsymbol{V}$, we have that $\boldsymbol{X}$ and $\boldsymbol{Y}$ are CA-independent given $\boldsymbol{Z}$ in U if and only if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent given $\boldsymbol{Z}$ in P.*

The proof is left as an exercise (exercise 22.3).

Based on this correspondence, many of the results and algorithms of chapter 4 now apply without change to utility functions. In particular, the proof of theorem 22.4 follows immediately (see exercise 22.4).

minimal CAI-map

As for probabilistic models, we can consider the task of constructing a graphical model that reflects the independencies that hold for a utility function. Specifically, we define $\mathcal{H}$ to be a *minimal CAI-map* if it is a CAI-map from which no further edges can be removed without rendering it not a CAI-map. Our goal is the construction of an undirected graph which is a minimal CAI-map for a utility function U. We addressed exactly the same problem in the context of probability functions in section 4.3.3 and provided two algorithms. One was based on checking pairwise independencies of the form $(X \perp Y \mid \mathcal{X} - \{X, Y\})$. The other was based on checking local (Markov blanket) independencies of the form $(X \perp \mathcal{X} - \{X\} - \boldsymbol{U} \mid \boldsymbol{U})$. Importantly, both of these types of independencies involve a disjoint and exhaustive partition of the set of variables into three subsets. Thus, we can apply these procedures without change using CA-independencies.

Because of the equivalence of lemma 22.1, and because $P \propto \exp(U)$ is a positive distribution, all of the results in section 4.3.3 hold without change. In particular, we can show that either of the two procedures described produces the unique minimal CAI-map for U. Indeed, we can

perfect CAI-map

prove an even stronger result: The unique minimal CAI-map $\mathcal{H}$ for U is a *perfect CAI-map* for U, in the sense that any CA-independence that holds for U is implied by separation in $\mathcal{H}$:

Theorem 22.5 *Let $\mathcal{H}$ be any minimal CAI-map for U, and let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be a disjoint partition of $\boldsymbol{V}$. Then if $\boldsymbol{X}$ is CA-independent of $\boldsymbol{Y}$ given $\boldsymbol{Z}$ in U, then $\boldsymbol{X}$ is separated from $\boldsymbol{Y}$ by $\boldsymbol{Z}$ in $\mathcal{H}$.*

The proof is left as an exercise (exercise 22.5).

completeness

Note that this result is a strong *completeness* property, allowing us to read any CA-independence

that holds in the utility function from a graph. One might wonder why a similar result was so elusive for probability distributions. The reason is not that utility functions are better expressed as graphical models than are probability distributions. Rather, the language of CA-independencies is substantially more limited than that of conditional independencies in the probabilistic case: For probability distributions, we can evaluate any statement of the form "$\boldsymbol{X}$ is independent of $\boldsymbol{Y}$ given $\boldsymbol{Z}$," whereas for utility functions, the corresponding (CA-independence) statement is well defined only when $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ form a disjoint partition of $\boldsymbol{V}$. In other words, although any CA-independence statement that holds in the utility function can be read from the graph, the set of such statements is significantly more restricted. In fact, a similar weak completeness statement can also be shown for probability distributions.

22.4.2.3 Generalized Additive Independence

Because of its limited expressivity, the notion of conditional additive independence allows us only to make fairly coarse assertions regarding independence — independence of two subsets of variables given all of the rest. As a consequence, the associated factorization is also quite coarse. In particular, we can only use CA-independencies to derive a factorization of the utility function over the maximal cliques in the Markov network. As was the case in probabilistic models (see section 4.4.1.1), this type of factorization can obscure the finer-grained structure in the function, and its parameterization may be exponentially larger.

In this section, we present the most general additive decomposition: the decomposition of equation (22.8), but with arbitrarily overlapping subsets. This type of decomposition is the utility analogue to a Gibbs distribution (definition 4.3), with the factors here combining additively rather than multiplicatively.

Once again, we can provide an independence-based formulation for this decomposition:

Definition 22.12 GA-independence

Let $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ be (not necessarily disjoint) subsets of $\boldsymbol{V}$. We say that $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ are generalized additively independent (GA-independent) *in $\prec$ if, for any lotteries π_1, π_2 that have the same marginals on all $\boldsymbol{Z}_i$, we have that π_1 and π_2 are indifferent under $\prec$.* ■

This definition is identical to that of additive independence (definition 22.8), with the exception that the subsets $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ are not necessarily mutually exclusive nor exhaustive. Thus, this definition allows us to consider cases where our preferences between two distributions depend only on some arbitrary set of marginals. It is also not hard to show that GA-independence subsumes CA-independence (see exercise 22.6).

Satisfyingly, a factorization theorem analogous to theorem 22.2 holds for GA-independence:

Theorem 22.6

Let $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ be (not necessarily disjoint) subsets of $\boldsymbol{V}$, and let $\prec, U$ be a corresponding pair of a preference ordering and a utility function. Then $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_k$ are GA-independent in $\prec$ if and only if U can be written as:

$$U(\boldsymbol{V}) = \sum_{i=1}^{k} U_i(\boldsymbol{Z}_i). \tag{22.10}$$

Thus, the set of possible factorizations associated with GA-independence strictly subsumes the set of factorizations associated with CA-independence. For example, using GA-independence,

we can obtain a factorization $U(X, Y, Z) = U_1(X, Y) + U_2(Y, Z) + U_3(X, Z)$. The Markov network associated with this factorization is a full clique over X, Y, Z, and therefore no CA-independencies hold for this utility function. Overall, GA-independence provides a rich and natural language for encoding complex utility functions (see, for example, box 22.A).

---

prenatal diagnosis

Box 22.A — Case Study: Prenatal Diagnosis. *An important problem involving utilities arises in the domain of* prenatal diagnosis*, where the goal is to detect chromosomal abnormalities present in a fetus in the early stages of the pregnancy. There are several tests available to diagnose these diseases. These tests have different rates of false negatives and false positives, costs, and health risks. The task is to decide which tests to conduct on a particular patient. This task is quite difficult. The patient's risk for having a child with a serious disease depends on the mother's age, child's sex and race, and the family history. Some tests are not very accurate; others carry a significant risk of inducing miscarriages. Both a miscarriage (spontaneous abortion or SAB) and an elective termination of the pregnancy (induced abortion or IAB) can affect the woman's chances of conceiving again.*

Box 23.A describes a decision-making system called PANDA (Norman et al. 1998) for assisting the parents in deciding on a course of action for prenatal testing. The PANDA system requires that we have a utility model for the different outcomes that can arise as part of this process. Note that, unlike for probabilistic models, we cannot simply construct a single utility model that applies to all patients. Different patients will typically have very different preferences regarding these outcomes, and certainly regarding lotteries over them. Interestingly, the standard protocol (and the one followed by many health insurance companies), which recommends prenatal diagnosis (under normal circumstances) only for women over the age of thirty-five, was selected so that the risk (probability) of miscarriage is equal to that of having a Down syndrome baby. Thus, this recommendation essentially assumes not only that all women have the same utility function, but also that they have equal utility for these two events.

The outcomes in this domain have many attributes, such as the inconvenience and expense of fairly invasive testing, the disease status of the fetus, the possibility of test-induced miscarriage, knowledge of the status of the fetus, and future successful pregnancy. Specifically, the utility could be viewed as a function of five attributes: pregnancy loss L, with domain {no loss, miscarriage, elective termination}; Down status D of the fetus, with domain: {normal, Down}; mother's knowledge K, with domain {none, accurate, inaccurate}; future pregnancy F, with domain {yes, no}; type of test T with domain {none, CVS, amnio}. An outcome is an assignment of values to all the attributes. For example, $\langle$no loss, normal, none, yes, none$\rangle$ is one possible outcomes. It represents the situation in which the fetus is not affected by Down syndrome, the patient decides not to take any tests (as a consequence, she is unaware of the Down status of the fetus until the end of the pregnancy), the pregnancy results in normal birth, and there is a future pregnancy. Another outcome, $\langle$miscarriage, normal, accurate, no, CVS$\rangle$ represents a situation where the patient decides to undergo the CVS test. The test result correctly asserts that the fetus is not affected by Down syndrome. However, a miscarriage occurs as a side effect of the procedure, and there is no future pregnancy. Our decision-making situation involves comparing lotteries involving complex (and emotionally difficult) outcomes such as these.

In this domain, we have three ternary attributes and two binary ones, so the total number of outcomes is 108*. Even if we remove outcomes that have probability zero (or are very unlikely), a*

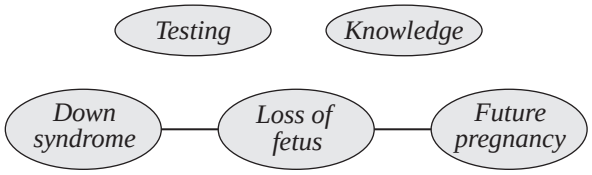


Figure 22.A.1 — Typical utility function decomposition for prenatal diagnosis

large number of outcomes remain. In order to perform utility elicitation, we must assign a numerical utility to each one. A standard utility elicitation process such as standard gamble involves a fairly large number of comparisons for each outcome. Such a process is clearly infeasible in this case.

However, in this domain, many of the utility functions elicited from patients decompose additively in natural ways. For example, as shown by Chajewska and Koller (2000), many patients have utility functions where invasive testing (T) and knowledge (K) are additively independent of other attributes; pregnancy loss (L) is correlated both with Down syndrome (D) and with future pregnancy (F), but there is no direct interaction between Down syndrome and future pregnancy. Thus, for example, one common decomposition is $U_1(T) + U_2(K) + U_3(D, L) + U_4(L, F)$, as encoded in the Markov network of figure 22.A.1.

Box 22.B — Case Study: Utility Elicitation in Medical Diagnosis. *In box 3.D we described the Pathfinder system, designed to assist a pathologist in diagnosing lymph-node diseases. The Pathfinder system was purely probabilistic — it produced only a probability distribution over possible diagnoses. However, the performance evaluation of the Pathfinder system accounted for the implications of a correct or incorrect diagnosis on the patient's utility. For example, if the patient has a viral infection and is diagnosed as having a bacterial infection, the consequences are not so severe: the patient may take antibiotics unnecessarily for a few days or weeks. On the other hand, if the patient has Hodgkin's disease and is incorrectly diagnosed as having a viral infection, the consequences — such as delaying chemotherapy — may be lethal. Thus, to evaluate more accurately the implications of Pathfinder's performance, a utility model was constructed that assigned, for every pair of diseases d, d', a utility value $u_{d,d'}$, which denotes the patient's utility for having the disease d and being diagnosed with disease d'.*

We might be tempted to evaluate the system's performance on a particular case by computing $u_{d^*,d} - u_{d^*,d^*}$, where d^* is the true diagnosis, and d is the most likely diagnosis produced by the system. However, this metric ignores the important distinction between the quality of the decision *and the quality of the* outcome*: A bad decision is one that is not optimal relative to the agent's state of knowledge, whereas a bad outcome can arise simply because the agent is unlucky. In this case, the set of observations may suggest (even to the most knowledgeable expert) that one disease is the most likely, even when another is actually the case. Thus, a better metric is the* inferential loss *— the difference in the expected utility between the* gold standard *distribution produced by an expert and the distribution produced by the system, given exactly the same set of observations.*

inferential loss

Estimating the utility values $u_{d,d'}$ is a nontrivial task. One complication arises from the fact that this situation involves outcomes whose consequences are fairly mild, and others that involve a significant risk of morbidity or mortality. Putting these on a single scale is quite challenging. The approach taken in Pathfinder is to convert all utilities to the micromort scale — a one-in-a-million chance of death. For severe outcomes (such as Hodgkins disease), one can ask the patient what probability of immediate, painless death he would be willing to accept in order to avoid both the disease d and the (possibly incorrect) diagnosis d'. For mild outcomes, where the micromort equivalent may be too low to evaluate reliably, utilities were elicited in terms of monetary equivalents — for example, how much the patient would be willing to pay to avoid taking antibiotics for two weeks. At this end of the spectrum, the "conversion" between micromorts and dollars is fairly linear (see section 22.3.2), and so the resulting dollar amounts can be converted into micromorts, putting these utilities on the same scale as that of severe outcomes.

The number of distinct utilities that need to be elicited even in this simple setting is impractically large: with sixty diseases, the number of utilities is $60^2 = 3,600$. Even aggregating diseases that have similar treatments and prognoses, the number of utilities is $36^2 = 1,296$. However, utility independence can be used to decompose the outcomes into independent factors, such as the disutility of a disease d when correctly treated, the disutility of delaying the appropriate treatment, and the disutility of undergoing an unnecessary treatment. This decomposition reduced the number of assessments by 80 percent, allowing the entire process of utility assessment to be performed in approximately sixty hours.

The Pathfinder IV system (based on a full Bayesian network) resulted in a mean inferential loss of 16 micromorts, as compared to 340 micromorts for the Pathfinder III system (based on a naive Bayes model). At a rate of $20/micromort (the rate elicited in the 1980s), the improvement in the expected utility of Pathfinder IV over Pathfinder III is equivalent to around $6,000 per case.

22.5 Summary

In this chapter, we discussed the use of probabilistic models within the context of a decision task. The key new element one needs to introduce in this context is some representation of the preferences of the agent (the decision maker). Under certain assumptions about these preferences, one can show that the agent, whether implicitly or explicitly, must be following the principle of maximum expected utility, relative to some utility function. It is important to note that the assumptions required for this result are controversial, and that they do not necessarily hold for human decision makers. Indeed, there are many examples where human decision makers do not obey the principle of maximum expected utility. Much work has been devoted to developing other precepts of rational decision making that better match human decision making. Most of this study has taken place in fields such as economics, psychology, or philosophy. Much work remains to be done on evaluating the usefulness of these ideas in the context of automated decision-making systems and on developing computational methods that allow them to be used in complex scenarios such as the ones we discuss in this book.

In general, an agent's utility function can involve multiple attributes of the state of the world. In principle, the complexity of the utility function representation grows exponentially with the number of attributes on which the utility function depends. Several representations have been

developed that assume some structure in the utility function and exploit it for reducing the number of parameters required to represent the utility function.

Perhaps the biggest challenge in this setting is that of acquiring an agent's utility functions. Unlike the case of probability distributions, where an expert can generally provide a single model that applies to an entire population, an agent's utility function is often highly personal and even idiosyncratic. Moreover, the introspection required for a user to understand her preferences and quantify them is a time-consuming and even distressing process. Thus, there is significant value in developing methods that speed up the process of utility elicitation.

A key step in that direction is in learning better models of utility functions. Here, we can attempt to learn a model for the structure of the utility function (for example, its decomposition), or for the parameters that characterize it. We can also attempt to learn richer models that capture the dependence of the utility function on the user's background and perhaps even its evolution over time. Another useful direction is to try to learn aspects of the agent's utility function by observing previous decisions that he made.

These models can be viewed as narrowing down the space of possibilities for the agent's utility function, allowing it to be elicited using fewer questions. One can also try to develop algorithms that intelligently reduce the number of utility elicitation questions that one needs to ask in order to make good decisions. It may be hoped that a combination of these techniques will make utility-based decision making a usable component in our toolbox.

22.6 Relevant Literature

Pascal's wager

The principle of maximum expected utility dates back at least to the seventeenth century, where it played a role in the famous *Pascal's wager* (Arnauld and Nicole 1662), a decision-theoretic analysis concerning the existence of God. Bernoulli (1738), analyzing the *St. Petersburg paradox*, made the distinction between monetary rewards and utilities. Bentham (1789) first proposed the idea that all outcomes should be reduced to numerical utilities.

Ramsey (1931) was the first to provide a formal derivation of numerical utilities from preferences. The axiomatic derivation described in this chapter is due to von Neumann and Morgenstern (1944). A Bayesian approach to decision theory was developed by Ramsey (1931); de Finetti (1937); Good (1950); Savage (1954). Ramsey and Savage both defined axioms that provide a simultaneous justification for both probabilities and utilities, in contrast to the axioms of von Neumann and Morgenstern, that take probabilities as given. These axioms motivate the Bayesian approach to probabilities in a decision-theoretic setting. The book by Kreps (1988) provides a good review of the topic.

The principle of maximum expected utility has also been the topic of significant criticism, both on normative and descriptive grounds. For example, Kahneman and Tversky, in a long series of papers, demonstrate that human behavior is often inconsistent with the principles of rational decision making under uncertainty for any utility function (see, for example, Kahneman, Slovic, and Tversky 1982). Among other things, Tversky and Kahneman show that people commonly use heuristics in their probability assessments that simplify the problem but often lead to serious errors. Specifically, they often pay disproportionate attention to low-probability events and treat high-probability events as though they were less likely than they actually are.

Motivated both by normative limitations in the MEU principle and by apparent inconsistencies

minimax risk

between the MEU principle and human behavior, several researchers have proposed alternative criteria for optimal decision making. For example, Savage (1951) proposed the *minimax risk* criterion, which asserts that we should associate with each outcome not only some utility value but also a regret value. This approach was later refined by Loomes and Sugden (1982) and Bell (1982), who show how regret theory can be used to explain such apparently irrational behaviors as gambling on negative-expected-value lotteries or buying costly insurance.

A discussion of utility functions exhibited by human decision makers can be found in von Winterfeldt and Edwards (1986). Howard (1977) provides an extensive discussion of attitudes toward risk and defines the notions of risk-averse, risk-seeking, and risk-neutral behaviors. Howard (1989) proposes the notion of micromorts for eliciting utilities regarding human life. The Pathfinder system is one of the first automated medical diagnosis systems to use a carefully constructed utility function; a description of the system, and of the process used to construct the utility function, can be found in Heckerman (1990) and Heckerman, Horvitz, and Nathwani (1992).

The basic framework of multiattribute utility theory is presented in detail in the seminal book of Keeney and Raiffa (1976). These ideas were introduced into the AI literature by Wellman (1985). The notion of generalized additive independence (GAI), under the name *interdependent value additivity*, was proposed by Fishburn (1967, 1970), who also provided the conditions under which a GAI model provides an accurate representation of a utility function. The idea of using a graphical model to represent utility decomposition properties was introduced by Wellman and Doyle (1992). The rigorous development was performed by Bacchus and Grove (1995), who also proposed the idea of GAI-networks. These ideas were subsequently extended in various ways by La Mura and Shoham (1999) and Boutilier, Bacchus, and Brafman (2001).

Much work has been done on the problem of utility elicitation. The standard gamble method was first proposed by von Neumann and Morgenstern (1947), based directly on their axioms for utility theory. Time trade-off was proposed by Torrance, Thomas, and Sackett (1972). The visual analog scale dates back at least to the 1970s (Patrick et al. 1973); see Drummond et al. (1997) for a detailed presentation. Chajewska (2002) reviews these different methods and some of the documented difficulties with them. She also provides a fairly recent overview of different approaches to utility elicitation.

There has been some work on eliciting the structure of decomposed utility functions from users (Keeney and Raiffa 1976; Anderson 1974, 1976). However, most often a simple (for example, fully additive) structure is selected, and the parameters are estimated using least-squares regression from elicited utilities of full outcomes. Chajewska and Koller (2000) show how the problem of inferring the decomposition of a utility function can be viewed as a Bayesian model selection problem and solved using techniques along the lines of chapter 18. Their work is based on the idea of explicitly representing an explicit probabilistic model over utility parameters, as proposed by Jimison et al. (1992). The prenatal diagnosis example is taken from Chajewska and Koller (2000), based on data from Kuppermann et al. (1997).

Several authors (for example, Heckerman and Jimison 1989; Poh and Horvitz 2003; Ha and Haddawy 1997, 1999; Chajewska et al. 2000; Boutilier 2002; Braziunas and Boutilier 2005) have proposed that model refinement, including refinement of utility assessments, should be viewed in terms of optimizing expected value of information (see section 23.7). In general, it can be shown that a full utility function need not be elicited to make optimal decisions, and that close-to-optimal decisions can be made after a small number of utility elicitation queries.

22.7 Exercises

Exercise 22.1

Complete the proof of theorem 22.1. In particular, let $U(s) := p$, as defined in equation (22.7), be our utility assignment for outcomes. Show that, if we use the MEU principle for selecting between two lotteries, the resulting preference over lotteries is equivalent to $\prec$. (Hint: Do not forget to address the case of compound lotteries.)

Exercise 22.2

Prove proposition 22.4.

Exercise 22.3

Prove lemma 22.1.

Exercise 22.4

Complete the proof of theorem 22.4.

Exercise 22.5⋆

Prove theorem 22.5. (Hint: Use exercise 4.11.)

Exercise 22.6⋆

Prove the following result without using the factorization properties of U. Let $\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}$ be a disjoint partition of $\boldsymbol{V}$. Then $\boldsymbol{X}, \boldsymbol{Z}$ and $\boldsymbol{Y}, \boldsymbol{Z}$ are GA-independent in $\prec$ if and only if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are CA-independent given $\boldsymbol{Z}$ in $\prec$.

This result shows that CA-independence and GA-independence are equivalent *over the scope of independence assertions to which CA-independence applies* (those involving disjoint partitions of $\boldsymbol{V}$).

Exercise 22.7

Consider the problem of computing the optimal action for an agent whose utility function we are uncertain about. In particular, assume that, rather than a known utility function over outcomes $\mathcal{O}$, we have a probability density function $P(U)$, which assigns a density for each possible utility function $U : \mathcal{O} \mapsto \mathbb{R}$.

a. What is the expected utility for a given action a, taking an expectation both over the outcomes of π_a, and over the possible utility functions that the agent might have?
b. Use your answer to provide an efficient computation of the optimal action for the agent.

Exercise 22.8⋆

As we discussed, different people have different utility functions. Consider the problem of learning a probability distribution over the utility functions found in a population. Assume that we have a set of samples $U[1], \ldots, U[M]$ of users from a population, where for each user m we have elicited a utility function $U[m] : Val(\boldsymbol{V}) \mapsto \mathbb{R}$.

a. Assume that we want to model our utility function as in equation (22.10). We want to use the same factorization for all users, but where different users have different subutility functions; that is, $U_i[m]$ and $U_i[m']$ are not the same. Moreover, the elicited values $U(\boldsymbol{v})[m]$ are noisy, so that they may not decompose exactly as in equation (22.10). We can model the actual elicited values for a given user as the sum in equation (22.10) plus Gaussian noise.

   Formulate the distribution over the utility functions in the population as a linear Gaussian graphical model. Using the techniques we learned earlier in this book, provide a learning algorithm for the parameters in this model.
b. Now, assume that we allow different users in the population to have one of several different factorizations of their utility functions. Show how you can extend your graphical model and learning algorithm accordingly. Show how your model allows you to infer which factorization a user is likely to have.

23 *Structured Decision Problems*

In the previous chapter, we described the basic principle of decision making under uncertainty — maximizing expected utility. However, our definition for a decision-making problem was completely abstract; it defined a decision problem in terms of a set of abstract states and a set of abstract actions. Yet, our overarching theme in this book has been the observation that the world is structured, and that we can obtain both representational and computational efficiency by exploiting this structure. In this chapter, we discuss structured representations for decision-making problems and algorithms that exploit this structure when addressing the computational task of finding the decision that maximizes the expected utility.

We begin by describing *decision trees* — a simple yet intuitive representation that describes a decision-making situation in terms of the scenarios that the decision maker might encounter. This representation, unfortunately, scales up only to fairly small decision tasks; still, it provides a useful basis for much of the later development. We describe *influence diagrams*, which extend Bayesian networks by introducing decisions and utilities. We then discuss algorithmic techniques for solving and simplifying influence diagrams. Finally, we discuss the concept of *value of information*, which is very naturally encoded within the influence diagram framework.

23.1 Decision Trees

23.1.1 Representation

A decision tree is a representation of the different scenarios that might be encountered by the decision maker in the context of a particular decision problem. A decision tree has two types of internal nodes (denoted t-nodes to distinguish them from nodes in a graphical model) — one set encoding decision points of the agent, and the other set encoding decisions of nature. The outgoing edges at an agent's t-node correspond to different decisions that the agent might make. The outgoing edges at one of nature's t-nodes correspond to random choices that are made by nature. The leaves of the tree are associated with outcomes, and they are annotated with the agent's utility for that outcome.

Definition 23.1 decision tree t-node

A decision tree $\mathcal{T}$ *is a rooted tree with a set of internal* t-nodes $\mathcal{V}$ *and leaves* $\mathcal{V}_L$. *The set* $\mathcal{V}$ *is partitioned into two disjoint sets —* agent t-nodes $\mathcal{V}_A$ *and* nature t-nodes $\mathcal{V}_N$. *Each t-node has some set of choices* $\mathcal{C}[v]$, *associated with its outgoing edges. We let* $succ(v, c)$ *denote the child of* v *reached via the edge labeled with* c. *Each of nature's t-nodes* v *is associated with a probability*

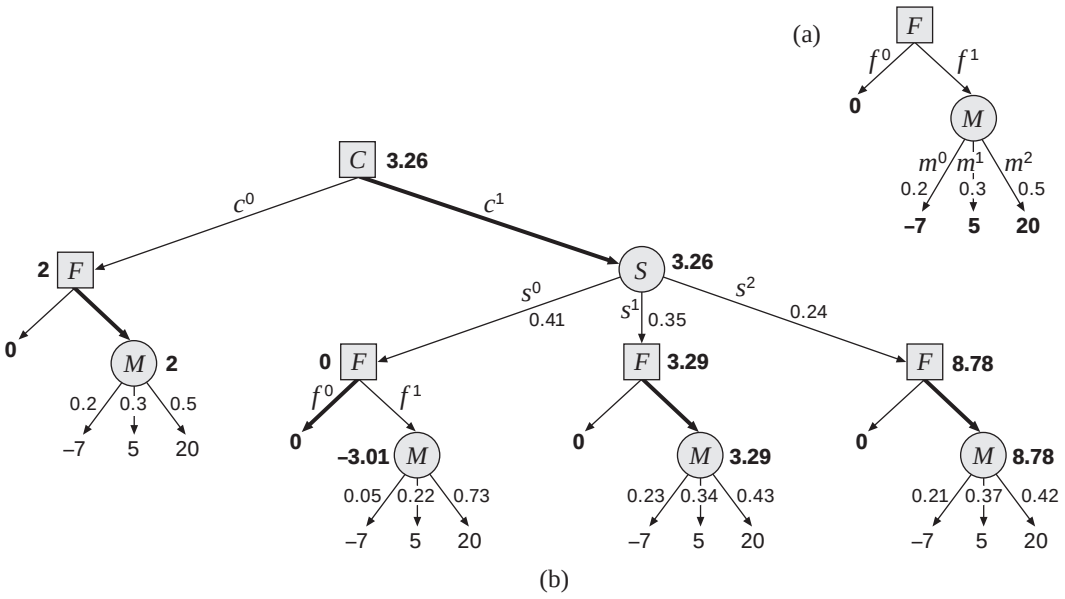


Figure 23.1 Decision trees for the Entrepreneur **example.** (a) one-stage scenario; (b) two-stage scenario, with the solution (optimal strategy) denoted using thicker edges.

distribution P_v over $C[v]$. Each leaf $v \in \mathcal{V}_L$ in the tree is annotated with a numerical value $U(v)$ corresponding to the agent's utility for reaching that leaf. ■

Most simply, in our basic decision-making scenario of definition 22.2, a lottery ℓ induces a two-layer tree. The root is an agent t-node v, and it has an outgoing edge for each possible action $a \in \mathcal{A}$, leading to some child $succ(v, a)$. Each node $succ(v, a)$ is a nature t-node; its children are leaves in the tree, with one leaf for each outcome in $\mathcal{O}$ for which $\ell_a(o) > 0$; the corresponding edge is labeled with the probability $\ell_a(o)$. The leaf associated with some outcome o is annotated with $U(o)$. Most simply, in our basic Entrepreneur scenario of example 22.3, the corresponding decision tree would be as shown in figure 23.1a. Note that if the agent decides not to found the company, there is no dependence of the outcome on the market demand, and the agent simply gets a utility of 0.

☞ **The decision-tree representation allows us to encode decision scenarios in a way that reveals much more of their internal structure than the abstract setting of outcomes and utilities. In particular, it allows us to encode explicitly sequential decision settings, where the agent makes several decisions; it also allows us to encode information that is available to the agent at the time a decision must be made.**

Example 23.1

Consider an extension of our basic Entrepreneur *example where the entrepreneur has the opportunity to conduct a survey on the demand for widgets before deciding whether to found the company. Thus, the agent now has a sequence of two decisions: the first is whether to conduct the survey, and the second is whether to found the company. If the agent conducts the survey, he obtains informa-*

tion about its outcome, which can be one of three values: a negative reaction, s^0, indicating almost no hope of widget sales; a neutral reaction, s^1, indicating some hope of sales; and an enthusiastic reaction, s^2, indicating a lot of potential demand. The probability distribution over the market demand is different for the different outcomes of the survey. If the agent conducts the survey, his decision on whether to found the company can depend on the outcome.

The decision tree is shown in figure 23.1b. At the root, the agent decides whether to conduct the survey (c^1) or not (c^0). If he does not conduct the survey, the next t-node is another decision by the agent, where he decides whether to found the company (f^1) or not (f^0). If the agent decides to found the company, nature decides on the market demand for widgets, which determines the final outcome. The situation if the agent decides to conduct the survey is more complex. Nature then probabilistically chooses the value of the survey. For each choice, the agent has a t-node where he gets to decide whether he founds the company or not. If he does, nature gets to decide on the distribution of market demand for widgets, which is different for different outcomes of the survey. ■

strategy

We can encode the agent's overall behavior in a decision problem encoded as a decision tree as a *strategy*. There are several possible definitions of a strategy. One that is simple and suitable for our purposes is a mapping from agent t-nodes to possible choices at that t-node.

Definition 23.2 decision-tree strategy

A decision-tree strategy σ *specifies, for each* $v \in \mathcal{V}_A$, *one of the choices labeling its outgoing edges.* ■

For example, in the decision tree of figure 23.1b, a strategy has to designate an action for the agent t-node, labeled C, and the four agent t-nodes, labeled F. One possible strategy is illustrated by the thick lines in the figures.

Decision trees provide a structured representation for complex decision problems, potentially involving multiple decisions, taken in sequence, and interleaved with choices of nature. However, they are still instances of the abstract framework defined in definition 22.2. Specifically, the outcomes are the leaves in the tree, each of which is annotated with a utility; the set of agent actions is the set of all strategies; and the probabilistic outcome model is the distribution over leaves induced by nature's random choices given a strategy (action) for the agent.

23.1.2 Backward Induction Algorithm

As in the abstract decision-making setting, our goal is to select the strategy that maximizes the agent's expected utility. This computational task, for the decision-tree representation, can be solved using a straightforward tree-traversal algorithm. This approach is an instance of an approach called *backward induction* in the game-theoretic and economic literature, and the *Expectimax* algorithm in the artificial intelligence literature.

backward induction

Expectimax

The algorithm proceeds from the leaves of the tree upward, computing the maximum expected utility MEU_v achievable by the agent at each t-node v in the tree — his expected utility if he plays the optimal strategy from that point on. At a leaf v, MEU_v is simply the utility $U(v)$ associated with that leaf's outcome. Now, consider an internal t-node v for whose children we have already computed the MEU. If v belongs to nature, the expected utility accruing to the agent if v is reached is simply the weighted average of the expected utilities at each of v's children, where the weighted average is taken relative to the distribution defined by nature over v's children. If v belongs to the agent, the agent has the ability to select the action at v.

Algorithm 23.1 Finding the MEU strategy in a decision tree

```
    Procedure MEU-for-Decision-Trees (
      T    // Decision tree
    )
1     L ← Leaves(T)
2     for each node v ∈ L
3       Remove v from L
4       Add v's parents to L
5       if v is a leaf then
6         MEU_v ← U(v)
7       else if v belongs to nature then
8         MEU_v ← Σ_{c∈C[v]} P_v(c) MEU_{succ(v,c)}
9       else    // v belongs to the Agent
10        σ(v) ← arg max_{c∈C[v]} MEU_{succ(v,c)}
11        MEU_v ← MEU_{succ(v,succ(v,))}
12    return (σ)
```

The optimal action for the agent is the one leading to the child whose MEU is largest, and the MEU accruing to the agent is the MEU associated with that child. The algorithm is shown in algorithm 23.1.

23.2 Influence Diagrams

The decision-tree representation is a significant improvement over representing the problem as a set of abstract outcomes; however, much of the structure of the problem is still not made explicit. For example, in our simple Entrepreneur scenario, the agent's utility if he founds the company depends only on the market demand M, and not on the results of the survey S. In the decision tree, however, the utility values appear in four separate subtrees: one for each value of the S variable, and one for the subtree where the survey is not performed. An examination of the utility values shows that they are, indeed, identical, but this is not apparent from the structure of the tree.

The tree also loses a subtler structure, which cannot be easily discerned by an examination of the parameters. The tree contains four nodes that encode a probability distribution over the values of the market demand M. These four distributions are different. We can presume that neither the survey nor the agent's decision has an effect on the market demand itself. The reason for the change in the distribution presumably arises from the effect of conditioning the distribution on different observations (or no observation) on the survey variable S. In other words, these distributions represent $P(M \mid s^0)$, $P(M \mid s^1)$, $P(M \mid s^2)$, and $P(M)$ (in the branch where the survey was not performed). These interactions between these different parameters are obscured by the decision-tree representation.

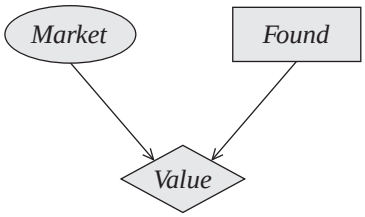


Figure 23.2 Influence diagram $\mathcal{I}_F$ for the basic Entrepreneur **example**

23.2.1 Basic Representation

influence diagram

An alternative representation is the *influence diagram* (sometimes also called a *decision network*), a natural extension of the Bayesian network framework. It encodes the decision scenario via a set of variables, each of which takes on values in some space. Some of the variables are random variables, as we have seen so far, and their values are selected by nature using some probabilistic model. Others are under the control of the agent, and their value reflects a choice made by him. Finally, we also have numerically valued variables encoding the agent's utility.

This type of model can be encoded graphically, using a directed acyclic graph containing three types of nodes — corresponding to *chance variables*, *decision variables*, and *utility variables*. These different node types are represented as ovals, rectangles, and diamonds, respectively. An influence diagram $\mathcal{I}$ is a directed acyclic graph over these nodes, such that the utility nodes have no children.

Example 23.2

The influence diagram $\mathcal{I}_F$ for our entrepreneur *example is shown in figure 23.2. The utility variable V_E encodes the utility of the entrepreneur's earnings, which are a deterministic function of the utility variable's parents. This function specifies the agent's real-valued utility for each combination of the parent nodes; in this case, the utility is a function from $Val(M) \times Val(F)$ to $\mathbb{R}$. We can represent this function as a table:*

	m^0	m^1	m^2
f^1	-7	5	20
f^0	0	0	0,

where f^1 represents the decision to found the company and f^0 the decision not to do so. The CPD for the M node is:

m^0	m^1	m^2
0.5	0.3	0.2.

■

chance variable

decision variable

More formally, in an influence diagram, the world in which the agent acts is represented by the set $\mathcal{X}$ of *chance variables*, and by a set $\mathcal{D}$ of *decision variables*. Chance variables are those whose values are chosen by nature. The decision variables are variables whose values the agent gets to choose. Each variable $V \in \mathcal{X} \cup \mathcal{D}$ has a finite domain $Val(V)$ of possible values. We can place this representation within the context of the abstract framework of definition 22.2: The possible actions $\mathcal{A}$ are all of the possible assignments $Val(\mathcal{D})$; the possible outcomes are all of

the joint assignments in $Val(\mathcal{X} \cup \mathcal{D})$. Thus, this framework provides a factored representation of both the action and the outcome space.

We can also decompose the agent's utility function. A standard decomposition (see discussion in section 22.4) is as a linear sum of terms, each of which represents a certain component of the agent's utility. More precisely, we have a set of *utility variables* $\mathcal{U}$, which take on real numbers as values. The agent's final utility is the sum of the value of V for all $V \in \mathcal{U}$.

utility variable

Let $\mathcal{Z}$ be the set of all variables in the network — chance, decision, and utility variables. We expand the notion of *outcome* to encompass a full assignment to $\mathcal{Z}$, which we denote as ζ.

outcome

The parents of a chance variable X represent, as usual, the direct influences on the choice of X's value. Note that the parents of X can be both other chance variables as well as decision variables, but they cannot be utility variables, since we assumed that utility nodes have no children. Each chance node X is associated with a CPD, which represents $P(X \mid \mathrm{Pa}_X)$.

The parents of a utility variable V represent the set of variables on which the utility V depends. The value of a utility variable V is a deterministic function of the values of Pa_V; we use $V(\boldsymbol{w})$ to denote the value that node V takes when $\mathrm{Pa}_V = \boldsymbol{w}$. Note that, as for any deterministic function, we can also view V as defining a CPD, where for each parent assignment, some value gets probability 1. When convenient, we will abuse notation and interpret a utility node as defining a factor.

Summarizing, we have the following definition:

Definition 23.3 influence diagram

An influence diagram *$\mathcal{I}$ over $\mathcal{Z}$ is a directed acyclic graph whose nodes correspond to $\mathcal{Z}$, and where nodes corresponding to utility variables have no children. Each chance variable $X \in \mathcal{X}$ is associated with a CPD $P(X \mid \mathrm{Pa}_X)$. Each utility variable $V \in \mathcal{U}$ is associated with a deterministic function $V(\mathrm{Pa}_V)$.* ■

23.2.2 Decision Rules

So far, we have not discussed the semantics of the decision node. For a decision variable $D \in \mathcal{D}$, Pa_D is the set of variables whose values the agent knows when he chooses a value for D. The edges incoming into a decision variable are often called *information edges*.

information edge

Example 23.3

Let us return to the setting of example 23.1. Here, we have the chance variable M that represents the market demand, and the chance variable S that represents the results of the survey. The variable S has the values s^0, s^1, s^2, and an additional value $s^\perp$, denoting that the survey was not taken. This additional value is needed in this case, because we allow the agent's decision to depend on the value of S, and therefore we need to allow some value for this variable when the survey is not taken. The variable S has two parents, C and M. We have that $P(s^\perp \mid c^0, m) = 1$, for any value of m. In the case c^1, the probabilities over values of S are:

	s^0	s^1	s^2
m^0	0.6	0.3	0.1
m^1	0.3	0.4	0.3
m^2	0.1	0.4	0.5

The entrepreneur knows the result of the survey before making his decision whether to found the company. Thus, there is an edge between S and his decision F. We also assume that conducting

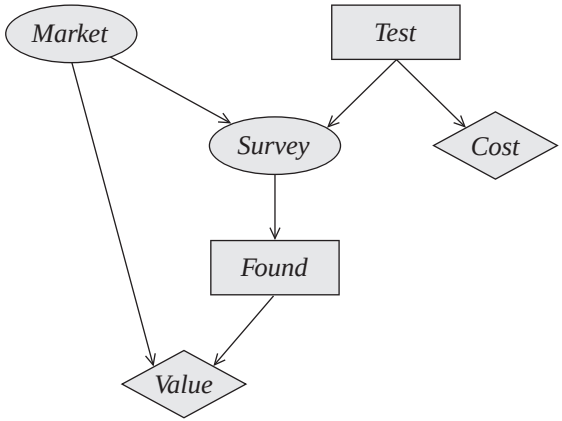


Figure 23.3 Influence diagram $\mathcal{I}_{F,C}$ for Entrepreneur **example with market survey**

the survey has some cost, so that we have an additional utility node V_S, with the parent C; V_S takes on the value -1 if $C = c^1$ and 0 otherwise. The resulting influence diagram $\mathcal{I}_{F,C}$ is shown in figure 23.3. ■

The influence diagram representation captures the causal structure of the problem and its parameterization in a much more natural way than the decision tree. It is clear in the influence diagram that S depends on M, that M is parameterized via a simple (unconditional) prior distribution, and so on.

The choice that the agent makes for a decision variable D can be contingent only on the values of its parents. More precisely, in any trajectory through the decision scenario, the agent will encounter D in some particular *information states*, where each information state is an assignment of values to Pa_D. An agent's strategy for D must tell the agent how to act at D, at each of these information states.

information state

Example 23.4

In example 23.3, for instance, the agent's strategy must tell him whether to found the company or not in each possible scenario he may encounter; the agent's information state at this decision is defined by the possible values of the decision variable C and the survey variable S. The agent must therefore decide whether to found the company in four different information states: if he chose not to conduct the survey, and in each of the three different possible outcomes of the survey. ■

A *decision rule* tells the agent how to act in each possible information state. Thus, the agent is choosing a local conditional model for the decision variable D. In effect, the agent has the ability to choose a CPD for D.

Definition 23.4

decision rule

deterministic decision rule

complete strategy

A decision rule *δ_D for a decision variable D is a conditional probability $P(D \mid \mathrm{Pa}_D)$ — a function that maps each instantiation pa_D of Pa_D to a probability distribution δ_D over $Val(D)$. A decision rule is* deterministic *if each probability distribution $\delta_D(D \mid \mathrm{pa}_D)$ assigns nonzero probability to exactly one value of D. A complete assignment σ of decision rules to every decision $D \in \mathcal{D}$ is called a* complete strategy*; we use σ_D to denote the decision rule at D.* ■

Example 23.5 *A decision rule for C is simply a distribution over its two values. A decision rule for F must define, for every value of C, and for every value $s^0, s^1, s^2, s^\perp$ of S, a probability distribution over values of F. Note, however, that there is a deterministic relationship between C and S, so that many of the combinations are inconsistent (for example, c^1 and $s^\perp$, or c^0 and s^1). For example, in the case c^1, s^1, one possible decision rule for the agent is f^0 with probability 0.7 and f^1 with probability 0.3.* ■

As we will see, in the case of single-agent decision making, one can always choose an optimal deterministic strategy for the agent. However, it is useful to view a strategy as an assignment of CPDs to the decision variables. Indeed, in this case, the parents of a decision node have the same semantics as the parents of a chance node: the agent's strategy can depend only on the values of the parent variables. Moreover, randomized decision rules will turn out to be a useful concept in some of our constructions that follow. In the common case of deterministic decision rules, which pick a single action $d \in Val(D)$ for each assignment $\boldsymbol{w} \in Val(\mathrm{Pa}_D)$, we sometimes abuse notation and use δ_D to refer to the decision-rule function, in which case $\delta_D(\boldsymbol{w})$ denotes the single action d that has probability 1 given the parent assignment $\boldsymbol{w}$.

23.2.3 Time and Recall

intervention

Unlike a Bayesian network, an influence diagram has an implicit causal semantics. One assumes that the agent can *intervene* at a decision variable D by selecting its value. This intervention will affect the values of variables downstream from D. By choosing a decision rule, the agent determines how he will intervene in the system in different situations.

The acyclicity assumption for influence diagrams, combined with the use of information edges, ensures that an agent cannot observe a variable that his action affects. Thus, acyclicity implies that the network respects some basic causal constraints. In the case of multiple decisions, we often want to impose additional constraints on the network structure. In many cases, one assumes that the decisions in the network are all made by a single agent in some sequence over time; in this case, we have a total ordering $\prec$ on $\mathcal{D}$. An additional assumption that is often made in this case is that the agent does not forget his previous decisions or information it once had. This assumption is typically called the *perfect recall* assumption (or sometime the *no forgetting* assumption), formally defined as follows:

Definition 23.5 *An influence diagram $\mathcal{I}$ is said to have a* temporal ordering *if there is some total ordering $\prec$ over $\mathcal{D}$, which is consistent with partial ordering imposed by the edges in $\mathcal{I}$. The influence diagram $\mathcal{I}$ satisfies the* perfect recall *assumption relative to $\prec$ if, whenever $D_i \prec D_j$, $\mathrm{Pa}_{D_j} \supset (\mathrm{Pa}_{D_i} \cup \{D_i\})$. The edges from $\mathrm{Pa}_{D_i} \cup \{D_i\}$ to D_j are called* recall edges. ■

temporal ordering

perfect recall

recall edge

Intuitively, a recall edge is an edge from a variable X (chance or decision) to a decision variable D whose presence is implied by the perfect recall assumption. In particular, if D' is a decision that precedes D in the temporal ordering, then we have recall edges $D' \to D$ and $X \to D$ for $X \in \mathrm{Pa}_{D'}$. To reduce visual clutter, we often omit recall edges in an influence diagram when the temporal ordering is known. For example, in figure 23.3, we omitted the edge from C to F.

Although the perfect recall assumption appears quite plausible at first glance, there are several arguments against it. First, it is not a suitable model for situations where the "agent" is actually

a compound entity, with individual decisions made by different "subagents." For example, our agent might be a large organization, with different members responsible for various decisions. It is also not suitable for cases where an agent might not have the resources (or the desire) to remember an entire history of all previous actions and observations.

The perfect recall assumption also has significant representational and computational ramifications. The size of the decision rule at a decision node is, in general, exponential in the number of parents of the decision node. In the case of perfect recall, the number of parents grows with every decision, resulting in a very high-dimensional space of possible decision rules for decision variables later in the temporal ordering. This blowup makes computations involving large influence diagrams with perfect recall intractable in many cases. The computational burden of perfect recall leads us to consider also influence diagrams in which the perfect recall assumption does not hold, also known as *limited memory influence diagrams* (or LIMIDs). In these networks, all information edges must be represented explicitly, since perfect recall is no longer universally true. We return to this topic in section 23.6.

limited memory influence diagram

23.2.4 Semantics and Optimality Criterion

A choice of a decision rule δ_D effectively turns D from a decision variable into a chance variable. Let $\sigma_{\boldsymbol{D}}$ be any partial strategy that specifies a decision rule for the decision variables $D \in \boldsymbol{D}$. We can replace each decision variable in $\boldsymbol{D}$ with the CPD defined by its decision rule in σ, resulting in an influence diagram $\mathcal{I}[\sigma]$ whose chance variables are $\mathcal{X} \cup \boldsymbol{D}$ and whose decision variables are $\mathcal{D} - \boldsymbol{D}$. In particular, when σ is a complete strategy, $\mathcal{I}[\sigma]$ is simply a Bayesian network, which we denote by $\mathcal{B}_{\mathcal{I}[\sigma]}$. This Bayesian network defines a probability distribution over possible outcomes ζ.

expected utility

The agent's *expected utility* in this setting is simply:

$$\mathrm{EU}[\mathcal{I}[\sigma]] = \sum_{\zeta} P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(\zeta) U(\zeta) \tag{23.1}$$

where the utility of an outcome is the sum of the individual utility variables in that outcome:

$$U(\zeta) = \sum_{V \in \mathcal{U}} \zeta\langle V \rangle.$$

The linearity of expectation allows us to simplify equation (23.1) by considering each utility variable separately, to obtain:

$$\begin{aligned} \mathrm{EU}[\mathcal{I}[\sigma]] &= \sum_{V \in \mathcal{U}} \boldsymbol{E}_{\mathcal{B}_{\mathcal{I}[\sigma]}}[V] \\ &= \sum_{V \in \mathcal{U}} \sum_{v \in Val(V)} P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(V = v) v. \end{aligned}$$

We often drop the subscript $\mathcal{B}_{\mathcal{I}[\sigma]}$ where it is clear from context.

An alternative useful formulation for this expected utility makes explicit the dependence on the factors parameterizing the network:

$$\mathrm{EU}[\mathcal{I}[\sigma]] = \sum_{\mathcal{X} \cup \mathcal{D}} \left[\left(\prod_{X \in \mathcal{X}} P(X \mid \mathrm{Pa}_X) \right) \left(\prod_{D \in \mathcal{D}} \delta_D \right) \left(\sum_{i \,:\, V_i \in \mathcal{U}} V_i \right) \right]. \tag{23.2}$$

The expression inside the summation is constructed as a product of three components. The first is a product of all of the CPD factors in the network; the second is a product of all of the factors corresponding to the decision rules (also viewed as CPDs); and the third is a factor that captures the agent's utility function as a sum of the subutility functions V_i. As a whole, the expression inside the summation is a single factor whose scope is $\mathcal{X} \cup \mathcal{D}$. The value of the entry in the factor corresponding to an assignment o to $\mathcal{X} \cup \mathcal{D}$ is a product of the probability of this outcome (using the decision rules specified by σ) and the utility of this outcome. The summation over this factor is simply the overall expected utility $\mathrm{EU}[\mathcal{I}[\sigma]]$.

Example 23.6

Returning to example 23.3, our outcomes are complete assignments m, c, s, f, u_s, u_f. The agent's utility in such an outcome is $u_s + u_f$. The agent's expected utility given a strategy σ is

$$P_{\mathcal{B}}(V_S = -1) \cdot -1 + P_{\mathcal{B}}(V_S = 0) \cdot 0 + P_{\mathcal{B}}(V_E = -7) \cdot -7 + P_{\mathcal{B}}(V_E = 5) \cdot 5 + P_{\mathcal{B}}(V_E = 20) \cdot 20 + P_{\mathcal{B}}(V_E = 0) \cdot 0),$$

where $\mathcal{B} = \mathcal{B}_{\mathcal{I}_{F,C}[\sigma]}$. It is straightforward to verify that the strategy that optimizes the expected utility is: $\delta_C = c^1$; $\delta_F(c^1, s^0) = f^0$, $\delta_F(c^1, s^1) = f^1$, $\delta_F(c^1, s^2) = f^1$. Because the event $C = c^0$ has probability 0 in this strategy, any choice of probability distributions for $\delta_F(c^0, S)$ is optimal. By following the definition, we can compute the overall expected utility for this strategy, which is 3.22, so that $\mathrm{MEU}[\mathcal{I}_{F,C}] = 3.22$. ■

According to the basic postulate of statistical decision theory, the agent's goal is to maximize his expected utility for a given decision setting. Thus, he should choose the strategy σ that maximizes $\mathrm{EU}[\mathcal{I}[\sigma]]$.

Definition 23.6

MEU strategy

MEU value

An MEU strategy *σ^* for an influence diagram $\mathcal{I}$ is one that maximizes $\mathrm{EU}[\mathcal{I}[\sigma]]$. The* MEU value *is $\mathrm{EU}[\mathcal{I}[\sigma^*]]$.* ■

In general, there may be more than one MEU strategy for a given influence diagram, but they all have the same expected utility.

This definition lays out the basic computational task associated with influence diagrams: Given an influence diagram $\mathcal{I}$, our goal is to find the MEU strategy $\mathrm{MEU}[\mathcal{I}]$. Recall that a strategy is an assignment of decision rules to all the decision variables in the network; thus, our goal is to find:

$$\arg \max_{\delta_{D_1}, \ldots, \delta_{D_k}} \mathrm{EU}[\mathcal{I}[\delta_{D_1}, \ldots, \delta_{D_k}]]. \tag{23.3}$$

Each decision rule is itself a complex function, assigning an action (or even a distribution over actions) to each information state. This complex optimization task appears quite daunting at first. Here we present two different ways of tackling it.

Box 23.A — Case Study: Decision Making for Prenatal Testing. *As we discussed in box 22.A,* prenatal diagnosis *offers a challenging domain for decision making. It incorporates a sequence of interrelated decisions, each of which has significant effects on variables that determine the patient's preferences. Norman et al. (1998) construct a system called PANDA (which roughly stands*

prenatal diagnosis

for "Prenatal Testing Decision Analysis"). PANDA uses an influence diagram to model the sequential decision process, the relevant random variables, and the patient's utility. The influence diagram contains a sequence of six decisions: four types of diagnostic test (CVS, triple marker screening, ultrasound, and amniocentesis), as well as early and late termination of the pregnancy. The model focuses on five diseases that are serious, relatively common, diagnosable using prenatal testing, and not readily correctable: Down syndrome, neural-tube defects, cystic fibrosis, sickle-cell anemia, and fragile X mental retardation. The probabilistic component of the network (43 variables) includes predisposing factors that affect the probability of these diseases, and it models the errors in the diagnostic ability of the tests (both false positive and false negative). Utilities were elicited for every patient and placed on a scale of 0–100*, where a utility of* 100 *corresponds to the outcome of a healthy baby with perfect knowledge throughout the course of the pregnancy, and a utility of* 0 *corresponds to the outcome of both maternal and fetal death.*

The strategy space in this model is very complex, since any decision (including a decision to take a test) can depend on the outcome of one or more earlier tests, As a consequence, there are about 1.62×10^{272} *different strategies, of which* 3.85×10^{38} *are "reasonable" relative to a set of constraints. This enormous space of options highlights the importance of using automated methods to guide the decision process.*

The system can be applied to different patients who vary both on their predisposing factors and on their utilities. Both the predisposing factors and the utilities give rise to very different strategies. However, a more relevant question is the extent to which the different strategy choices make a difference to the patient's final utility. To provide a reasonable scale for answering this question, the algorithm was applied to select for each patient their best and worst strategy. As an example, for one such patient (a young woman with predisposing factors for sickle-cell anemia), the optimal strategy achieved an expected utility of 98.77 *and the worst strategy an expected utility of* 87.85*, for a difference of* 10.92 *utility points. Other strategies were then evaluated in terms of the percentage of these* 10.92 *points that they provided to the patient. For many patients, most of the reasonable strategies performed fairly well, achieving over 99 percent of the utility gap for that patient. However, for some patients, even reasonable strategies gave very poor results. For example, for the patient with sickle-cell anemia, strategies that were selected as optimal for other patients in the study provided her only 65–70 percent of the utility gap. Notably, the "recommended" strategy for women under the age of thirty-five, which is to perform no tests, performed even worse, achieving only 64.7 percent of the utility gap.*

Overall, this study demonstrates the importance of personalizing medical decision making to the information and the utility for individual patients.

23.3 Backward Induction in Influence Diagrams

We now turn to the problem of selecting the optimal strategy in an influence diagram. Our first approach to addressing this problem is a fairly simple algorithm that mirrors the backward induction algorithm for decision trees described in section 23.1.2. As we will show, this algorithm can be implemented effectively using the techniques of variable elimination of chapter 9. This algorithm applies only to influence diagrams satisfying the perfect recall assumption, a restriction that has significant computational ramifications.

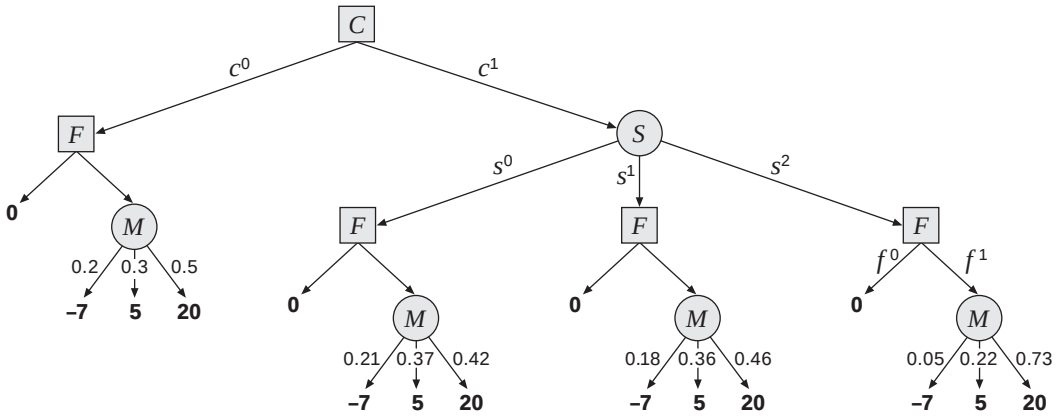


Figure 23.4 Decision tree for the influence diagram $\mathcal{I}_{F,C}$ in the Entrepreneur **example.** For clarity, probability zero events are omitted, and edges are labeled only at representative nodes.

23.3.1 Decision Trees for Influence Diagrams

Our starting point for the backward induction algorithm is to view an influence diagram as defining a set of possible trajectories, defined from the perspective of the agent. A trajectory includes both the observations made by the agent and the decisions he makes. The set of possible trajectories can be organized into a decision tree, with a split for every chance variable and every decision variable. We note that this construction gives rise to an exponentially large decision tree. Importantly, we never have to construct this tree explicitly. As we will show, we can use this tree as a conceptual construct, which forms the basis for defining a variable elimination algorithm. The VE algorithm works directly on the influence diagram, never constructing the exponentially large tree.

We begin by illustrating the decision tree construction on a simple example.

Example 23.7

Consider the possible trajectories that might be encountered by our entrepreneur of example 23.3. Initially, he has to decide whether to conduct the survey or not (C). He then gets to observe the value of the survey (S). He then has to decide whether to found the company or not (F). The variable M influences his utility, but he never observes it (at least not in a way that influences any of his decisions). Finally, the utility is selected based on the entire trajectory. We can organize this set of trajectories into a tree, where the first split is on the agent's decision C, the second split (on every branch) is nature's decision regarding the value of S, the third split is on the agent's decision F, and the final split is on M. At each leaf, we place the utility value corresponding to the scenario. Thus, for example, the agent's utility at the leaf of the trajectory c^1, s^1, f^1, m^1 is $V_S(c^1) + V_E(f^1, m^1) = -1 + 5$. The decision tree for this example is the same one shown in figure 23.4. ■

Note that the ordering of the nodes in the tree is defined by the agent's observations, not by the topological ordering of the underlying influence diagram. Thus, in this example, S precedes M, despite the fact that, viewed from the perspective of generative causal model, M is "determined

by nature" before S.

More generally, we assume (as stated earlier) that the influence diagram satisfies perfect recall relative to some temporal ordering $\prec$ on decisions. Without loss of generality, assume that $D_1 \prec \ldots \prec D_k$. We extend $\prec$ to a partial ordering over $\mathcal{X} \cup \mathcal{D}$ which is consistent with the information edges in the influence diagrams; that is, whenever W is a parent of D for some $D \in \mathcal{D}$ and $W \in \mathcal{X} \cup \mathcal{D}$, we have that $W \prec D$. This ordering is guaranteed to extend the total ordering $\prec$ over $\mathcal{D}$ postulated in definition 23.5, allowing us to abuse notation and use $\prec$ for both.

This partial ordering constrains the orderings that we can use to define the decision tree. Let $\boldsymbol{X}_1$ be the set of variables X such that $X \prec D_1$; these variables are the ones that the agent observes for the first time at decision D_1. More generally, let $\boldsymbol{X}_i$ be those variables X such that $X \prec D_i$ but not $X \prec D_{i-1}$. These variables are the ones that the agent observes for the first time at decision D_i. With the perfect recall assumption, the agent's decision rule at D_i can depend on all of $\boldsymbol{X}_1 \cup \ldots \cup \boldsymbol{X}_i \cup \{D_1, \ldots, D_{i-1}\}$. Let $\boldsymbol{Y}$ be the variables that are not observed prior to any decision. The sets $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, \boldsymbol{Y}$ form a disjoint partition of $\mathcal{X}$. We can then define a tree where the first split is on the set of possible assignments $\boldsymbol{x}_1$ to $\boldsymbol{X}_1$, the second is on possible decisions in $Val(D_1)$, and so on, and where the final split is on possible assignments $\boldsymbol{y}$ to $\boldsymbol{Y}$.

The choices at nature's chance moves are associated with probabilities. These probabilities are not the same as the generative probabilities (as reflected in the CPDs in the influence diagrams), but reflect the agent's subjective beliefs in nature's choices given the evidence observed so far.

Example 23.8

Consider the decision tree of figure 23.4, and consider nature's choice for the branch $S = s^1$ at the node corresponding to the trajectory $C = c^1$. The probability that the survey returns s^1 is the marginal probability $\sum_M P(M) \cdot P(s^1 \mid M, c^1)$. Continuing down the same branch, and assuming $F = f^1$, the branching probability for $M = m^1$ is the conditional probability of $M = m^1$ given s^1 (and the two decision variables, although these are irrelevant to this probability). ■

In general, consider a branch down the tree associated with the choices $\boldsymbol{x}_1, d_1, \ldots, \boldsymbol{x}_{i-1}, d_{i-1}$. At this vertex, we have a decision of nature, splitting on possible instantiations $\boldsymbol{x}_i$ to $\boldsymbol{X}_i$. We associate with this vertex a distribution $P(\boldsymbol{x}_i \mid \boldsymbol{x}_1, d_1, \ldots, \boldsymbol{x}_{i-1}, d_{i-1})$.

As written, this probability expression is not well defined, since we have not specified a distribution relative to which it is computed. Specifically, because we do not have a decision rule for the different decision variables in the influence diagram, we do not yet have a fully specified Bayesian network. We can ascribe semantics to this term using the following lemma:

Lemma 23.1

Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, d_1, \ldots, d_{i-1}$ be an assignment to $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{i-1}, D_1, \ldots, D_{i-1}$ respectively. Let σ_1, σ_2 be any two strategies in which $P_{\mathcal{B}_{\mathcal{I}[\sigma_i]}}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, d_1, \ldots, d_{i-1}) \neq 0$ ($i = 1, 2$). Then

$$P_{\mathcal{B}_{\mathcal{I}[\sigma_1]}}(\boldsymbol{X}_i \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, d_1, \ldots, d_{i-1}) = P_{\mathcal{B}_{\mathcal{I}[\sigma_2]}}(\boldsymbol{X}_i \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, d_1, \ldots, d_{i-1}).$$

The proof is left as an exercise (exercise 23.4). Thus, the probability of $\boldsymbol{X}_i$ given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, d_1, \ldots, d_{i-1}$ does not depend on the choice of strategy σ, and so we can define a probability for this event without defining a particular strategy σ. We use $P(\boldsymbol{X}_i \mid \boldsymbol{x}_1, d_1, \ldots, \boldsymbol{x}_{i-1}, d_{i-1})$ as shorthand for this uniquely defined probability distribution.

23.3.2 Sum-Max-Sum Rule

Given an influence diagram $\mathcal{I}$, we can construct the decision tree using the previous procedure and then simply run MEU-for-Decision-Trees (algorithm 23.1) over the resulting tree. The algorithm computes both the MEU value of the tree and the optimal strategy. We now show that this MEU value and strategy are also the optimal value and strategy for the influence diagram.

Our first key observation is that, in the decision tree we constructed, we can choose a different action at each t-node in the layer for a decision variable D. In other words, the decision tree strategy allows us to take a different action at D for each assignment to the decision and observation variables preceding D in $\prec$. The perfect-recall assumption asserts that these variables are precisely the parents of D in the influence diagram $\mathcal{I}$. Thus, a decision rule at D is precisely as expressive as the set of individual decisions at the t-nodes corresponding to D, and the decision tree algorithm is simply selecting a set of decision rules for all of the decision variables in $\mathcal{I}$ — that is, a complete strategy for $\mathcal{I}$.

Example 23.9

In the F layer (the third layer) of the decision tree in figure 23.4, we maximize over different possible values of the decision variable F. Importantly, this layer is not selecting a single decision, but a (possibly) different action at each node in the layer. Each of these nodes corresponds to an information state — an assignment to C and S. Altogether, the set of decisions at this layer selects the entire decision rule δ_F. ■

Note that the perfect recall assumption is critical here. The decision tree semantics (as we defined it) makes the implicit assumption that we can make an independent decision at each t-node in the tree. Hence, if D' follows D in the decision tree, then every variable on the path to D t-nodes also appears on the path to D' t-nodes. Thus, the decision tree semantics can be consistent with the influence diagram semantics only when the influence diagram satisfies the perfect recall assumption.

We now need to show that the strategy selected by this algorithm is the one that maximizes the expected utility for $\mathcal{I}$. To do so, let us examine more closely the expression computed by MEU-for-Decision-Trees when applied to the decision tree constructed before.

Example 23.10

In example 23.3, our computation for the value of the entire tree can be written using the following expression:

$$\max_C \sum_S P(S \mid C) \max_F \sum_M P(M \mid S, F, C)[V_S(C) + V_E(M, F)].$$

Note that we can simplify some of the conditional probability terms in this expression using the conditional independence properties of the network (which are also invariant under any choice of decision rules). For example, M is independent of F, C given S, so that $P(M \mid S, F, C) = P(M \mid S)$. ■

More generally, consider an influence diagram where, as before, the sequence of chance and decision variables is: $\boldsymbol{X}_1, D_1, \ldots, \boldsymbol{X}_k, D_k, \boldsymbol{Y}$. We can write the value of the decision-making situation using the following expression, known as the *sum-max-sum rule*:

sum-max-sum rule

$$\begin{aligned}\mathrm{MEU}[\mathcal{I}] = &\textstyle\sum_{\boldsymbol{X}_1} P(\boldsymbol{X}_1) \max_{D_1} \sum_{\boldsymbol{X}_2} P(\boldsymbol{X}_2 \mid \boldsymbol{X}_1, D_1) \max_{D_2} \ldots \\ &\textstyle\sum_{\boldsymbol{X}_k} P(\boldsymbol{X}_k \mid \boldsymbol{X}_1, \ldots, \boldsymbol{X}_{k-1}, D_1, \ldots, D_{k-1}) \\ &\textstyle\max_{D_k} \sum_{\boldsymbol{Y}} P(\boldsymbol{Y} \mid \boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, D_1, \ldots, D_k) U(\boldsymbol{Y}, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, D_1, \ldots, D_k).\end{aligned}$$

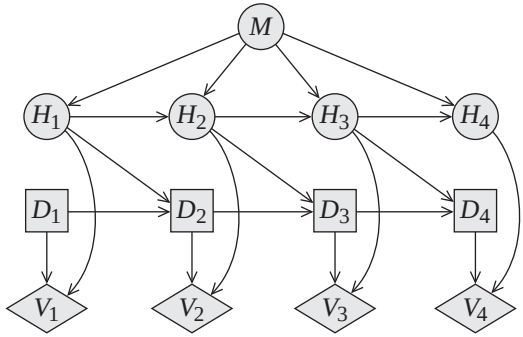


Figure 23.5 Iterated optimization versus variable elimination. An influence diagram that allows an efficient solution using iterated optimization, but where variable elimination techniques are considerably less efficient.

This expression is effectively performing the same type of backward induction that we used in decision trees.

We can now push in the conditional probabilities into the summations or maximizations. This operation is the inverse to the one we have used so often earlier in the book, where we move probability factors out of a summation or maximization; the same equivalence is used to justify both. Once all the probabilities are pushed in, all of the conditional probability expressions cancel each other, so that we obtain simply:

$$\text{MEU}[\mathcal{I}] = \sum_{\boldsymbol{X}_1} \max_{D_1} \sum_{\boldsymbol{X}_2} \max_{D_2} \ldots \sum_{\boldsymbol{X}_k} \max_{D_k} \sum_{\boldsymbol{Y}} P(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, \boldsymbol{Y} \mid D_1, \ldots, D_k) U(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, \boldsymbol{Y}, D_1, \ldots, D_k). \quad (23.4)$$

If we view this expression in terms of factors (as in equation (23.2)), we can decompose the joint probability $P(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, \boldsymbol{Y} \mid D_1, \ldots, D_k) = P(\mathcal{X} \mid \mathcal{D})$ as the product of all of the factors corresponding to the CPDs of the variables $\mathcal{X}$ in the influence diagram. The joint utility $U(\boldsymbol{X}_1, \ldots, \boldsymbol{X}_k, \boldsymbol{Y}, D_1, \ldots, D_k) = U(\mathcal{X}, \mathcal{D})$ is the sum of all of the utility variables in the network.

Now, consider a strategy σ — an assignment of actions to all of the agent t-nodes in the tree. Given a fixed strategy, the maximizations become vacuous, and we are simply left with a set of summations over the different chance variables in the network. It follows directly from the definitions that the result of this summation is simply the expected utility of σ in the influence diagram, as in equation (23.2). The fact that the sum-max-sum computation results in the MEU strategy now follows directly from the optimality of the strategy produced by the decision tree algorithm.

The form of equation (23.4) suggests an alternative method for computing the MEU value and strategy, one that does not require that we explicitly form a decision tree. Rather, we can apply a *variable elimination* algorithm that directly computes the the sum-max-sum expression: We eliminate both the chance and decision variables, one at a time, using the $\sum$ or $\max$

variable elimination

operations, as appropriate. At first glance, this approach appears straightforward, but the details are somewhat subtle. Unlike most of our applications of the variable elimination algorithm, which involve only two operations (either sum-product or max-product), this expression involves four — sum-marginalization, max-marginalization, factor product (for probabilities and utilities), and factor addition (for utilities). The interactions between these different operations require careful treatment, and the machinery required to handle them correctly has a significant effect on the design and efficiency of the variable elimination algorithm. The biggest complication arises from the fact that sum-marginalization and max-marginalization do not commute, and therefore elimination operations can be executed only in an order satisfying certain constraints; as we showed in section 13.2.3 and section 14.3.1, such constraints can cause inference even in simple networks to become intractable. The same issues arise here:

Example 23.11

Consider a setting where a student must take a series of exams in a course. The hardness H_i of each exam i is not known in advance, but one can assume that it depends on the hardness of the previous exam H_{i-1} (if the class performs well on one exam, the next one tends to be harder, and if the class performs poorly, the next one is often easier). It also depends on the overall meanness of the instructor. The student needs to decide how much to study for each exam (D_i); studying more makes her more likely to succeed in the exam, but it also reduces her quality of life. At the time the student needs to decide on D_i, she knows the difficulty of the previous one and whether she studied for it, but she does not remember farther back than that. The meanness of the instructor is never observed. The influence diagram is shown in figure 23.5.

If we apply a straightforward variable elimination algorithm based on equation (23.4), we would have to work from the inside out in an order that is consistent with the operations in the equation. Thus, we would first have to eliminate M, which is never observed. This step has the effect of creating a single factor over all of the H_i variables, whose size is exponential in k. ■

Fortunately, as we discuss in section 23.5, there are better solution methods for influence diagrams, which are not based on variable elimination and hence avoid some of these difficulties.

23.4 Computing Expected Utilities

In constructing a more efficient algorithm for finding the optimal decision in an influence diagram, we first consider the special case of an influence diagram with no decision variables. This problem is of interest in its own right, since it allows us to evaluate the expected utility of a given strategy. More importantly, it is also a key subroutine in the algorithm for finding an optimal strategy.

We begin our discussion with the even more restricted setting, where there is a single utility variable, and then discuss how it can be extended to the case of several utility variables. As we will see, although there are straightforward generalizations, an efficient implementation for this extension can involve some subtlety.

23.4.1 Simple Variable Elimination

Assume we have a single utility factor U. In this case, the expected utility is simply a product of factors: the CPDs of the chance variables, the decision rules, and the utility function of the

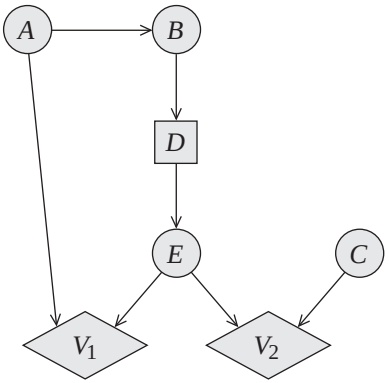


Figure 23.6 An influence diagram with multiple utility variables

single utility factor U, summed out over all of the variables in the network. **Thus, in the setting of a single utility variable, we can apply our standard variable elimination algorithm in a straightforward way, to the set of factors defining the expected utility.** Because variable elimination is well defined for any set of factors (whether derived from probabilities or not), there is no obstacle to applying it in this setting.

Example 23.12

Consider the influence diagram in figure 23.6. The influence diagram is drawn with two utility variables, but (proceeding with our assumption of a single utility variable) we analyze the computation for each of them in isolation, assuming it is the only utility variable in the network.

We begin with the utility variable V_1, and use the elimination ordering C, A, E, B, D. Note that C is barren relative to V_1 (that is, it has no effect on the utility V_1) and can therefore be ignored. (Eliminating C would simply produce the all 1's factor.) Eliminating A, we obtain

$$\mu_1^1(B, E) = \sum_A V_1(A, E)P(B \mid A)P(A).$$

Eliminating E, we obtain

$$\mu_2^1(B, D) = \sum_E P(E \mid D)\mu_1^1(B, E).$$

We can now proceed to eliminate D and B, to compute the final expected utility value.

Now, consider the same variable elimination algorithm, with the same ordering, applied to the utility variable V_2. In this case, C is not barren, so we compute:

$$\mu_1^2(E) = \sum_C V_2(C, E)P(C).$$

The variable A does not appear in the scope of μ_1^2, and hence we do not use the utility factor in this step. Rather, we obtain a standard probability factor:

$$\phi_1^2(B) = \sum_A P(A)P(B \mid A).$$

Eliminating E, we obtain:

$$\mu_2^2(D) = \sum_E P(E \mid D)\mu_1^2(E).$$

To eliminate B, we multiply $P(D \mid B)$ (which is a decision rule, and hence simply a CPD) with $\phi_1^2(B)$, and then marginalize out B from the resulting probability factor. Finally, we multiply the result with $\mu_2^2(D)$ to obtain the expected utility of the influence diagram, given the decision rule for D. ■

23.4.2 Multiple Utility Variables: Simple Approaches

An efficient extension to multiple utility variables is surprisingly subtle. One obvious solution is to collapse all of the utility factors into a single large factor $U = \sum_{V \in \mathcal{U}} V$. We are now back to the same situation as above, and we can run the variable elimination algorithm unchanged. Unfortunately, this solution can lead to unnecessary computational costs:

Example 23.13 *Let us return to the influence diagram of example 23.12, but where we now assume that we have both V_1 and V_2. In this simple solution, we add both together to obtain $U(A, E, C)$. If we now run our variable elimination process (with the same ordering), it produces the following factors: $\mu_1^U(A, E) = \sum_C P(C)U(A, C, E)$; $\mu_2^U(B, E) = \sum_A P(A)P(B \mid A)\mu_1^U(A, E)$; and $\mu_3^U(B, D) = \sum_E P(E \mid D)\mu_2^U(B, E)$. Thus, this process produces a factor over the scope A, C, E, which is not created by either of the two preceding subcomputations; if, for example, both A and C have a large domain, this factor might result in high computational costs.* ■

Thus, this simple solution requires that we sum up the individual subutility functions to construct a single utility factor whose scope is the union of the scopes of the subutilities. As a consequence, this transformation loses the structure of the utility function and creates a factor that may be exponentially larger. In addition to the immediate costs of creating this larger factor, factors involving more variables can also greatly increase the cost of the variable elimination algorithm by forcing us to multiply in more factors as variables are eliminated.

A second simple solution is based on the linearity of expectations:

$$\sum_{\mathcal{X} - \mathrm{Pa}_D} \prod_{X \in \mathcal{X}} P(X \mid \mathrm{Pa}_X)\Big(\sum_{i \,:\, V_i \in \mathcal{U}} V_i\Big) = \sum_{i \,:\, V_i \in \mathcal{U}} \left(\sum_{\mathcal{X} - \mathrm{Pa}_D} \prod_{X \in \mathcal{X}} P(X \mid \mathrm{Pa}_X) V_i \right).$$

Thus, we can run multiple separate executions of variable elimination, one for each utility factor V_i, computing for each of them an expected utility factor μ^i_{-D}; we then sum up these expected utility factors and optimize the decision rule relative to the resulting aggregated utility factor. The limitation of this solution is that, in some cases, it forces us to replicate work that arises for multiple utility factors.

Example 23.14 *Returning to example 23.12, assume that we replace the single variable E between D and the two utility variables with a long chain $D \to E_1 \to \ldots \to E_k$, where E_k is the parent of V_1 and V_2. If we do a separate variable elimination computation for each of V_1 and V_2, we would be executing twice the steps involved in eliminating $E_1, \ldots, E_k$, rather than reusing the computation for both utility variables.* ■

23.4.3 Generalized Variable Elimination ⋆

A solution that addresses both the limitations described before is to perform variable elimination with multiple utility factors simultaneously, but allow the algorithm to add utility factors to each other, as called for by the variable elimination algorithm. In other words, just as we multiply factors together when we eliminate a variable that they have in common, we would combine two utility factors together in the same situation.

Example 23.15

Let us return to example 23.12 using the same elimination ordering C, A, E. The first steps, of eliminating C and A, are exactly those we took in that example, as applied to each of the two utility factors separately. In other words, the elimination of C does not involve V_1, and hence produces precisely the same factor $\mu_1^1(B, E)$ as before; similarly, the elimination of A does not involve V_2, and produces $\mu_1^2(E), \phi_1^2(B)$. However, when we now eliminate E, we must somehow combine the two utility factors in which E appears. At first glance, it appears as if we can simply add these two factors together. However, a close examination reveals an important subtlety. The utility factor $\mu_1^2(E) = \sum_C P(C)V_2(C, E)$ is a function that defines, for each assignment to E, an expected utility given E. However, the entries in the other utility factor,

$$\begin{aligned}\mu_1^1(B, E) &= \sum_A P(A)P(B \mid A)V_1(A, E) \\ &= \sum_A P(B)P(A \mid B)V_1(A, E) = P(B)\sum_A P(A \mid B)V_1(A, E),\end{aligned}$$

do not *represent an expected utility; rather, they are a product of an expected utility with the probability $P(B)$. Thus, the two utility factors are on "different scales," so to speak, and cannot simply be added together. To remedy this problem, we must convert both utility factors into the "utility scale" before adding them together. To do so, we must keep track of $P(B)$ as we do the elimination and divide $\mu_1^1(B, E)$ by $P(B)$ to rescale it appropriately, before adding it to μ_1^2.* ■

Thus, in order to perform the variable elimination computation correctly with multiple utility variables, we must keep track not only of utility factors, but also of the probability factors necessary to normalize them. This intuition suggests an algorithm where our basic data structures — our factors — are actually pairs of factors $\gamma = (\phi, \mu)$, where ϕ is a probability factor, and μ is a utility factor. Intuitively, ϕ is the probability factor that can bring μ into the "expected utility scale." More precisely, assume for simplicity that the probability and utility factors in a joint factor have the same scope; we can make this assumption without loss of generality by simply increasing the scope of either or both factors, duplicating entries as required. Intuitively, the probability factor is maintained as an auxiliary factor, used to normalize the utility factor when necessary, so as to bring it back to the standard "utility scale." Thus, if we have a joint factor $(\phi(\boldsymbol{Y}), \mu(\boldsymbol{Y}))$, then $\mu(\boldsymbol{Y})/\phi(\boldsymbol{Y})$ is a factor whose entries are expected utilities associated with the different assignments $\boldsymbol{y}$.

Our goal is to define a variable-elimination-style algorithm using these joint factors. As for any variable elimination algorithm, we must define operations that combine factors, and operations that marginalize — sum out — variables out of a factor. We now define both of these steps. We consider, for the moment, factors associated only with probability variables and utility variables; we will discuss later how to handle decision variables.

Initially, each variable W that is associated with a CPD induces a probability factor ϕ_W; such variables include both chance variables in $\mathcal{X}$ and decision variables associated with a decision rule. (As we discussed, a decision rule for a decision variable D essentially turns D into a chance variable.) We convert ϕ_W into a joint factor γ_W by attaching to it an all-zeros utility factor over the same scope, $\mathbf{0}_{Scope[\phi_W]}$. Similarly, each utility variable $V \in \mathcal{U}$ is associated with a utility factor μ_V, which we convert to a joint factor by attaching to it an all-ones probability factor: $\gamma_V = (\mathbf{1}_{\mathrm{Pa}_V}, V)$ for $V \in \mathcal{U}$.

Intuitively, we want to multiply probability components (as usual) and add utility components. Thus, we define the *joint factor combination* operation as follows:

- For two joint factors $\gamma_1 = (\phi_1, \mu_1)$, $\gamma_2 = (\phi_2, \mu_2)$, we define the *joint factor combination* operation:

$$\gamma_1 \bigoplus \gamma_2 = (\phi_1 \cdot \phi_2, \mu_1 + \mu_2). \tag{23.5}$$

We see that, if all of the joint factors in the influence diagram are combined, we obtain a single (exponentially large) probability factor that defines the joint distribution over outcomes, and a single (exponentially large) utility factor that defines the utilities of the outcomes. Of course, this procedure is not one that we would ever execute; rather, as in variable elimination, we want to interleave combination steps and marginalization steps in a way that preserves the correct semantics.

The definition of the marginalization operation is subtler. Intuitively, we want the probability of an outcome to be multiplied with its utility. However, as suggested by example 23.15, we must take care that the utility factors derived as intermediate results all maintain the same scale, so that they can be correctly added in the factor combination operation. Thus, when marginalizing a variable W, we divide the utility factor by the associated probability factor, ensuring that it maintains its expected utility interpretation:

- For a joint factor $\gamma = (\phi, \mu)$ over scope $\boldsymbol{W}$, we define the *joint factor marginalization* operation for $\boldsymbol{W}' \subset \boldsymbol{W}$ as follows:

$$\mathit{marg}_{\boldsymbol{W}'}(\gamma) = \left(\sum_{\boldsymbol{W}'} \phi, \frac{\sum_{\boldsymbol{W}'} \phi \cdot \mu}{\sum_{\boldsymbol{W}'} \phi} \right). \tag{23.6}$$

Intuitively, this operation marginalizes out (that is, eliminates) the variables in $\boldsymbol{W}'$, handling both utility and probability factors correctly.

Finally, at the end of the process, we can combine the probability and utility factors to obtain a single factor that corresponds to the overall expected utility:

- For a joint factor $\gamma = (\phi, \mu)$, we define the *joint factor contraction* operation as the factor product of the two components:

$$\mathrm{cont}(\gamma) = \phi \cdot \mu. \tag{23.7}$$

To understand these definitions, consider again the problem of computing the expected utility for some (complete) strategy σ for the influence diagram $\mathcal{I}$. Thus, we now have a probability

Algorithm 23.2 Generalized variable elimination for joint factors in influence diagrams

Procedure Generalized-VE-for-IDs (
  Φ, // Set of joint (probability,utility) factors
  $W_1, \ldots, W_k$ // List of variables to be eliminated
)
for $i = 1, \ldots, k$
  $\Phi' \leftarrow \{\phi \in \Phi : W_i \in \mathit{Scope}[\phi]\}$
  $\psi \leftarrow \bigoplus_{\phi \in \Phi'} \phi$
  $\tau \leftarrow \mathit{marg}_{W_i}(\psi)$
  $\Phi \leftarrow \Phi - \Phi' \cup \{\tau\}$
$\phi^* \leftarrow \bigoplus_{\phi \in \Phi} \phi$
return ϕ^*

factor for each decision variable. Recall that our expected utility is defined as:

$$\mathrm{EU}[\mathcal{I}[\sigma]] = \sum_{W \in \mathcal{X} \cup \mathcal{D}} \prod_{W \in \mathcal{X} \cup \mathcal{D}} \phi_W \cdot (\sum_{V \in \mathcal{U}} \mu_V).$$

Let γ^* be the marginalization over all variables of the combination of all of the joint factors:

$$\gamma^* = (\phi^*, \mu^*) = \mathit{marg}_{\emptyset}(\bigoplus_{(W \in \mathcal{X} \cup \mathcal{U})} [\gamma_W]). \tag{23.8}$$

Note that the factor has empty scope and is therefore simply a pair of numbers. We can now show the following simple result:

Proposition 23.1 *For γ^* defined in equation (23.8), we have: $\gamma^* = (1, \mathrm{EU}[\mathcal{I}[\sigma]])$.*

The proof follows directly from the definitions and is left as an exercise (exercise 23.2).

Of course, as we discussed, we want to interleave the marginalization and combination steps. An algorithm implementing this idea is shown in algorithm 23.2. The algorithm returns a single joint factor (ϕ, μ).

Example 23.16 *Let us consider the behavior of this algorithm on the influence diagram of example 23.12, assuming again that we have a decision rule for D, so that we have only chance variables and utility variables. Thus, we initially have five joint factors derived from the probability factors for A, B, C, D, E; for example, we have $\gamma_B = (P(B \mid A), \mathbf{0}_{A,B})$. We have two joint factors γ_1, γ_2 derived from the utility variables V_1, V_2; for example, we have $\gamma_2 = (\mathbf{1}_{C,E}, V_2(C, E))$.*

Now, consider running our generalized variable elimination algorithm, using the elimination ordering C, A, E, B, D. Eliminating C, we first combine γ_C, γ_2 to obtain:

$$\gamma_C \bigoplus \gamma_2 = (P(C), V_2(C, E)),$$

where the scope of both components is taken to be C, E. We then marginalize C to obtain:

$$\begin{aligned} \gamma_3(E) &= \left(\mathbf{1}_E, \frac{\sum_C (P(C) V_2(C, E))}{\mathbf{1}_E}\right) \\ &= (\mathbf{1}_E, \boldsymbol{E}_{P(C)}[V_2(C, E)]). \end{aligned}$$

Continuing to eliminate A, we combine γ_A, γ_B, and γ_1 and marginalize A to obtain:

$$\begin{aligned}\gamma_4(B,E) &= \left(\sum_A P(A)P(B \mid A), \frac{\sum_A P(A)P(B \mid A)V_1(A,E)}{\sum_A P(A)P(B \mid A)}\right) \\ &= (P(B), \sum_A P(A \mid B)V_1(A,E)) \\ &= (P(B), \mathbf{E}_{P(A|B)}[V_1(A,E)]).\end{aligned}$$

Importantly, the utility factor here can be interpreted as the expected utility over V_1 given B, where the expectation is taken over values of A. It therefore keeps this utility factor on the same scale as the others, avoiding the problem of incomparable utility factors that we had in example 23.15.

We next eliminate E. We first combine γ_E, γ_3, and γ_4 to obtain:

$$(P(E \mid D)P(B), \mathbf{E}_{P(C)}[V_2(C,E)] + \mathbf{E}_{P(A|B)}[V_1(A,E)]).$$

Marginalizing E, we obtain:

$$\gamma_5(B,D) = (P(B), \mathbf{E}_{P(C,E|D)}[V_2(C,E)] + \mathbf{E}_{P(A,E|B,D)}[V_1(A,E)]).$$

To eliminate B, we first combine γ_5 and γ_D, to obtain:

$$(P(D \mid B)P(B), \mathbf{E}_{P(C,E|D)}[V_2(C,E)] + \mathbf{E}_{P(A,E|D)}[V_1(A,E)]).$$

We then marginalize B, obtaining:

$$\begin{aligned}\gamma_6(D) &= \left(P(D), \frac{\sum_B (\mathbf{E}_{P(C,E,D,B)}[V_2(C,E)] + \mathbf{E}_{P(A,E,D,B)}[V_1(A,E)])}{P(D)}\right) \\ &= \left(P(D), \mathbf{E}_{P(C,E|D)}[V_2(C,E)] + \mathbf{E}_{P(A,E|B,D)}[V_1(A,E)]\right).\end{aligned}$$

Finally, we have only to marginalize D, obtaining:

$$\gamma_7(\emptyset) = \left(1, \mathbf{E}_{P(C,E)}[V_2(C,E)] + \mathbf{E}_{P(A,E)}[V_1(A,E)]\right),$$

as desired. ■

How do we show that it is legitimate to reorder these marginalization and combination operators? In exercise 9.19, we defined the notion of generalized marginalize-combine factor operators and stated a result showing that, for any pair of operators satisfying certain conditions, any legal reordering of the operators led to the same result. In particular, this result implied, as special cases, correctness of sum-product, max-product, and max-sum variable elimination. The same analysis can be used for the operators defined here, showing the following result:

Theorem 23.1 *Let Φ be a set of joint factors over $\boldsymbol{Z}$.* Generalized-VE-for-IDs$(\Phi, \boldsymbol{W})$ *returns the joint factor*

$$\mathit{marg}_{\boldsymbol{W}}\left(\bigoplus_{(\gamma \in \Phi)} \gamma\right).$$

The proof is left as an exercise (exercise 23.3).

Note that the complexity of the algorithm is the same (up to a constant factor) as that of a standard VE algorithm, applied to an analogous set of factors — with the same scope — as our initial probability and utility factors. In other words, for a given elimination ordering, the cost of the algorithm grows as the induced tree-width of the graph generated by this initial set of factors.

So far, we have discussed the problem of computing the expected utility of a complete strategy. How can we apply these ideas to our original task, of optimizing a single decision rule? The idea is essentially the same as in section 23.4.1. As there, we apply Generalized-VE-for-IDs to eliminate all of the variables other than $\mathrm{Family}_D = \{D\} \cup \mathrm{Pa}_D$. In this process, the probability factor induced by the decision rule for D is only combined with the other factors at the final step of the algorithm, when the remaining factors are all combined. It thus has no effect on the factors produced up to that point. We can therefore omit ϕ_D from our computation, and produce a joint factor $\gamma_{-D} = (\phi_{-D}, \mu_{-D})$ over Family_D based only on the other factors in the network.

For any decision rule δ_D, if we run Generalized-VE-for-IDs on the factors in the original influence diagram plus a joint factor $\gamma_D = (\delta_D, \mathbf{0}_{\mathrm{Family}_D})$, we would obtain the factor

$$\gamma_{\delta_D} = \gamma_{-D} \bigoplus \gamma_D.$$

Rewriting this expression, we see that the overall expected utility for the influence diagram given the decision rule δ_D is then:

$$\sum_{\boldsymbol{w} \in Val(\mathrm{Pa}_D), d \in Val(D)} \mathrm{cont}(\gamma_{-D})(\boldsymbol{w}, d)\delta_D(\boldsymbol{w}).$$

Based on this observation, and on the fact that we can always select an optimal decision rule that is a deterministic function from $Val(\mathrm{Pa}_D)$ to $Val(D)$, we can easily optimize δ_D. For each assignment $\boldsymbol{w}$ to Pa_D, we select

$$\delta_D(\boldsymbol{w}) = \arg \max_{d \in Val(D)} \mathrm{cont}(\gamma_{-D})(\boldsymbol{w}, d).$$

As before, the problem of optimizing a single decision rule can be solved using a standard variable elimination algorithm, followed by a simple optimization. In this case, we must use a generalized variable elimination algorithm, involving both probability and utility factors.

23.5 Optimization in Influence Diagrams

We now turn to the problem of selecting an optimal strategy in an influence diagram. We begin with the simple case, where we have only a single decision variable. We then show how to extend these ideas to the more general case.

23.5.1 Optimizing a Single Decision Rule

We first make the important observation that, for the case of a single decision variable, the task of finding an optimal decision rule can be reduced to that of computing a single utility factor.

We begin by rewriting the expected utility of the influence diagram in a different order:

$$\mathrm{EU}[\mathcal{I}[\sigma]] = \sum_{D,\mathrm{Pa}_D} \delta_D \sum_{\mathcal{X}-\mathrm{Pa}_D} \prod_{X\in\mathcal{X}} P(X \mid \mathrm{Pa}_X)(\sum_{V\in\mathcal{U}} V). \tag{23.9}$$

Our task is to select δ_D.

expected utility factor

We now define the *expected utility factor* to be the value of the internal summation in equation (23.9):

$$\mu_{-D} = \sum_{\mathcal{X}-\mathrm{Pa}_D} \prod_{X\in\mathcal{X}} P(X \mid \mathrm{Pa}_X)(\sum_{V\in\mathcal{U}} V). \tag{23.10}$$

This expression is the marginalization of this product onto the variables $D \cup \mathrm{Pa}_D$; importantly, it does not depend on our choice of decision rule for D. Given μ_{-D}, we can compute the expected utility for any decision rule δ_D as:

$$\sum_{D,\mathrm{Pa}_D} \delta_D \mu_{-D}(D, \mathrm{Pa}_D).$$

Our goal is to find δ_D that maximizes this expression.

Proposition 23.2 *Consider an influence diagram $\mathcal{I}$ with a single decision variable D. Letting μ_{-D} be as in equation (23.10), the optimal decision rule for D in $\mathcal{I}$ is defined as:*

$$\delta_D(\boldsymbol{w}) = \arg\max_{d\in Val(D)} \mu_{-D}(d, \boldsymbol{w}) \qquad \forall \boldsymbol{w} \in Val(\mathrm{Pa}_D). \tag{23.11}$$

The proof is left as an exercise (see exercise 23.1).

Thus, we have shown how the problem of optimizing a single decision rule can be solved very simply, once we have computed the utility factor $\mu_{-D}(D, \mathrm{Pa}_D)$.

Importantly, any of the algorithms described before, whether the simpler ones in section 23.4.1 and 23.4.2, or the more elaborate generalized variable elimination algorithm of section 23.4.3, can be used to compute this expected utility factor. We simply structure our elimination ordering to eliminate only the variables other than D, Pa_D; we then combine all of the factors that are computed via this process, to produce a single integrated factor $\mu_{-D}(D, \mathrm{Pa}_D)$. We can then use this factor as in proposition 23.2 to find the optimal decision rule for D, and thereby solve the influence diagram.

How do we generalize this approach to the case of an influence diagram with multiple decision rules $D_1, \ldots, D_k$? In principle, we could generate an expected utility factor where we eliminated all variables other than the union $\boldsymbol{Y} = \cup_i(\{D_i\} \cup \mathrm{Pa}_{D_i})$ of all of the decision variables and all of their parents. Intuitively, this factor would specify the expected utility of the influence diagram given an assignment to $\boldsymbol{Y}$. However, in this case, the optimization problem is much more complex, in that it requires that we consider simultaneously the decisions at all of the decision variables in the network. Fortunately, as we show in the next section, we can perform this multivariable optimization using localized optimization steps over single variables.

23.5.2 Iterated Optimization Algorithm

In this section, we describe an iterated approach that breaks up the problem into a series of simpler ones. Rather than optimize all of the decision rules at the same time, we fix all of

the decision rules but one, and then optimize the remaining one. The problem of optimizing a single decision rule is significantly simpler, and admits very efficient algorithms, as shown in section 23.5.1. This algorithm is very similar in its structure to the local optimization approach for marginal MAP problems, presented in section 13.7. Both algorithms are intended to deal with the same computational bottleneck: the exponentially large factors generated by a constrained elimination ordering. They both do so by optimizing one variable at a time, keeping the others fixed. The difference is that here we are optimizing an entire decision rule for the decision variable, whereas there we are simply picking a single value for the MAP variable.

We will show that, under certain assumptions, this iterative approach is guaranteed to converge to the optimal strategy. Importantly, this approach also applies to influence diagrams with imperfect recall, and can therefore be considerably more efficient.

The basic idea behind this algorithm is as follows. The algorithm proceeds by sequentially optimizing individual decision rules. We begin with some (almost) arbitrary strategy σ, which assigns a decision rule to all decision variables in the network. We then optimize a single decision rule relative to our current assignment to the others. This decision rule is used to update σ, and another decision rule is now optimized relative to the new strategy. More precisely, let σ_{-D} denote the decision rules in a strategy σ other than the one for D. We say that a decision rule δ_D is *locally optimal* for a strategy σ if, for any other decision rule δ'_D,

locally optimal decision rule

$$\mathrm{EU}[\mathcal{I}[(\sigma_{-D}, \delta_D)]] \geq \mathrm{EU}[\mathcal{I}[(\sigma_{-D}, \delta'_D)]].$$

Our algorithm starts with some strategy σ, and then iterates over different decision variables D. It then selects a locally optimal decision rule δ_D for σ, and updates σ by replacing σ_D with the new δ_D. Note that the influence diagram $\mathcal{I}[\sigma_{-D}]$ is an influence diagram with the single decision variable D, which can be solved using a variety of methods, as described earlier. The algorithm terminates when no decision rule can be improved by this process.

Perhaps the most important property of this algorithm is its ability to deal with the main computational limitation of the simple variable elimination strategy described in section 23.3.2: the fact that the constrained variable elimination ordering can require creation of large factors even when the network structure does not force them.

Example 23.17

Consider again example 23.11; here, we would begin with some set of decision rules for all of $D_1, \ldots, D_k$. We would then iteratively compute the expected utility factor μ_{-D_i} for one of the D_i variables, using the (current) decision rules for the others. We could then optimize the decision rule for D_i, and continue the process. Importantly, the only constraint on the variable elimination ordering is that D_i and its parents be eliminated last. With these constraints, the largest factor induced in any of these variable elimination procedures has size 4, avoiding the exponential blowup in k that we saw in example 23.11. ■

In a naive implementation, the algorithm runs variable elimination multiple times — once for each iteration — in order to compute μ_{-D_i}. However, using the approach of joint (probability, utility) factors, as described in section 23.4.3, we can provide a very efficient implementation as a clique tree. See exercise 23.10.

So far, we have ignored several key questions that affect both the algorithm's complexity and its correctness. Most obviously, we can ask whether this iterative algorithm even converges. When we optimize D, we either improve the agent's overall expected utility or we leave the

decision rule unchanged. Because the expected utility is bounded from above and the total number of strategies is discrete, the algorithm cannot improve the expected utility indefinitely. Thus, at some point, no additional improvements are possible, and the algorithm will terminate. A second question relates to the quality of the solution obtained. Clearly, this solution is locally optimal, in that no change to a single decision rule can improve the agent's expected utility. However, local optimality does not, in general, imply that the strategy is globally optimal.

Example 23.18

Consider an influence diagram containing only two decision variables D_1 and D_2, and a utility variable $V(D_1, D_2)$ defined as follows:

$$V(d_1, d_2) = \begin{cases} 2 & d_1 = d_2 = 1 \\ 1 & d_1 = d_2 = 0 \\ 0 & d_1 \neq d_2. \end{cases}$$

The strategy $(0, 0)$ is locally optimal for both decision variables, since the unique optimal decision for D_i when $D_j = 0$ $(j \neq i)$ is $D_i = 0$. On the other hand, the globally optimal strategy is $(1, 1)$. ■

However, under certain conditions, local optimality does imply global optimality, so that the iterated optimization process is guaranteed to converge to a globally optimal solution. These conditions are more general than perfect recall, so that this algorithm works in every case where the algorithm of the previous section applies. In this case, we can provide an ordering for applying the local optimization steps that guarantees that this process converges to the globally optimal strategy after modifying each decision rule exactly once. However, this algorithm also applies to networks that do not satisfy the perfect recall assumption, and in certain such cases it is even guaranteed to find an optimal solution. By relaxing the perfect recall assumption, we can avoid some of the exponential blowup of the decision rules in terms of the number of decisions in the network.

23.5.3 Strategic Relevance and Global Optimality ⋆

The algorithm described before iteratively changes the decision rule associated with individual decision variables. In general, changing the decision rule for one variable D' can cause a decision rule previously optimal for another variable D to become suboptimal. Therefore, the algorithm must revisit D and possibly select a new decision rule for it. In this section, we provide conditions under which we can guarantee that changing the decision rule for D' will not necessitate a change in the decision rule for D. In other words, we define conditions under which the decision rule for D' may not be relevant for optimizing the decision rule for D. Thus, if we choose a decision rule for D and later select one for D', we do not have to revisit the selection made for D. As we show, under certain conditions, this criterion allows us to optimize all of the decision rules using a single iteration through them.

23.5.3.1 Strategic Relevance

Intuitively, we would like to define a decision variable D' as *strategically relevant* to D if, to optimize the decision rule at D, the decision maker needs to consider the decision rule at D'. That is, we want to say that D' is is relevant to D if there is a partial strategy profile σ over

$\mathcal{D} - \{D, D'\}$, two decision rules $\delta_{D'}$ and $\delta'_{D'}$, and a decision rule δ_D, such that $(\sigma, \delta_D, \delta_{D'})$ is optimal, but $(\sigma, \delta_D, \delta'_{D'})$ is not.

Example 23.19

Consider a simple influence diagram where we have two decision variables $D_1 \rightarrow D_2$, and a utility $V(D_1, D_2)$ that is the same as the one used in example 23.18. Pick an arbitrary decision rule δ_{D_1} (not necessarily deterministic), and consider the problem of optimizing δ_{D_2} relative to δ_{D_1}. The overall expected utility for the agent is

$$\sum_{d_1} \delta_{D_1}(d_1) \sum_{d_2} \delta_{D_2}(d_2 \mid d_1) V(d_1, d_2).$$

An optimal decision for D_2 given the information state d_1 is $\arg\max_{d_2} V(d_1, d_2)$, regardless of the choice of decision rule for D_1. Thus, in this setting, we can pick an arbitrary decision rule δ_{D_1} and optimize δ_{D_2} relative to it; our selected decision rule will then be locally optimal relative to any decision rule for D_1. However, there is a subtlety that makes the previous statement false in certain settings. Let $\delta'_{D_1} = d_1^0$. Then one optimal decision rule for D_2 is $\delta'_{D_2}(d_1^0) = \delta'_{D_2}(d_1^1) = d_2^0$. Clearly, d_2^0 is the right choice when $D_1 = d_1^0$, but it is suboptimal when $D_1 = d_1^1$. However, because δ_{D_1} gives this latter event probability 0, this choice for δ_{D_2} is locally optimal relative to δ_{D_1}. ■

As this example shows, a decision rule can make arbitrary choices in information states that have probability zero without loss in utility. In particular, because δ'_{D_1} assigns probability zero to d_1^1, the "suboptimal" δ'_{D_2} is locally optimal relative to δ'_{D_1}; however, δ'_{D_2} is not locally optimal relative to other decision rules for D_1. Thus, if we use the previous definition, D_1 appears relevant to D_2 despite our intuition to the contrary. We therefore want to avoid probability-zero events, which allow situations such as this. We say that a decision rule is *fully mixed* if each probability distribution $\delta_D(D \mid \mathrm{pa}_D)$ assigns nonzero probability to all values of D. We can now formally define strategic relevance.

fully mixed decision rule

Definition 23.7

strategic relevance

Let D and D' be decision nodes in an influence diagram $\mathcal{I}$. We say that D' is strategically relevant *to D (or that D* strategically relies *on D') if there exist:*

- *a partial strategy profile σ over $\mathcal{D} - \{D, D'\}$;*
- *two decision rules $\delta_{D'}$ and $\delta'_{D'}$ such that $\delta_{D'}$ is fully mixed;*
- *a decision rule δ_D that is optimal for $(\sigma, \delta_{D'})$ but not for $(\sigma, \delta'_{D'})$.* ■

This definition does not provide us with an operative procedure for determining relevance. We can obtain such a procedure by considering an alternative mathematical characterization of the notion of local optimality.

Proposition 23.3

Let δ_D be a decision rule for a decision variable D in $\mathcal{I}$, and let σ be a strategy for $\mathcal{I}$. Then δ_D is locally optimal for σ if and only if for every instantiation $\boldsymbol{w}$ of Pa_D where $P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(\boldsymbol{w}) > 0$, the probability distribution $\delta_D(D \mid \boldsymbol{w})$ is a solution to

$$\arg\max_{q(D)} \sum_{d \in Val(D)} q(d) \sum_{V \in \mathcal{U}_{\succ D}} \sum_{v \in Val(V)} P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(v \mid d, \boldsymbol{w}) \cdot v, \tag{23.12}$$

where $\mathcal{U}_{\succ D}$ is the set of utility nodes in $\mathcal{U}$ that are descendants of D in $\mathcal{I}$.

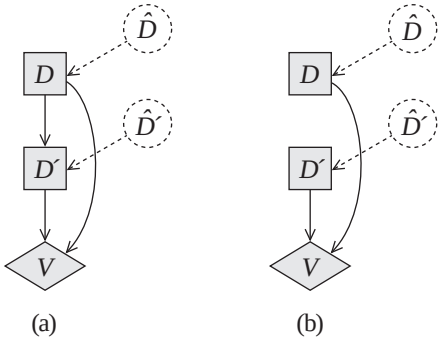


Figure 23.7 Influence diagrams, augmented to test for s-reachability

The proof is left as an exercise (exercise 23.6).

The significance of this result arises from two key points. First, the only probability expressions appearing in the optimization criterion are of the form $P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(V \mid \mathrm{Family}_D)$ for some utility variable V and decision variable D. Thus, we care about a decision rule $\delta_{D'}$ only if the CPD induced by this decision rule affects the value of one of these probability expressions. Second, the only utility variables that participate in these expressions are those that are descendants of D in the network.

23.5.3.2 S-Reachability

requisite CPD

We have reduced our problem to one of determining which decision rule CPDs might affect the value of some expression $P_{\mathcal{B}_{\mathcal{I}[\sigma]}}(V \mid \mathrm{Family}_D)$, for $V \in \mathcal{U}_{\succ D}$. In other words, we need to determine whether the decision variable is a *requisite CPD* for this query. We also encountered this question in a very similar context in section 21.3.1, when we wanted to determine whether an intervention (that is, a decision) was relevant to a query. As described in exercise 3.20, we can determine whether the CPD for a variable Z is requisite for answering a query $P(\boldsymbol{X} \mid \boldsymbol{Y})$ with a simple graphical criterion: We introduce a new "dummy" parent $\widehat{Z}$ whose values correspond to different choices for the CPD of Z. Then Z is a requisite probability node for $P(\boldsymbol{X} \mid \boldsymbol{Y})$ if and only if $\widehat{Z}$ has an active trail to $\boldsymbol{X}$ given $\boldsymbol{Y}$.

Based on this concept and equation (23.12), we can define *s-reachability* — a graphical criterion for detecting strategic relevance.

Definition 23.8 s-reachable

A decision variable D' is s-reachable *from a decision variable D in an ID $\mathcal{I}$ if there is some utility node $V \in \mathcal{U}_{\succ D}$ such that if a new parent $\widehat{D'}$ were added to D', there would be an active path in $\mathcal{I}$ from $\widehat{D'}$ to V given Family_D, where a path is active in an ID if it is active in the same graph, viewed as a BN.* ■

Note that unlike d-separation, s-reachability is not necessarily a symmetric relation.

Example 23.20

Consider the simple influence diagrams in figure 23.7, representing example 23.19 and example 23.18 respectively. In (a), we have a perfect-recall setting. Because the agent can observe D when deciding

on the decision rule for D', he does not need to know the decision rule for D in order to evaluate his options at D'. Thus, D' does not strategically rely on D. Indeed, if we add a dummy parent $\widehat{D}$ to D, we have that V is d-separated from $\widehat{D}$ given $\mathrm{Family}_{D'} = \{D, D'\}$. Thus, D is not s-reachable from D'. Conversely, the agent's decision rule at D' does influence his payoff at D, and so D' is relevant to D. Indeed, if we add a dummy parent $\widehat{D'}$ to D', we have that V is not d-separated from $\widehat{D'}$ given D, Pa_D.

By contrast, in (b), the agent forgets his action at D when observing D'; as his utility node is influenced by both decisions, we have that each decision is relevant to the other. The s-reachability analysis using d-separation from the dummy parents supports this intuition. ■

The notion of s-reachability is sound and complete for strategic relevance (almost) in the same sense that d-separation is sound and complete for independence in Bayesian networks. As for d-separation, the soundness result is very strong: without s-reachability, one decision cannot be relevant to another.

Theorem 23.2 *If D and D' are two decision nodes in an ID $\mathcal{I}$ and D' is not s-reachable from D in $\mathcal{I}$, then D does not strategically rely on D'.*

Proof Let σ be a strategy profile for $\mathcal{I}$, and let δ_D be a decision rule for D that is optimal for σ. Let $\mathcal{B} = \mathcal{B}_{\mathcal{I}[\sigma]}$. By proposition 23.3, for every $\boldsymbol{w} \in Val(\mathrm{Pa}_D)$ such that $P_{\mathcal{B}}(\boldsymbol{w}) > 0$, the distribution $\delta_D(D \mid \boldsymbol{w})$ must be a solution of the maximization problem:

$$\arg\max_{P(D)} \sum_{d \in Val(D)} P(d) \sum_{V \in \mathcal{U}_{\succ D}} \sum_{v \in Val(V)} P_{\mathcal{B}}(v \mid d, \boldsymbol{w}) \cdot v. \tag{23.13}$$

Now, let σ' be any strategy profile for $\mathcal{I}$ that differs from σ only at D', and let $\mathcal{B}' = \mathcal{B}_{\mathcal{I}[\sigma']}$. We must construct a decision rule δ'_D for D that agrees with δ_D on all $\boldsymbol{w}$ where $P_{\mathcal{B}}(\boldsymbol{w}) > 0$, and that is optimal for σ'. By proposition 23.3, it suffices to show that for every $\boldsymbol{w}$ where $P_{\mathcal{B}'}(\boldsymbol{w}) > 0$, $\delta'_D(D \mid \boldsymbol{w})$ is a solution of:

$$\arg\max_{P(D)} \sum_{d \in Val(D)} P(d) \sum_{V \in \mathcal{U}_{\succ D}} \sum_{v \in Val(V)} P_{\mathcal{B}'}(v \mid d, \boldsymbol{w}) \cdot v. \tag{23.14}$$

If $P_{\mathcal{B}}(\boldsymbol{w}) = 0$, then our choice of $\delta'_D(D \mid \boldsymbol{w})$ is unconstrained; we can simply select a distribution that satisfies equation (23.14). For other $\boldsymbol{w}$, we must let $\delta'_D(D \mid \boldsymbol{w}) = \delta_D(D \mid \boldsymbol{w})$. We know that $\delta_D(D \mid \boldsymbol{w})$ is a solution of equation (23.13), and the two expressions are different only in that equation (23.13) uses $P_{\mathcal{B}}(v \mid d, \boldsymbol{w})$ and equation (23.14) uses $P_{\mathcal{B}'}(v \mid d, \boldsymbol{w})$. The two networks $\mathcal{B}$ and $\mathcal{B}'$ differ only in the CPD for D'. Because D' is not a requisite probability node for any $V \in \mathcal{U}_{\succ D}$ given D, Pa_D, we have that $P_{\mathcal{B}}(v \mid d, \boldsymbol{w}) = P_{\mathcal{B}'}(v \mid d, \boldsymbol{w})$, and that $\delta'_D(D \mid \boldsymbol{w}) = \delta_D(D \mid \boldsymbol{w})$ is a solution of equation (23.14), as required. ■

Thus, s-reachability provides us with a sound criterion for determining which decision variables D' are strategically relevant for D. As for d-separation, the completeness result is not as strong: s-reachability does not imply relevance in *every* ID. We can choose the probabilities and utilities in the ID in such a way that the influence of one decision rule on another does not manifest itself. However, s-reachability is the most precise graphical criterion we can use: it will not identify a strategic relevance unless that relevance actually exists in some ID that has the given graph structure.

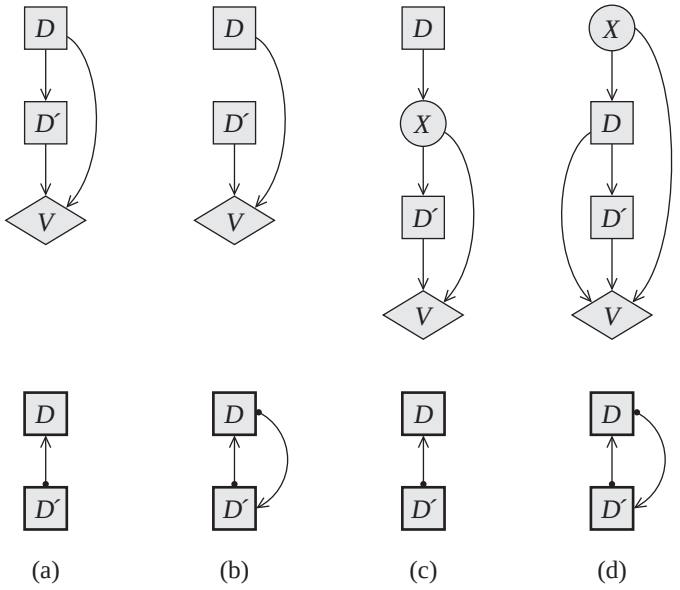


Figure 23.8 Four simple influence diagrams (top), and their relevance graphs (bottom).

Theorem 23.3 *If a node D' is s-reachable from a node D in an ID, then there is some ID with the same graph structure in which D strategically relies on D'.*

This result is roughly analogous to theorem 3.4, which states that there exists some parameterization that manifests the dependencies not induced by d-separation. A result analogous to the strong completeness result of theorem 3.5 is not known for this case.

23.5.3.3 The Relevance Graph

We can get a global view of the strategic dependencies between different decision variables in an influence diagram by putting them within a single graphical data structure.

Definition 23.9 relevance graph

The relevance graph *for an influence diagram $\mathcal{I}$ is a directed (possibly cyclic) graph whose nodes correspond to the decision variables $\mathcal{D}$ in $\mathcal{I}$, and where there is a directed edge $D' \rightarrow D$ if D' is strategically relevant to D.* ■

To construct the graph for a given ID, we need to determine, for each decision node D, the set of nodes D' that are s-reachable from D. Using standard methods from chapter 3, we can find this set for any given D in time linear in the number of chance and decision variables in the ID. By repeating the algorithm for each D, we can derive the relevance graph in time $O((n+k)k)$ where $n = |\mathcal{X}|$ and $k = |\mathcal{D}|$.

Recall our original statement that a decision node D strategically relies on a decision node D' if one needs to know the decision rule for D' in order to evaluate possible decision rules for

D. Intuitively, if the relevance graph is acyclic, we have a decision variable that has no parents in the graph, and hence relies on no other decisions. We can optimize the decision rule at this variable relative to some arbitrary strategy for the other decision rules. Having optimized that decision rule, we can fix its strategy and proceed to optimize the next one. Conversely, if we have a cycle in the relevance graph, then we have some set of decisions all of which rely on each other, and their decision rules need to be optimized together. In this case, the simple iterative approach we described no longer applies.

However, before we describe this iterative algorithm formally and prove its correctness, it is instructive to examine some simple IDs and see when one decision node relies on another.

Example 23.21

Consider the four examples shown in figure 23.8, all of which relate to a setting where the agent first makes decision D and then D'. Examples (a) and (b) are the ones we previously saw in example 23.20, showing the resulting relevance graphs. As we saw, in (a), we have that D relies on D' but not vice versa, leading to the structure shown on the bottom. In (b) we have that each decision relies on the other, leading to a cyclic relevance graph. Example (c) represents a situation where the agent does not remember D when making the decision D'. However, the agent knows everything he needs to about D: his utility does not depend on D directly, but only on the chance node, which he can observe. Hence D' does not rely on D.

One might conclude that a decision node D' never relies on another D when D is observed by D', but the situation is subtler. Consider example (d), which represents a simple card game: the agent observes a card and decides whether to bet (D); at a later stage, the agent remembers only his bet but not the card, and decides whether to raise his bet (D'); the utility of both depends on the total bet and the value of the card. Even though the agent does remember the actual decision at D, he needs to know the decision rule for D in order to know what the value of D tells him about the value of the card. Thus, D' relies on D; indeed, when D is observed, there is an active trail from a hypothetical parent $\hat{D}$ that runs through the chance node to the utility node. ■

However, it is the case that perfect recall — remembering both the previous decisions and the previous observations, does imply that the underlying relevance graph is acyclic.

Theorem 23.4

Let $\mathcal{I}$ be an influence diagram satisfying the perfect recall assumption. Then the relevance graph for $\mathcal{I}$ is acyclic.

The proof follows directly from properties of d-separation, and it is left as an exercise (exercise 23.7). We note that the ordering of the decisions in the relevance graph will be the opposite of the ordering in the original ID, as in figure 23.8a.

23.5.3.4 Global Optimality

Using the notion of a relevance graph, we can now provide an algorithm that, under certain conditions, is guaranteed to find an MEU strategy for the influence diagram. In particular, consider an influence diagram $\mathcal{I}$ whose relevance graph is acyclic, and let $D_1, \ldots, D_k$ be a topological ordering of the decision variables according to the relevance graph. We now simply execute the algorithm of section 23.5.2 in the order $D_1, \ldots, D_k$.

Why does this algorithm guarantee global optimality of the inferred strategy? When selecting the decision rule for D_i, we have two cases: for $j < i$, by induction, the decision rules for D_j

Algorithm 23.3 Iterated optimization for influence diagrams with acyclic relevance graphs

```
Procedure Iterated-Optimization-for-IDs (
    I,    // Influence diagram
    G     // Acyclic relevance graph for I
)
1   Let D_1, ..., D_k be an ordering of D that is a topological ordering for G
2   Let σ^0 be some fully mixed strategy for I
3   for i = 1, ..., k
4     Choose δ_{D_i} to be locally optimal for σ^{i-1}
5     σ^i ← (σ^i_{-D_i}, δ_{D_i})
6   return σ^k
```

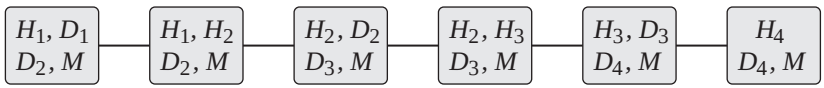


Figure 23.9 Clique tree for the imperfect-recall influence diagram of figure 23.5. Although the network has many cascaded decisions, our ability to "forget" previous decisions allows us to solve the problem using a bounded tree-width clique tree.

are already stable, and so will never need to change; for $j > i$, the decision rules for D_j are irrelevant, so that changing them will not require revisiting D_i.

One subtlety with this argument relates, once again, to the issue of probability-zero events. If our arbitrary starting strategy σ assigns probability zero to a certain decision $d \in Val(D)$ (in some setting), then the local optimization of another decision rule D' might end up selecting a suboptimal decision for the zero probability cases. If subsequently, when optimizing the decision rule for D, we ascribe nonzero probability to $D = d$, our overall strategy will not be optimal. To avoid this problem, we can use as our starting point any fully mixed strategy σ. One obvious choice is simply the strategy that, at each decision D and for each assignment to Pa_D, selects uniformly at random between all of the possible values of D.

The overall algorithm is shown in algorithm 23.3.

Theorem 23.5 *Applying* Iterated-Optimization-for-IDs *on an influence diagram $\mathcal{I}$ whose relevance graph is acyclic, returns a globally optimal strategy for $\mathcal{I}$.*

The proof is not difficult and is left as an exercise (exercise 23.8).

Thus, this algorithm, by iteratively optimizing individual decision rules, finds a globally optimal solution. The algorithm applies to any influence diagram whose relevance graph is acyclic, and hence to any influence diagrams satisfying the perfect recall assumption. Hence, it is at least as general as the variable elimination algorithm of section 23.3. However, as we saw, some influence diagrams that violate the perfect recall assumption have acyclic relevance graphs nonetheless; this algorithm also applies to such cases.

Example 23.22 *Consider again the influence diagram of example 23.11. Despite the lack of perfect recall in this*

network, an s-reachability analysis shows that each decision variable D_i strategically relies only on D_j for $j > i$. For example, if we add a dummy parent $\widehat{D_1}$ to D_1, we can verify that it is d-separated from V_3 and V_4 given $\mathrm{Pa}_{D_3} = \{H_2, D_2\}$, so that the resulting relevance graph is acyclic. ■

The ability to deal with problems where the agent does not have to remember his entire history can provide very large computational savings in large problems. Specifically, we can solve this influence diagram using the clique tree of figure 23.9, at a cost that grows linearly rather than exponentially in the number of decision variables.

This algorithm is guaranteed to find a globally optimal solution only in cases where the relevance graph is acyclic. However, we can extend this algorithm to find a globally optimal solution in more general cases, albeit at some computational cost. In this extension, we simultaneously optimize the rules for subsets of interdependent decision variables. Thus, for example, in example 23.18, we would optimize the decision rules for D_1 and D_2 together, rather than each in isolation. (See exercise 23.9.) This approach is guaranteed to find the globally optimal strategy, but it can be computationally expensive, depending on the number of interdependent decisions that must be considered together.

Box 23.B — Case Study: Coordination Graphs for Robot Soccer. *One subclass of problem in decision making is that of making a joint decision for a team of agents with a shared utility function. Let the world state be defined by a set of variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$. We now have a team of m agents, each with a decision variable A_i. The team's utility function is described by a function $U(\boldsymbol{X}, \boldsymbol{A})$ (for $\boldsymbol{A} = \{A_1, \ldots, A_m\}$). Given an assignment $\boldsymbol{x}$ to $X_1, \ldots, X_n$, our goal is to find the optimal* joint action $\arg\max_{\boldsymbol{a}} U(\boldsymbol{x}, \boldsymbol{a})$.

joint action

In a naive encoding, the representation of the utility function grows exponentially both in the number of state variables and in the number of agents. However, we can come up with more efficient algorithms by exploiting the same type of factorization that we have utilized so far. In particular, we assume that we can decompose U as a sum of subutility functions, each of which depends only on the actions of some subset of the agents. More precisely,

$$U(X_1, \ldots, X_n, A_1, \ldots, A_m) = \sum_i V_i(\boldsymbol{X}_i, \boldsymbol{A}_i),$$

where V_i is some subutility function with scope $\boldsymbol{X}_i, \boldsymbol{A}_i$.

This optimization problem is simply a max-sum problem over a factored function, a problem that is precisely equivalent to the MAP problem that we addressed in chapter 13. Thus, we can apply any of the algorithms we described there. In particular, max-sum variable elimination can be used to produce optimal joint actions, whereas max-sum belief propagation can be used to construct approximate max-marginals, which we can decode to produce approximate solutions. The application of these message passing algorithms in this type of distributed setting is satisfying, since the decomposition of the utility function translates to a limited set of interactions between agents who need to coordinate their choice of actions. Thus, this approach has been called a coordination graph.

coordination graph

Kok, Spaan, and Vlassis (2003), in their UvA (Universiteit van Amsterdam) Trilearn team, applied coordination graphs to the RoboSoccer *domain, a particularly challenging application of decision*

RoboSoccer

making under uncertainty. RoboSoccer is an annual event where teams of real or simulated robotic agents participate in a soccer competition. This application requires rapid decision making under uncertainty and partial observability, along with coordination between the different team members. The simulation league allows teams to compete purely on the quality of their software, eliminating the component of hardware design and maintenance. However, key challenges are faithfully simulated in this environment. For example, each agent can sense its environment via only three sensors: a visual sensor, a body sensor, and an aural sensor. The visual sensor measures relative distance, direction, and velocity of the objects in the player's current field of view. Noise is added to the true quantities and is larger for objects that are farther away. The agent has only a partial view of the world and needs to take viewing actions (such as turning its neck) deliberately in order to view other parts of the field. Players in the simulator have different abilities; for example, some can be faster than others, but they will also tire more easily. Overall, this tournament provides a challenge for real-time, multiagent decision-making architectures.

Kok et al. hand-coded a set of utility rules, each of which represents the incremental gain or loss to the team from a particular combination of joint actions. At every time point t they instantiate the variables representing the current state and solve the resulting coordination graph. Note that there is no attempt to address the problem of sequential decision making, where our choice of action at time t should consider its effect on actions at subsequent time points. The myopic nature of the decision making is based on the assumption that the rules summarize the long-term benefit to the team from a particular joint action.

To apply this framework in this highly dynamic, continuous setting, several adaptations are required. First, to reduce the set of possible actions that need to be considered, each agent is assigned a role: interceptor, passer, receiver, or passive. The assignment of roles is computed directly from the current state information. For example, the fastest player close to the ball will be assigned the passer role when he is able to kick the ball, and the interceptor role otherwise. The assignment of roles defines the structure of the coordination graph: interceptors, passers, and receivers are connected, whereas passive agents do not need to be considered in the joint-action selection process. The roles also determine the possible actions for each agent, which are discrete, high-level actions such as passing a ball to another agent in a given direction. The state variables are also defined as a high-level abstraction of the continuous game state; for example, there is a variable pass-blocked(i, j, d) that indicates whether a pass from agent i to agent j in direction d is blocked by an opponent. With this symbolic representation, one can write value rules that summarize the value gained by a particular combination of actions. For example, one rule says:

$$[\textit{has-role-receiver}(j) \wedge \neg \textit{isPassBlocked}(i, j, d) \wedge A_i = \textit{passTo}(j, d) \wedge A_j = \textit{moveTo}(d) : V(j, d)]$$

where $V(j, d)$ depends on the position where the receiving agent j receive the pass — the closer to the opponent goal the better.

A representation of a utility function as a set of rules is equivalent to a feature-based representation of a Markov network. To perform the optimization efficiently using this representation, we can easily adapt the rule-based variable-elimination scheme described in section 9.6.2.1. Note that the actual rules used in the inference are considerably simpler, since they are conditioned on the state variables, which include the role assignment of the agents and the other aspects of the state (such as isPassBlocked). However, this requirement introduces other complications: because of the limited communication bandwidth, each agent needs to solve the coordination graph on its own. Moreover, the state of the world is not generally fully observed to the agent; thus, one needs to ensure

that the agents take the necessary observation actions (such as turning the neck) to obtain enough information to condition the relevant state variables. Depending on the number of agents and their action space, one can now solve this problem using either variable elimination or belief propagation.

The coordination graph framework allows the different agents in the team to conduct complex maneuvers, an agent j would move to receive a pass from agent i even before agent i was in position to kick the ball; by contrast, previous methods required j to observe the trajectory of the ball before being able to act accordingly. This approach greatly increased the capabilities of the UVA Trilearn team. Whereas their entry took fourth place in the RoboSoccer 2002 competition, in 2003 it took first place among the forty-six qualifying team, with a total goal count of 177–7.

23.6 Ignoring Irrelevant Information ⋆

As we saw, there are several significant advantages to reducing the amount of information that the agent considers at each decision. Eliminating an information edge from a variable W into a decision variable D reduces the complexity of its decision rule, and hence the cognitive load on the decision maker. Computationally, it decreases the cost of manipulating its factor and of computing the decision rule. In this section, we consider a procedure for removing information edges from an influence diagram.

Of course, removing information edges reduces the agent's strategy space, and therefore can potentially significantly decrease his maximum expected utility value. If we want to preserve the agent's MEU value, we need to remove information edges with care. We focus here on removing only information edges that do not reduce the agent's MEU. We therefore study when a variable $W \in \mathrm{Pa}_D$ is irrelevant to making the optimal decision at D. In this section, we provide a graphical criterion for guaranteeing that W is irrelevant and can be dropped without penalty from the set Pa_D.

Intuitively, W is not relevant when it has no effect on utility nodes that participate in determining the decision at D.

Example 23.23

Consider the influence diagram $\mathcal{I}_S$ of figure 23.10. Intuitively, the edge from Difficulty (D) to Apply (A) is irrelevant. To understand why, consider its effect on the different utility variables in the network. On one hand, it influences V_S; however, given the variable Grade, which is also observed at A, D is irrelevant to V_S. On the other hand, it influences V_Q; however, V_Q cannot be influenced by the decision at A, and hence is not considered by the decision maker when determining the strategy at A. Overall, D is irrelevant to A given A's other parents. ■

We can make this intuition precise as follows:

Definition 23.10

irrelevant information edge

An information edge $W \rightarrow D$ from a (chance or decision) variable W is irrelevant *for a decision variable D if there is no active trail from W to $\mathcal{U}_{\succ D}$ given $\mathrm{Pa}_D - \{W\}$.* ■

According to this criterion, D is irrelevant for A, supporting our intuitive argument. We note that certain recall edges can also be irrelevant according to this definition. For example, assume that we add an edge from *Difficulty* to the decision variable *Take*. The *Difficulty* $\rightarrow$ *Take* edge

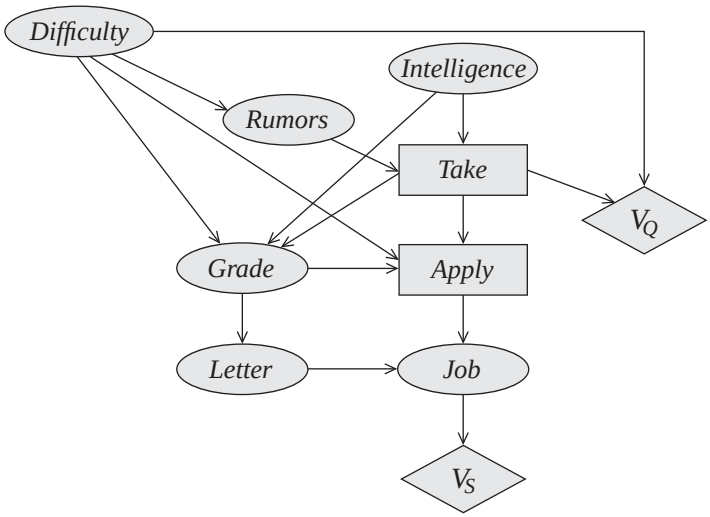


Figure 23.10 More complex influence diagram $\mathcal{I}_S$ for the Student **scenario.** Recall edges that follow from the definition are omitted for clarity.

is not irrelevant, but the *Difficulty* $\rightarrow$ *Apply* edge, which would be implied by perfect recall, is irrelevant.

We can show that irrelevant edges can be removed without penalty from the network.

Proposition 23.4 *Let $\mathcal{I}$ be an influence diagram, and $W \rightarrow D$ an irrelevant edge in $\mathcal{I}$. Let $\mathcal{I}'$ be the influence diagram obtained by removing the edge $W \rightarrow D$. Then for any strategy σ in $\mathcal{I}$, there exists a strategy σ' in $\mathcal{I}'$ such that* $\mathrm{EU}[\mathcal{I}'[\sigma']] \geq \mathrm{EU}[\mathcal{I}[\sigma]]$.

The proof follows from proposition 23.3 and is left as an exercise (exercise 23.11).

An influence diagram $\mathcal{I}'$ that is obtained from $\mathcal{I}$ via the removal of irrelevant edges is called a *reduction* of $\mathcal{I}$. An immediate consequence of proposition 23.4 is the following result:

reduction

Theorem 23.6 *If $\mathcal{I}'$ is a reduction of $\mathcal{I}$, then any strategy σ that is optimal for $\mathcal{I}'$ is also optimal for $\mathcal{I}$.*

The more edges we remove from $\mathcal{I}$, the simpler our computational problem. We would thus like to find a reduction that has the fewest possible edges. One simple method for obtaining a minimal reduction — one that does not admit the removal of any additional edges — is to remove irrelevant edges iteratively from the network one at a time until no further edges can be removed. An obvious question is whether the order in which edges are removed makes a difference to the final result. Fortunately, the following result implies otherwise:

Theorem 23.7 *Let $\mathcal{I}$ be an influence diagram and $\mathcal{I}'$ be any reduction of it. An arc $W \rightarrow D$ in $\mathcal{I}'$ is irrelevant in $\mathcal{I}'$ if and only if it is irrelevant in $\mathcal{I}$.*

The proof follows from properties of d-separation, and it is left as an exercise (exercise 23.12).

This theorem implies that we can examine each edge independently, and test whether it is irrelevant in $\mathcal{I}$. All such edges can then be removed at once. Thus, we can find all irrelevant edges using a single global computation of d-separation on the original ID.

The removal of irrelevant edges has several important computational benefits. First, it decreases the size of the strategy representation in the ID. Second, by removing edges in the network, it can reduce the complexity of the variable-elimination-based algorithms described in section 23.5. Finally, as we now show, it also has the effect of removing edges from the relevance graph associated with the ID. By breaking cycles in the relevance graph, it allows more decision rules to be optimized in sequence, reducing the need for iterations or for jointly optimizing the decision rules at multiple variables.

Proposition 23.5 *If $\mathcal{I}'$ is a reduction of $\mathcal{I}$, then the relevance graph of $\mathcal{I}'$ is a subset (not necessarily strict) of the relevance graph of $\mathcal{I}$.*

PROOF It suffices to show the result for the case where $\mathcal{I}'$ is a reduction of $\mathcal{I}$ by a single irrelevant edge. We will show that if D' is not s-reachable from D in $\mathcal{I}$, then it is also not s-reachable from D in $\mathcal{I}'$. If D' is s-reachable from D in $\mathcal{I}'$, then for a dummy parent $\widehat{D'}$, we have that there is some $V \in \mathcal{U}_{\succ D}$ and an active trail in $\mathcal{I}'$ from $\widehat{D}$ to V given $D, \mathrm{Pa}_D^{\mathcal{I}'}$. By assumption, that same trail is not active in $\mathcal{I}$. Since removal of edges cannot make a trail active, this situation can occur only if $\mathrm{Pa}_D^{\mathcal{I}'} = \mathrm{Pa}_D^{\mathcal{I}} - \{W\}$, and W blocks the trail from $\widehat{D'}$ to V in $\mathcal{I}$. Because observing W blocks the trail, it must be part of the trail, in which case there is a subtrail from W to V in $\mathcal{I}$. This subtrail is active given $(\mathrm{Pa}_D^{\mathcal{I}} - \{W\}), D$. However, observing D cannot activate a trail where we condition on D's parents (because then v-structures involving D are blocked). Thus, this subtrail must form an active trail from W to V given $\mathrm{Pa}_D^{\mathcal{I}} - \{W\}$, violating the assumption that $W \to D$ is an irrelevant edge. ■

23.7 Value of Information

So far, we have focused on the problem of decision making. Influence diagrams provide us with a representation for structured decision problems, and a basis for efficient decision-making algorithms.

One particularly useful type of task, which arises in a broad range of applications, is that of determining which variables we want to observe. Most obviously, in any diagnostic task, we usually have a choice of different tests we can perform. Because tests usually come at a cost (whether monetary or otherwise), we want to select the tests that are most useful in our particular setting. For example, in a medical setting, a diagnostic test such as a biopsy may involve significant pain to the patient and risk of serious injury, as well as high monetary costs. In other settings, we may be interested in determining if and where it is worthwhile to place sensors — such as a thermostat or a smoke alarm — so as to provide the most useful information in case of a fire.

☞ **The decision-theoretic framework provides us with a simple and elegant measure for the value of making a particular observation. Moreover, the influence diagram representation allows us to formulate this measure using a simple, graph-based criterion, which also provides considerable intuition.**

23.7.1 Single Observations

We begin with the question of evaluating the benefit of a single observation. In the setting of influence diagrams, we can model this question as one of computing the value of observing the value of some variable. Our *Survey* variable in the Entrepreneur example is precisely such a situation. Although we could (and did) analyze this type of decision using our general framework, it is useful to consider such decisions as a separate (and simpler) class. By doing so, we can gain insight into questions such as these.

The key idea is that the benefit of making an observation is the utility the agent can gain by observing the associated variable, assuming he acts optimally in both settings.

Example 23.24

Let us revisit the Entrepreneur *example, and consider the value to the entrepreneur of conducting the survey, that is, of observing the value of the Survey variable. In effect, we are comparing two scenarios and the utility to the entrepreneur in each of them: One where he conducts the survey, and one where he does not. If the agent does not observe the S variable, that node is barren in the network, and it can therefore be simply eliminated. This would result precisely in the influence diagram of figure 23.2. In example 22.3 we analyzed the agent's optimal action in this setting and showed that his MEU is 2. The second case is one in which the agent conducts the survey. This situation is equivalent to the influence diagram of figure 23.3, where we restrict to strategies where $C = c^1$. As we have already discussed, $C = c^1$ is the optimal strategy in this setting, so that the optimal utility obtainable by the agent in this situation is 3.22, as computed in example 23.6. Hence, the improvement in the entrepreneur's utility, assuming he acts optimally in both cases, is 1.22.* ■

More generally, we define:

Definition 23.11

value of perfect information

Let $\mathcal{I}$ be an influence diagram, X a chance variable, and D a decision variable such that there is no (causal) path from D to X. Let $\mathcal{I}'$ be the same as $\mathcal{I}$, except that we add an information edge from X to D, and to all decisions that follow D (that is, we have perfect information about X from D onwards). The value of perfect information *for X at D, denoted $\mathrm{VPI}_{\mathcal{I}}(D \mid X)$, is the difference between the MEU of $\mathcal{I}'$ and the MEU of $\mathcal{I}$.* ■

Let us analyze the concept of value of perfect information. First, it is not difficult to see that it cannot be negative; if the information is free, it cannot hurt to have it.

Proposition 23.6

Let $\mathcal{I}$ be an influence diagram, D a decision variable in $\mathcal{I}$, and X a chance variable that is a nondescendant of D. Let σ^* be the optimal strategy in $\mathcal{I}$. Then $\mathrm{VPI}_{\mathcal{I}}(D \mid X) \geq 0$, and equality holds if and only if σ^* is still optimal in the new influence diagram with X as a parent of D.

The proof is left as an exercise (exercise 23.13).

Does information always help? What if the numbers had been such that the entrepreneur would have founded the company regardless of the survey? In that case, the expected utility with the survey and without it would have been identical; that is, the VPI of S would have been zero. This property is an important one: **there is no value to information if it does not change the selected action(s) in the optimal strategy.**

Let us analyze more generally when information helps. To do that, consider a different decision problem.

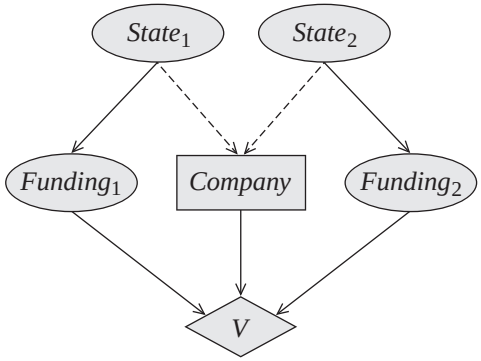


Figure 23.11 Influence diagram for VPI computation in example 23.25. We can compute the value of information for each of the two *State* variables by comparing the value with/without the dashed information edges.

Example 23.25

Our budding entrepreneur has decided that founding a startup is not for him. He is now choosing between two job opportunities at existing companies. Both positions are offering a similar starting salary, so his utility depends on his salary a year down the road, which depends on whether the company is still doing well at that point. The agent has the option of obtaining some information about the current state of the companies.

More formally, the entrepreneur has a decision variable C, whose value c_i is accepting a job with company i ($i = 1, 2$). For each company, we have a variable S_i that represents the current state of the company (quality of the management, the engineering team, and so on); this value takes three values, with s_i^3 being a very high-quality company and s_i^1 a poor company. We also have a binary-valued variable F_i, which represents the funding status of the company in the future, with f_i^1 representing the state of having funding. We assume that the utility of the agent is 1 if he takes a job with a company for which $F_i = f_i^1$ and 0 otherwise. We want to evaluate the value of information of observing S_1 (the case of observing S_2 is essentially the same). The structure of the influence diagram is shown in figure 23.11; the edges that would be added to compute the value of information are shown as dashed.

We now consider three different scenarios and compute the value of information in each of them.

Scenario 1: *Company 1 is well established, whereas Company 2 is a small startup. Thus, $P(S_1) = (0.1, 0.2, 0.7)$ (that is, $P(s_1^1) = 0.1$), and $P(S_2) = (0.4, 0.5, 0.1)$. The economic climate is poor, so the chances of getting funding are not great. Thus, for both companies, $P(f_i^1 \mid S_i) = (0.1, 0.4, 0.9)$ (that is, $P(f_i^1 \mid s_1^1) = 0.1$). Without additional information, the optimal strategy is c_1, with MEU value 0.72. Intuitively, in this case, the information obtained by observing S_1 does not have high value. Although it is possible that c_1 will prove less reliable than c_2, this outcome is very unlikely; with very high probability, c_1 will turn out to be the better choice even with the information. Thus, the probability that the information changes our decision is low, and the value of the information is also low. More formally, a simple calculation shows that the optimal strategy changes to c_2 only if we observe s_1^1, which happens with probability 0.1. The MEU value in this scenario is 0.743, which is not a significant improvement over our original MEU value. If observing*

S_1 costs more than 0.023 utility points, the agent should not make the observation.

Scenario 2: *The economic climate is still bad, but now c_1 and c_2 are both small startups. In this case, we might have $P(S_2)$ as in Scenario 1, and $P(S_1) = (0.3, 0.4, 0.3)$; $P(F_i \mid S_i)$ is also as in Scenario 1. Intuitively, our value of information in this case is quite high. There is a reasonably high probability that the observation will change our decision, and therefore a high probability that we would gain a lot of utility by finding out more information and making a better decision. Indeed, if we go through the calculation, the MEU strategy in the case without the additional observation is c_1, and the MEU value is 0.546. However, with the observation, we change our decision to c_2 both when $S_1 = s_1^1$ and when $S_1 = s_1^2$, events that are fairly probable. The MEU value in this case is 0.6882, a significant increase over the uninformed MEU.*

Scenario 3: *In this case, c_1 and c_2 are still both small startups, but the time is the middle of the Internet boom, so both companies are likely to be funded by investors desperate to get into this area. Formally, $P(S_1)$ and $P(S_2)$ are as above, but $P(f_i^1 \mid S_i) = (0.6, 0.8, 0.99)$. In this case, the probability that the observation changes the agent's decision is reasonably high, but the change to the agent's expected utility when the decision changes is low. Specifically, the uninformed optimal strategy is c_1, with MEU value 0.816. Observing s_1^1 changes the decision to c_2; but, while this observation occurs with probability 0.3, the difference in the expected utility between the two decisions in this case is less than 0.2. Overall, the MEU of the informed case is 0.8751, which is not much greater than the uninformed MEU value.* ■

Overall, we see that our definition of VPI allows us to make fairly subtle trade-offs.

The value of information is critical in many applications. For example, in medical or fault diagnosis, it often serves to tell us which diagnostic tests to perform (see box 23.C). Note that its behavior is exactly appropriate in this case. We do not want to perform a test just because it will help us narrow down the probability of the problem. We want to perform tests that will change our diagnosis. For example, if we have an invasive, painful test that will tell us which type of flu a patient has, but knowing that does not change our treatment plan (lie in bed and drink a lot of fluids), there is no point in performing the test.

23.7.2 Multiple Observations

We now turn to the more complex setting where we can make multiple simultaneous observations. In this case, we must decide which subset of the m potentially observable variables we choose to observe. For each such subset, we can evaluate the MEU value with the observations, as in the single variable case, and select the subset whose MEU value is highest. However, this approach is overly simplistic in several ways. First, the number of possible subsets of observations is exponentially large (2^m). A doctor, for example, might have available a large number of tests that she can perform, so the number of possible subsets of tests that she might select is huge. Even if we place a bound on the number of observations that can be performed or on the total cost of these observations, the number of possibilities can be very large.

More importantly, in practice, we often do not select in advance a set of observations to be performed, and then perform all of them at once. Rather, observations are typically made in sequence, so that the choice of which variable to observe next can be made with knowledge about the outcome of the previous observations. In general, the value of an observation can depend strongly on the outcome of a previous one. For example, in example 23.25, if we observe that the current state of Company 1 is excellent — $S_1 = s_1^3$, observing the state of Company 2

is significantly less useful than in a situation where we observe that $S_1 = s_1^2$. Thus, the optimal choice of variable to observe generally depends on the outcomes of the previous observations.

Therefore, when we have the ability to select a sequence of observations, the optimal selection has the form of a *conditional plan*: Start by observing X_1; if we observe $X_1 = x_1^1$, observe X_2; if we observe $X_1 = x_1^2$, observe X_3; and so on. Each such plan is exponentially large in the number k of possible observations that we are allowed to perform. The total number of such plans is therefore doubly exponential. Selecting an optimal observation plan is computationally a very difficult task, for which no good algorithms exist in general.

myopic value of information

The most common solution to this problem is to approximate the solution using *myopic value of information*, where we incrementally select at each stage the optimal single observation, ignoring its effect on the later choices that we will have to make. The optimal single observation can be selected easily using the methods described in the previous section. This myopic approximation can be highly suboptimal. For example, we might have an observation that, by itself, provides very little information useful for our decision, but does tell us which of two other observations is the most useful one to make.

In situations where the myopic approximation is complex, we can try to generate a conditional plan, as described before. One approach for solving such a problem is to formulate it as an influence diagram, with explicit decisions for which variable to observe. This type of transformation is essentially the one underlying the very simple case of example 23.3, where the variable C represents our decision on whether to observe S or not. The optimal strategy for this extended influence diagram also specifies the optimal observation plan. However, the resulting influence diagram can be quite complex, and finding an optimal strategy for it can be very expensive and often infeasible.

Box 23.C — Case Study: Decision Making for Troubleshooting. *One of the most commonly used applications of Bayesian network technology is to the task of fault diagnosis and repair. Here, we construct a probabilistic model of the device in question, where random variables correspond to different faults and different types of observations about the device state. Actions in this type of domain correspond both to diagnostic tests that can help indicate where the problem lies, and to actions that repair or replace a broken component. Both types of actions have a cost. One can now apply decision-theoretic techniques to help select a sequence of observation and repair actions.*

One of the earliest and largest fielded applications of this type was the decision-theoretic troubleshooting system incorporated into the Microsoft's Windows 95™ operating system. The system, described in Heckerman, Breese, and Rommelse (1995) and Breese and Heckerman (1996), included hundreds of Bayesian networks, each aimed at troubleshooting a type of fault that commonly arises in the system (for example, a failure in printing, or an application that does not launch). Each fault had its own Bayesian network model, ranging in size from a few dozen to a few hundred variables. To compute the probabilities required for an analysis involving repair actions, which intervene in the model, one must take into account the fact that the system state from before the repair also persists afterward (except for the component that was repaired). For this computation, a counterfactual twinned network *model was used, as described in box 21.C.*

counterfactual twinned network

The probabilistic models were augmented with utility models for observing the state of a component in the system (that is, whether it is faulty) and for replacing it. Under carefully crafted assumptions (such as a single fault hypothesis), it was possible to define an optimal series of re-

pair/observation actions, given a current state of information $\boldsymbol{e}$, and thereby compute an exact formula for the expected cost of repair *ECR($\boldsymbol{e}$). (See exercise 23.15.) This formula could then be used to compute exactly the benefit of any diagnostic test D, using a standard value of information computation:*

$$\sum_{d \in Val(D)} P(D = d \mid \boldsymbol{e}) ECR(\boldsymbol{e}, D = d).$$

One can then add the cost of the observation of D to choose the optimal diagnostic test. Note that the computation of ECR($\boldsymbol{e}, D = d$) estimates the cost of the full trajectory of repair actions following the observation, a trajectory that is generally different for different values of the observation d. Thus, although this analysis is still myopic in considering only a single observation action D at a time, it is nonmyopic in evaluating the cost of the plan of action following the observation.

Empirical results showed that this technique was very valuable. One test, for example, was applied to the printer diagnosis network of box 5.A. Here, the cost was measured in terms of minutes to repair. In synthetic cases with known failures, sampled from the network, the system saved about 20 percent of the time over the best predetermined plan. Interestingly, the system also performed well, providing equal or even better savings, in cases where there were multiple faults, violating the assumptions of the model.

At a higher level, decision-theoretic techniques are particularly valuable in this setting for several reasons. The standard system used up to that point was a standard static flowchart where the answer to a question would lead to different places in the flowchart. From the user side, the experience was significantly improved in the decision-theoretic system, since there was considerably greater flexibility: diagnostic tests are simply treated as observed variables in the network, so if a user chooses not to answer a question at a particular point, the system can still proceed with other questions or tests. Users also felt that the questions they were asked were intuitive and made sense in context. Finally, there was also significant benefit for the designer of the system, because the decision-theoretic system allowed modular and easily adaptable design. For example, if the system design changes slightly, the changes to the corresponding probabilistic models are usually small (a few CPDs may change, or maybe some variables are added/deleted); but the changes to the "optimal" flowchart are generally quite drastic. Thus, from a software engineering perspective, this approach was also very beneficial.

This application is one of the best-known examples of decision-theoretic troubleshooting, but similar techniques have been successfully used in a large number of applications, including in a decision-support system for car repair shops, in tools for printer and copier repair, and many others.

23.8 Summary

In this chapter, we placed the task of decision making using decision-theoretic principles within the graphical modeling framework that underlies this entire book. Whereas a purely probabilistic graphical model provides a factorized description of the probability distribution over possible states of the world, an influence diagram provides such a factorized representation for the agent's actions and utility function as well. The influence diagram clearly encodes the

breakdown of these three components of the decision-making situation into variables, as well as the interactions between these variables. These interactions are both probabilistic, where one variable affects the distribution of another, and informational, where observing a variable allows an agent to actively change his action (or his decision rule).

We showed that dynamic programming algorithms, similar to the ones used for pure probabilistic inference, can be used to find an optimal strategy for the agent in an influence diagram. However, as we saw, inference in an influence diagram is more complex than in a Bayesian network, both conceptually and computationally. This complexity is due to the interactions between the different operations involved: products for defining the probability distribution; summation for aggregating utility variables; and maximization for determining the agent's optimal actions.

The influence diagram representation provides a compact encoding of rich and complex decision problems involving multiple interrelated factors. It provides an elegant framework for considering such important issues as which observations are required to make optimal decisions, the definition of recall and the value of the perfect recall assumption, the dependence of a particular decision on particular observations or components of the agent's utility function, and the like. Value of information — a concept that plays a key role in many practical applications — is particularly easy to capture in the influence diagram framework.

However, there are several factors that can cause the complexity of the influence diagram to grow unreasonably large, and significantly reduce its usability in many real-world settings. One such limitation is the perfect recall assumption, which can lead the decision rules to grow exponentially large in the number of actions and observations the agent makes. We note that this limitation is not one of the representation, but rather of the requirements imposed by the notion of optimality and by the algorithms we use to find solutions. A second source of blowup arises when the scenario that arises following one decision by the agent is very different from the scenario following another. For example, imagine that the agent has to decide whether to go from San Francisco to Los Angeles by air or by car. The subsequent decisions he has to make and the variables he may observe in these two cases are likely to be very different. This example is an instance of context-specificity, as described in section 5.2.2; however, the simple solution of modifying our CPD structure to account for context-specificity is usually insufficient to capture compactly these very broad changes in the model structure. The decision-tree structure is better able to capture this type of structure, but it too has its limitations; several works have tried to combine the benefits of both representations (see section 23.9).

Finally, the basic formalism for sequential decision making under uncertainty is only a first step toward a more general formalism for planning and acting under uncertainty in many settings: single-agent, multiagent distributed decision making, and multiagent strategic (game-theoretic) interactions. A complete discussion of the ideas and methods in any of these areas is a book in itself; we encourage the reader who is interested in these topics to pursue some additional readings, some of which are mentioned in section 23.9.

23.9 Relevant Literature

The influence diagram representation was introduced by Howard and Matheson (1984a), albeit more as a guide to formulating a decision problem than as a formal language with well-defined semantics. See also Oliver and Smith (1990) for an overview.

Olmsted (1983) and Shachter (1986, 1988) provided the first algorithm for decision making in influence diagrams, using local network transformations such as edge reversal. This algorithm was gradually improved and refined over the years in a series of papers (Tatman and Shachter 1990; Shenoy 1992; Shachter and Ndilikilikesha 1993; Ndilikilikesha 1994). The most recent algorithm of this type is due to Jensen, Jensen, and Dittmer (1994); their algorithm utilizes the clique tree data structure for addressing this task. All of these solutions use a constrained elimination ordering, and they are therefore generally feasible only for fairly small influence diagrams.

A somewhat different approach is based on reducing the problem of solving an influence diagram to inference in a standard Bayesian network. The first algorithm along these lines is due to Cooper (1988), whose approach applied only to a single decision variable. This idea was subsequently extended and improved considerably by Shachter and Peot (1992) and Zhang (1998).

Nilsson and Lauritzen (2000) and Lauritzen and Nilsson (2001) provide an algorithm based on the concept of limited memory influence diagrams, which relaxes the perfect recall assumption made in almost all previous work. This relaxation allows them to avoid the constraints on the elimination ordering, and thereby leads to a much more efficient clique tree algorithm. Similar ideas were also developed independently by Koller and Milch (2001). The clique-tree approach was further improved by Madsen and Nilsson (2001).

The simple influence diagram framework poses many restrictions on the type of decision-making situation that can be expressed naturally. Key restrictions include the perfect recall assumption (also called "no forgetting"), and the assumption of the uniformity of the paths that traverse the influence diagram.

Regarding this second point, a key limitation of the basic influence diagram representation is that it is designed for encoding situations where all trajectories through the system go through the same set of decisions in the same fixed order. Several authors (Qi et al. 1994; Covaliu and Oliver 1995; Smith et al. 1993; Shenoy 2000; Nielsen and Jensen 2000) propose extensions that deal with asymmetric decision settings, where a choice taken at one decision variable can lead to different decision being encountered later on. Some approaches (Smith et al. 1993; Shenoy 2000) use an approach based on context-specific independence, along the lines of the tree-CPDs of section 5.3. These approaches are restricted to cases where the sequence of observations and decisions is fixed in all trajectories of the system. The approach of Nielsen and Jensen (1999, 2000) circumvents this limitation, allowing for a partial ordering over observations and decisions. The partial ordering allows them to reduce the set of constraints on the elimination ordering in a variable elimination algorithm, resulting in computational savings. This approach was later extended by Jensen and Vomlelová (2003).

In a somewhat related trajectory, Shachter (1998, 1999) notes that some parents of a decision node may be irrelevant for constructing the optimal decision rule, and provided a graphical procedure, based on his BayesBall algorithm, for identifying such irrelevant chance nodes. The LIMID framework of Nilsson and Lauritzen (2000); Lauritzen and Nilsson (2001) makes these notions more explicit by specifically encoding in the influence diagram representation the subset of potentially observable variables relevant to each decision. This allows a relaxation of the ordering constraints induced by the perfect recall assumption. They also define a graphical procedure for identifying which decision rules depend on which others. This approach forms the basis for the recursive algorithm presented in this chapter, and for its efficient implementation using clique trees.

The concept of value of information was first defined by Howard (1966). Over the years, various

algorithms (Zhang et al. 1993; Chávez and Henrion 1994; Ezawa 1994) have been proposed for performing value of information computations efficiently in an influence diagram, culminating in the work of Dittmer and Jensen (1997) and Shachter (1999). All of these papers focus on the myopic case and provide an algorithm for computing the value of information only for all single variables in this network (allowing the decision maker to decide which one is best to observe). Recent work of Krause and Guestrin (2005a,b) addresses the nonmyopic problem of selecting an entire sequence of observations to make, within the context of a particular class of utility functions.

There have been several fielded systems that use the decision-theoretic approach described in this chapter, although many use a Bayesian network and a simple utility function rather than a full-fledged influence diagram. Examples of this latter type include the Pathfinder system of Heckerman (1990); Heckerman et al. (1992), and Microsoft's system for decision-theoretic troubleshooting (Heckerman et al. 1995; Breese and Heckerman 1996) that was described in box 23.C. The Vista system of Horvitz and Barry (1995) used an influence diagram to make decisions on display of information at NASA Mission Control Center. Norman et al. (1998) present an influence-diagram system for prenatal testing, as described in box 23.A. Meyer et al. (2004) present a fielded application of an influence diagram for selecting radiation therapy plans for prostate cancer.

Markov decision process

A framework closely related to influence diagrams is that of *Markov decision processes* (MDPs) and its extension to the *partially observable* case (partially observable Markov decision processes, or POMDPs). The formal foundations for this framework were set forth by Bellman (1957); Bertsekas and Tsitsiklis (1996) and Puterman (1994) provide an excellent modern introduction to this topic. Although both an MDP and an influence diagram encode decision problems, the focus of influence diagrams has been on richly spaces that involve rich structure in terms of the state description (and sometimes the utility function), but only a few decisions; conversely, much of the focus of MDPs has been on state spaces that are fairly unstructured (encoded simply as a set of states), but on complex decision settings with long (often infinite) sequences of decisions.

Several groups have worked on the synthesis of these two fields, tackling the problem of sequential decision making in large, richly structured state spaces. Boutilier et al. (1989, 2000) were the first to explore this extension; they used a DBN representation of the MDP, and relied on the use of context-specific structure both in the system dynamics (tree-CPDs) and in the form of the value function. Boutilier, Dean, and Hanks (1999) provide a comprehensive survey of the representational issues and of some of the earlier algorithms in this area. Koller and Parr (1999); Guestrin et al. (2003) were the first to propose the use of factored value functions, which decompose additively as a sum of subutility functions with small scope. Building on the rule-based variable elimination approach described in section 9.6.2.1, they also show how to make use of both context-specific structure and factorization.

Another interesting extension that we did not discuss is the problem of decision making in multiagent systems. At a high level, one can consider two different types of multiagent systems: ones where the agents share a utility function, and need to cooperate in a decentralized setting with limited communication; and ones where different agents have different utility functions, and must optimize their own utility while accounting for the other agents' actions. Guestrin et al. (2003) present some results for the cooperative case, and introduce the notion of coordination graph; they focus on issues that arise within the context of MDPs, but some of their ideas can also be applied to influence diagrams. The coordination graph structure was the basis for the

RoboSoccer application of Kok et al. (2003); Kok and Vlassis (2005), described in box 23.B.

game theory

The problem of optimal decision making in the presence of strategic interactions is the focus of most of the work in the field of *game theory*. In this setting, the notion of a "rational strategy" is somewhat more murky, since what is optimal for one player depends on the actions taken by others. Fudenberg and Tirole (1991) and Osborne and Rubinstein (1994) provide a good introduction to the field of game theory, and to the standard solution concepts used. Generally, work in game theory has represented multiagent interactions in a highly unstructured way: either in the *normal form*, which lists a large matrix indexed by all possible strategies of all agents, or in the *extensive form* — a game tree, a multiplayer version of a decision tree. More recently, there have been several proposals for game representations that build on ideas in graphical models. These proposals include graphical games (Kearns et al. 2001), multiagent influence diagrams (Koller and Milch 2003), and game networks (La Mura 2000). Subsequent work (Vickrey and Koller 2002; Blum et al. 2006) has shown that ideas similar to those used for inference in graphical models and influence diagrams can be used to provide efficient algorithms for finding Nash equilibria (or approximate Nash equilibria) in these structured game representations.

23.10 Exercises

Exercise 23.1

Show that the decision rule δ_D that maximizes: $\sum_{D,\mathrm{Pa}_D} \delta_D \mu_{-D}(D, \mathrm{Pa}_D)$ is defined as:

$$\delta_D(\boldsymbol{w}) = \arg\max_{d \in Val(D)} \mu_{-D}(d, \boldsymbol{w}) \quad \text{for all } \boldsymbol{w} \in Val(\mathrm{Pa}_D).$$

Exercise 23.2

Prove proposition 23.1. In particular:

a. Show that for γ^* defined as in equation (23.8), we have that $\phi^* = \prod_{W \in \mathcal{X} \cup \mathcal{D}} \phi_W$, $\mu^* = \prod_{V \in \mathcal{U}} \mu_V$.

b. For $\boldsymbol{W}' \subset \boldsymbol{W}$, show that

$$\mathrm{cont}(\mathit{marg}_{\boldsymbol{W}'}(\gamma)) = \sum_{\boldsymbol{W} - \boldsymbol{W}'} \mathrm{cont}(\gamma),$$

that is, that contraction and marginalization interchange appropriately.

c. Use your previous results to prove proposition 23.1.

Exercise 23.3⋆

Prove theorem 23.1 by showing that the combination and marginalization operations defined in equation (23.5) and equation (23.6) satisfy the axioms of exercise 9.19:

a. Commutativity and associativity of combination:

$$\begin{aligned} \gamma_1 \bigoplus \gamma_2 &= \gamma_2 \bigoplus \gamma_1 \\ \gamma_1 \bigoplus (\gamma_2 \bigoplus \gamma_3) &= (\gamma_1 \bigoplus \gamma_2) \bigoplus \gamma_3. \end{aligned}$$

b. Consonance of marginalization: Let γ be a factor over scope $\boldsymbol{W}$ and let $\boldsymbol{W}_2 \subseteq \boldsymbol{W}_1 \subseteq \boldsymbol{W}$. Then:

$$\mathit{marg}_{\boldsymbol{W}_2}(\mathit{marg}_{\boldsymbol{W}_1}(\gamma)) = \mathit{marg}_{\boldsymbol{W}_2}(\gamma).$$

c. Interchanging marginalization and combination: Let γ_1 and γ_2 be potentials over $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ respectively. Then:

$$\mathit{marg}_{\boldsymbol{W}_1}((\gamma_1 \bigoplus \gamma_2)) = \gamma_1 \bigoplus \mathit{marg}_{\boldsymbol{W}_1}(\gamma_2).$$

Exercise 23.4⋆

Prove lemma 23.1.

Exercise 23.5⋆⋆

Extend the variable elimination algorithm of section 23.3 to the case of multiple utility variables, using the mechanism of joint factors used in section 23.4.3. (Hint: Define an operation of max-marginalization, as required for optimizing a decision variable, for a joint factor.)

Exercise 23.6⋆

Prove proposition 23.3. (Hint: The proof is based on algebraic manipulation of the expected utility $\mathrm{EU}[\mathcal{I}[(\sigma_{-D}, \delta_D)]]$.)

Exercise 23.7

Prove theorem 23.4, as follows:

a. Show that if D_i and D_j are two decisions such that $D_i, \mathrm{Pa}_{D_i} \subseteq \mathrm{Pa}_{D_j}$, then D_i is not s-reachable from D_j.
b. Use this result to conclude the theorem.
c. Show that the nodes in the relevance graph in this case will be totally ordered, in the opposite order to the temporal ordering $\prec$ over the decisions in the influence diagram.

Exercise 23.8⋆

In this exercise, you will prove theorem 23.5, using two steps.

a. We first need to prove a result analogous to theorem 23.2, but showing that a decision rule δ_D remains optimal even if the decision rules at several decisions D' change.
   Let σ be a fully mixed strategy, and δ_D a decision rule for D that is locally optimal for σ. Let σ' be another strategy such that, whenever $\sigma'(D') \neq \sigma(D')$, then D' is not s-reachable from D. Prove that δ_D is also optimal for σ'.
b. Now, let σ^k be the strategy returned by Iterated-Optimization-for-IDs, and σ' be some other strategy for the agent. Let $D_1, \ldots, D_k$ be the ordering on decisions used by the algorithm. Show that $\mathrm{EU}[\mathcal{I}[\sigma^n]] \geq \mathrm{EU}[\mathcal{I}[\sigma']]$. (Hint: Use induction on the number of variables l at which σ^k and σ' differ.)

Exercise 23.9⋆⋆

Extend the algorithm of algorithm 23.3 to find a globally optimal solution even in influence diagrams with cyclic relevance graphs. Your algorithm will have to optimize several decision rules simultaneously, but it should not always optimize all decision rules simultaneously. Explain precisely how you jointly optimize multiple decision rules, and how you select the order in which decision rules are optimized.

Exercise 23.10⋆⋆

In this exercise, we will define an efficient clique tree implementation of the algorithm of algorithm 23.3.

a. Describe a clique tree algorithm for a setting where cliques and sepsets are each parameterized with a joint (probability, utility) potential, as described in section 23.4.3. Define: (i) the clique tree initialization in terms of the network parameterization and a complete strategy σ, and (ii) the message passing operations.

b. Show how we can use the clique-tree data structure to reuse computation between different steps of the iterated optimization algorithm. In particular, show how we can easily retract the current decision rule δ_D from the calibrated clique tree, compute a new optimal decision rule for D, and then update the clique tree accordingly. (Hint: Use the ideas of section 10.3.3.1.)

Exercise 23.11⋆

Prove proposition 23.4.

Exercise 23.12

Prove theorem 23.7: Let $\mathcal{I}$ be an influence diagram and $\mathcal{I}'$ be any reduction of it, and let $W \to D$ be some arc in $\mathcal{I}'$.

a. (easy) Prove that if $W \to D$ is irrelevant in $\mathcal{I}$, then it is also irrelevant in $\mathcal{I}'$.
b. (hard) Prove that if $W \to D$ is irrelevant in $\mathcal{I}'$, then it is also irrelevant in $\mathcal{I}$.

Exercise 23.13

a. Prove proposition 23.6.
b. Is the value of learning the values of two variables equal to the sum of the values of learning each of them? That is to say, is

$$\mathrm{VPI}(\mathcal{I}, D, \{X, Y\}) = \mathrm{VPI}(\mathcal{I}, D, X) + \mathrm{VPI}(\mathcal{I}, D, Y)?$$

Exercise 23.14⋆

Consider an influence diagram $\mathcal{I}$, and assume that we have computed the optimal strategy for $\mathcal{I}$ using the clique tree algorithm of section 23.5.2. Let D be some decision in D, and X some variable not observed at D in $\mathcal{I}$. Show how we can efficiently compute $\mathrm{VPI}_{\mathcal{I}}(D \mid X)$, using the results of our original clique tree computation, when:

a. D is the only decision variable in $\mathcal{I}$.
b. The influence diagram contains additional decision variables, but the relevance graph is acyclic.

Exercise 23.15⋆

Consider a setting where we have a faulty device. Assume that the failure can be caused by a failure in one of n components, exactly one of which is faulty. The probability that repairing component c_i will repair the device is p_i. By the single-fault hypothesis, we have that $\sum_{i=1}^{n} p_i = 1$. Further assume that each component c_i can be examined with cost C_i^o and then repaired (if faulty) with cost C_i^r. Finally, assume that the costs of observing and repairing any component do not depend on any previous actions taken.

a. Show that if we observe and repair components in the order $c_1, \ldots, c_n$, then the expected cost until the device is repaired is:

$$\sum_{i=1}^{n} \left[\left(1 - \sum_{j=1}^{i-1} p_j \right) C_i^o + p_i C_i^r \right].$$

b. Use that to show that the optimal sequence of actions is the one in which we repair components in order of their p_i / C_i^o ratio.
c. Extend your analysis to the case where some components can be replaced, but not observed; that is, we cannot determine whether they are broken or not.

Exercise 23.16

value of control

The value of perfect information measures the change in our MEU if we allow observing a variable that was not observed before. In the same spirit, define a notion of a *value of control*, which is the gain to the agent if she is allowed to intervene at a chance variable X and set its value. Make reasonable assumptions about the space of strategies available to the agent, but state your assumptions explicitly.

24 *Epilogue*

Why Probabilistic Graphical Models?

In this book, we have presented a framework of structured probabilistic models. This framework rests on two foundations:

- the use of a probabilistic model — a joint probability distribution — as a representation of our domain knowledge;
- the use of expressive data structures (such as graphs or trees) to encode structural properties of these distributions.

The first of these ideas has several important ramifications. First, **our domain knowledge is encoded *declaratively*, using a representation that has its own inherent semantics. Thus, the conclusions induced by the model are intrinsic to it, and not dependent on a specific implementation or algorithm. This property gives us the flexibility to develop a range of inference algorithms, which may be appropriate in different settings. As long as each algorithm remains faithful to the underlying model semantics, we know it to be correct.**

declarative representation

Moreover, because the basic operations of the calculus of probabilities (conditioning, marginalization) are generally well accepted as being sound reasoning patterns, we obtain an important guarantee: If we obtain surprising or undesirable conclusions from our probabilistic model, the problem is with our model, not with our basic formalism. Of course, this conclusion relies on the assumption that we are using exact probabilistic inference, which implements (albeit efficiently) the operations of this calculus; when we use approximate inference, errors induced by the algorithm may yield undesirable conclusions. Nevertheless, **the existence of a declarative representation allows us to separate out the two sources of error — modeling error and algorithmic error — and consider each separately.** We can ask separately whether our model is a correct reflection of our domain knowledge, and whether, for the model we have, approximate inference is introducing overly large errors. Although the answer to each of these questions may not be trivial to determine, each is more easily considered in isolation. For example, to test the model, we might try different queries or perform sensitivity analysis. To test an approximate inference algorithm, we might try the algorithm on fragments of the network, try a different (approximate or exact) inference algorithm, or compare the probability of the answer obtained to that of an answer we may expect.

A third benefit to the use of a declarative probabilistic representation is the fact that the same representation naturally and seamlessly supports multiple types of reasoning. We can

compute the posterior probability of any subset of variables given observations about any others, subsuming reasoning tasks such as prediction, explanation, and more. We can compute the most likely joint assignment to all of the variables in the domain, providing a solution to a problem known as *abduction*. With a few extensions to the basic model, we can also answer causal queries and make optimal decisions under uncertainty.

abduction

The second of the two ideas is the key to making probabilistic inference practical. **The ability to exploit structure in the distribution is the basis for providing a compact representation of high-dimensional (or even infinite-dimensional) probability spaces. This compact representation is highly modular, allowing a flexible representation of domain knowledge that can easily be adapted, whether by a human expert or by an automated algorithm.** This property is one of the key reasons for the use of probabilistic models. For example, as we discussed in box 23.C, a diagnostic system designed by a human expert to go through a certain set of menus asking questions is very brittle: even small changes to the domain knowledge can lead to a complete reconstruction of the menu system. By contrast, a system that uses inference relative to an underlying probabilistic model can easily be modified simply by revising the model (or small parts of it); these changes automatically give rise to a new interaction with the user.

The compact representation is also the key for the construction of effective reasoning algorithms. All of the inference algorithms we discussed exploit the structure of the graph in fundamental ways to make the inference feasible. Finally, the graphical representation also provides the basis for learning these models from data. First, the smaller parameter space utilized by these models allows parameter estimation even of high-dimensional distributions from a reasonable amount of data. Second, the space of sparse graph structures defines an effective and natural bias for structure learning, owing to the ubiquity of (approximate) conditional independence properties in distributions arising in the real world.

The Modeling Pipeline

The framework of probabilistic graphical models provides support for natural representation, effective inference, and feasible model acquisition. Thus, it naturally leads to an integrated methodology for tackling a new application domain — a methodology that relies on all three of these components.

Consider a new task that we wish to address. We first define a class of models that encode the key properties of the domain that are critical to the task. We then use learning to fill in the missing details of the model. The learned model can be used as the basis for knowledge discovery, with the learned structure and parameters providing important insights about properties of the domain; it can also be used for a variety of reasoning tasks: diagnosis, prediction, or decision making.

Many important design decisions must be made during this process. One is the form of the graphical model. We have described multiple representations throughout this book — directed and undirected, static and temporal, fixed or template-based, with a variety of models for local interactions, and so forth. These should not be considered as mutually exclusive options, but rather as useful building blocks. Thus, a model does not have to be either a Bayesian network or a Markov network — perhaps it should have elements of both. A model may be neither a full dynamic Bayesian network nor a static one: perhaps some parts of the system can be modeled

as static, and others as dynamic.

In another decision, when designing our class of models, we can provide a fairly specific description of the models we wish to consider, or one that is more abstract, specifying only high-level properties such as the set of observed variables. Our prior knowledge can be incorporated in a variety of ways: as hard constraints on the learned model, as a prior, or perhaps even only as an initialization for the learning algorithm. Different combinations will be appropriate for different applications.

These decisions, of course, influence the selection of our learning algorithm. In some cases, we will need to fill in only (some) parameters; in others, we can learn significant aspects of the model structure. In some cases, all of the variables will be known in advance; in others, we will need to infer the existence and role of hidden variables.

When designing a class of models, it is critical to keep in mind the basic trade-off between faithfulness — accurately modeling the variables and interactions in the domain — and identifiability — the ability to reliably determine the details of the model. Given the richness of the representations one can encode in the framework of probabilistic models, it is often very tempting to select a highly expressive representation, which really captures everything that we think is going on in the domain. Unfortunately, such models are often hard to identify from training data, owing both to the potential for overfitting and to the large number of local maxima that can make it difficult to find the optimal model (even when enough training data are available). Thus, one should always keep in mind Einstein's maxim:

Everything should be made as simple as possible, but not simpler.

There are many other design decisions that influence our learning algorithm. Most obviously, there are often multiple learning algorithms that are applicable to the same class of models. Other decisions include what priors to use, when and how to introduce hidden variables, which features to construct, how to initialize the model, and more. Finally, if our goal is to use the model for knowledge discovery, we must consider issues such as methods for evaluating our confidence in the learned model and its sensitivity to various choices that we made in the design. Currently, these decisions are primarily made using individual judgment and experience.

Finally, if we use the model for inference, we also have various decisions to make. For any class of models, there are multiple algorithms — both exact and approximate — that one can apply. Each of these algorithms works well in certain cases and not others. It is important to remember that here, too, we are not restricted to using only a pure version of one of the inference algorithms we described. We have already presented hybrid methods such as collapsed sampling methods, which combine exact inference and sampling. However, many other hybrids are possible and useful. For example, we might use collapsed particle methods combined with belief propagation rather than exact inference, or use a variational approximation to provide a better proposal distribution for MCMC methods.

Overall, it is important to realize that what we have provided is a set of ideas and tools. One can be flexible and combine them in different ways. Indeed, one can also extend these ideas, constructing new representations and algorithms that are based on these concepts. This is precisely the research endeavor in this field.

Some Current and Future Directions

Despite the rapid advances in this field, there are many directions in which significant open problems remain. Clearly, one cannot provide a comprehensive list of all of the interesting open problems; indeed, identifying an open problem is often the first step in a research project. However, we describe here some broad categories of problems where there is clearly much work that needs to be done.

On the pragmatic side, probabilistic models have been used as a key component in addressing some very challenging applications involving automated reasoning and decision making, data analysis, pattern recognition, and knowledge discovery. We have mentioned some of these applications in the case studies provided in this book, but there are many others to which this technology is being applied, and still many more to which it could be applied. There is much work to be done in further developing these methods in order to allow their effective application to an increasing range of real-world problems.

However, our ability to easily apply graphical models to solve a range of problems is limited by the fact that many aspects of their application are more of an art than a science. As we discussed, there are many important design decisions in the selection of the representation, the learning procedure, and the inference algorithm used. Unfortunately, there is no systematic procedure that one can apply in navigating these design spaces. Indeed, there is not even a comprehensive set of guidelines that tell us, for a particular application, which combination of ideas are likely to be useful. At the moment, the design process is more the result of trial-and-error experimentation, combined with some rough intuitions that practitioners learn by experience. It would be an important achievement to turn this process from a black art into a science.

At a higher level, one can ask whether the language of probabilistic graphical models is adequate for the range of problems that we eventually wish to address. Thus, a different direction is to extend the expressive power of probabilistic models to incorporate a richer range of concepts, such as multiple levels of abstractions, complex events and processes, groups of objects with a rich set of interactions between them, and more. If we wish to construct a representation of general world knowledge, and perhaps to solve truly hard problems such as perception, natural language understanding, or commonsense reasoning, we may need a representation that accommodates concepts such as these, as well as associated inference and learning algorithms. Notably, many of these issues were tackled, with varying degrees of success, within the disciplines of philosophy, psychology, linguistics, and traditional knowledge representation within artificial intelligence. Perhaps some of the ideas developed in this long-term effort can be integrated into a probabilistic framework, which also supports reasoning from limited observations and learning from data, providing an alternative starting point for this very long-term endeavor.

The possibility that these models can be used as the basis for solving problems that lie at the heart of human intelligence raises an entirely new and different question: Can we use models such as these as a tool for understanding human cognition? In other words, can these structured models, with their natural information flow over a network of concepts, and their ability to integrate intelligently multiple pieces of weak evidence, provide a good model for human cognitive processes? Some preliminary evidence on this question is promising, and it suggests that this direction is worthy of further study.

A Background Material

A.1 Information Theory

Information theory deals with questions involving efficient coding and transmission of information. To address these issues, one must consider how to encode information so as to maximize the amount of data that can sent on a given channel, and how to deal with noisy channels. We briefly touch on some technical definitions that arise in information theory, and use compression as our main motivation. Cover and Thomas (1991) provides an excellent introduction to information theory, including historical perspective on the development and applications of these notions.

A.1.1 Compression and Entropy

Suppose that one plans to transmit a large corpus of say English text over a digital line. One option is to send the text using standard (for example, ASCII) encoding that uses a fixed number of bits per character. A somewhat more efficient approach is to use a code that is tailored to the task of transmitting English text. For example, if we construct a dictionary of all words, we can use binary encoding to describe each word; using 16 bits per word, we can encode a dictionary of up to 65,536 words, which covers most English text.

compression

We can gain an additional boost in *compression* by building a *variable-length code*, which encodes different words in bit strings of different length. The intuition is that words that are frequent in English should be encoded by shorter code words, and rare words should be encoded by longer ones. To be unambiguously decodable, a variable-length code must be *prefix free*: no codeword can be a strict prefix of another. Without this property, we would not be able to tell (at least not using a simple scan of the data) when one code word ends and the next begins.

It turns out that variable-length codes can significantly improve our compression rate:

Example A.1

Assume that our dictionary contains four words — w_1, w_2, w_3, w_4 — with frequencies $P(w_1) = 1/2$, $P(w_2) = 1/4$, $P(w_3) = 1/8$, and $P(w_4) = 1/8$. One prefix-free encoding for this dictionary is to encode w_1 using a single bit codeword, say "0"; we would then encode w_2 using the 2-bit sequence "10", and w_3 and w_4 using three bits each "110" and "111".

Now, consider the expected number of bits that we would need for a message sent with this frequency distribution. We must encode the word w_1 on average half the time, and it costs us 1 bit. We must encode the word w_2 a quarter of the time, and it costs us 2 bits. Overall, we get that the

expected number of bits used is:

$$\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75.$$

■

One might ask whether a different encoding would give us better compression performance in this example. It turns out that this encoding is the best we can do, relative to the word-frequency distribution. To provide a formal analysis for this statement, suppose we have a random variable X that denotes the next item we need to encode (for example, a word). In order to analyze the performance of a compression scheme, we need to know the distribution over different values of X. So we assume that we have a distribution $P(X)$ (for example, frequencies of different words in a large corpus of English documents).

The notion of the entropy of a distribution provides us with a precise lower bound for the expected number of bits required to encode instances sampled from $P(X)$.

Definition A.1 entropy

Let $P(X)$ be a distribution over a random variable X. The entropy *of X is defined as*

$$\boldsymbol{H}_P(X) = \boldsymbol{E}_P\left[\log \frac{1}{P(x)}\right] = \sum_x P(x) \log \frac{1}{P(x)},$$

where we treat $0 \log 1/0 = 0$.[1]

■

When discussing entropies (and other information-theoretic measures) we use logarithms of base 2. We can then interpret the entropy in terms of bits.

The central result in information theory is a theorem by Shannon showing that the entropy of X is the lower bound on the average number of bits that are needed to encode values of X. That is, if we consider a proper codebook for values of X (one that can be decoded unambiguously), then the expected code length, relative to the distribution $P(X)$, cannot be less than $\boldsymbol{H}_P(X)$ bits.

Going back to our example, we see that the average number of bits for this code is precisely the entropy. Thus, the lower bound is tight in this case, in that we can construct a code that achieves precisely that bound. As another example, consider a uniform distribution $P(X)$. In this case, the optimal encoding is to represent each word using the same number of bits, $\log |Val(X)|$. Indeed, it is easy to verify that $\boldsymbol{H}_P(X) = \log |Val(X)|$, so again the bound is tight (at least for cases where $|Val(X)|$ is a power of 2.) Somewhat surprisingly, the entropy bound is tight in general, in that there are codes that come very close to the "optimum" of assigning the value x a code of length $-\log P(x)$.[2]

☞ **Another way of viewing the entropy is as a measure of our uncertainty about the value of X.** Consider a game where we are allowed to ask yes/no questions until we pinpoint the value X. Then the entropy of X is average number of questions we need to ask to get to the answer (if we have a good strategy for asking them). If we have little uncertainty about X, then we get to the value with few questions. An extreme case is when $\boldsymbol{H}_P(X) = 0$. It is easy to verify that this can happen only when one value of X has probability 1 and the rest probability

1. To justify this, note that $\lim_{\epsilon \to 0} \epsilon \log \frac{1}{\epsilon} = 0$.

2. This value is not generally an integer, so one cannot directly map x to a code word with $-\log P(x)$ bits. However, by coding longer sequences rather than individual values, we can come arbitrarily close to this bound.

0. In this case, we do not need to ask any questions to get to the value of X. On the other hand, if the value of X is very uncertain, then we need to ask many questions.

This discussion in fact identifies the two boundary cases for $\mathbf{H}_P(X)$.

Proposition A.1

$$0 \leq \mathbf{H}_P(X) \leq \log |Val(X)|$$

The definition of entropy naturally extends to multiple variables.

Definition A.2
joint entropy

Suppose we have a joint distribution over random variables $X_1, \ldots, X_n$. Then the joint entropy *of $X_1, \ldots, X_n$ is*

$$\mathbf{H}_P(X_1, \ldots, X_n) = \mathbf{E}_P\left[\log \frac{1}{P(X_1, \ldots, X_n)}\right].$$

■

The joint entropy captures how many bits are needed (on average) to encode joint instances of the variables.

A.1.2 Conditional Entropy and Information

Suppose we are encoding the values of X and Y. A natural question is what is the cost of encoding X if we are already encoding Y. Formally, we can examine the difference between $\mathbf{H}_P(X, Y)$ — the number of bits needed (on average) to encode of both variables, and $\mathbf{H}_P(Y)$ — the number of bits needed to encode Y alone.

Definition A.3
conditional entropy

The conditional entropy *of X given Y is*

$$\mathbf{H}_P(X \mid Y) = \mathbf{H}_P(X, Y) - \mathbf{H}_P(Y) = \mathbf{E}_P\left[\log \frac{1}{P(X \mid Y)}\right].$$

■

This quantity captures the additional cost (in terms of bits) of encoding X when we are already encoding Y. The definition gives rise to the *chain rule of entropy*:

entropy chain rule

Proposition A.2

For any distribution $P(X_1, \ldots, X_n)$, we have that

$$\mathbf{H}_P(X_1, \ldots, X_n) = \mathbf{H}_P(X_1) + \mathbf{H}_P(X_2 \mid X_1) + \ldots + \mathbf{H}_P(X_n \mid X_1, \ldots, X_{n-1}).$$

That is, to encode a joint value of $X_1, \ldots, X_n$, we first need to encode X_1, then encode X_2 given that we know the value of X_1, then encode X_3 given the first two, and so on. Note that, similarly to the chain rule of probabilities, we can expand the chain rule in any order we prefer; that is, all orders result in precisely the same value.

Intuitively, we would expect $\mathbf{H}_P(X \mid Y)$, the additional cost of encoding X when we already encode Y, to be at least as small as the cost of encoding X alone. To motivate that, we see that the worst case scenario is where we encode X as though we did not know the value of Y. Indeed, one can formally show

Proposition A.3

$$\mathbf{H}_P(X \mid Y) \leq \mathbf{H}_P(X).$$

The difference between these two quantities is of special interest.

Definition A.4 mutual information

The mutual information *between* X *and* Y *is*

$$\boldsymbol{I}_P(X;Y) = \boldsymbol{H}_P(X) - \boldsymbol{H}_P(X \mid Y) = \boldsymbol{E}_P\left[\log \frac{P(X \mid Y)}{P(X)}\right]. \quad \blacksquare$$

The mutual information captures how many bits we save (on average) in the encoding of X if we know the value of Y. Put in other words, it represents the extent to which the knowledge of Y reduces our uncertainty about X.

The mutual information satisfies several nice properties.

Proposition A.4

- $0 \leq \boldsymbol{I}_P(X;Y) \leq \boldsymbol{H}_P(X)$.
- $\boldsymbol{I}_P(X;Y) = \boldsymbol{I}_P(Y;X)$.
- $\boldsymbol{I}_P(X;Y) = 0$ *if and only if* X *and* Y *are independent.*

Thus, the mutual information is nonnegative, and equal to 0 if and only if the two variables are independent of each other. This is fairly intuitive, since if X and Y are independent, then learning the value of Y does not tell us any thing new about the value of X. In fact, we can view the mutual information as a quantitative measure of the strength of the dependency between X and Y. The bigger the mutual information, the stronger the dependency. The extreme upper value of the mutual information is when X is a deterministic function of Y (or vice versa). In this case, once we know Y we are certain about the value of X, and so $\boldsymbol{I}_P(X;Y) = \boldsymbol{H}_P(X)$. That is, Y supplies the maximal amount of information about X.

A.1.3 Relative Entropy and Distances Between Distributions

In many situations when doing probabilistic reasoning, we want to compare two distributions. For example, we might want to approximate a distribution by one with desired qualities (say, simpler representation, more efficient to reason with, and so on) and want to evaluate the quality of a candidate approximation. Another example is in the context of learning a distribution from data, where we want to compare the learned distribution to the "true" distribution from which the data was generated.

distance measure

Thus, we want to construct a *distance measure* d that evaluates the distance between two distributions. There are some properties that we might wish for in such a distance measure:

Positivity: $d(P,Q)$ is always nonnegative, and is zero if and only if $P = Q$;

Symmetry: $d(P,Q) = d(Q,P)$.

Triangle inequality: for any three distributions P, Q, R, we have that

$$d(P,R) \leq d(P,Q) + d(Q,R).$$

distance metric

When a distance measure d satisfies these criteria, it is called a *distance metric*.

We now review several common approaches used to compare distributions. We begin by describing one important measure that is motivated by information-theoretic considerations. It also turns out to arise very naturally in a wide variety of probabilistic settings.

A.1.3.1 Relative Entropy

Consider the preceding discussion of compression. As we discussed, the entropy measures the performance of "optimal" code that assigns the value x a code of length $-\log P(x)$. However, in many cases in practice, we do not have access to the true distribution P that generates the data we plan to compress. Thus, instead of using P we use another distribution Q (say one we estimated from prior data, or supplied by a domain expert), which is our best guess for P.

Suppose we build a code using Q. Treating Q as a proxy to the real distribution, we use $-\log Q(x)$ bits to encode the value x. Thus, the expected number of bits we use on data generated from P is

$$\mathbf{E}_P\left[\log \frac{1}{Q(x)}\right].$$

A natural question is how much we lost, due to the inaccuracy of using Q. Thus, we can examine the difference between this encoding and the best achievable one, $\mathbf{H}_P(X)$. This difference is called the relative entropy.

Definition A.5 relative entropy

Let P and Q be two distributions over random variables $X_1, \ldots, X_n$. The relative entropy *of P and Q is*

$$\mathbf{D}(P(X_1, \ldots, X_n) \| Q(X_1, \ldots, X_n)) = \mathbf{E}_P\left[\log \frac{P(X_1, \ldots, X_n)}{Q(X_1, \ldots, X_n)}\right]. \quad \blacksquare$$

When the set of variables in question is clear from the context, we use the shorthand notation $\mathbf{D}(P\|Q)$. This measure is also often known as the *Kullback-Liebler divergence* (or *KL-divergence*).

This discussion suggests that the relative entropy measures the additional cost imposed by using a wrong distribution Q instead of P. Thus, Q is close, in the sense of relative entropy, to P if this cost is small. As we expect, the additional cost of using the wrong distribution is always positive. Moreover, the relative entropy is 0 if and only if the two distributions are identical:

Proposition A.5

$\mathbf{D}(P\|Q) \geq 0$, and is equal to zero if and only if $P = Q$.

It is also natural to ask whether the relative entropy is also bounded from above. As we can quickly convince ourselves, if there is a value x such that $P(x) > 0$ and $Q(x) = 0$, then the relative entropy $\mathbf{D}(P\|Q)$ is infinite. More precisely, if we consider a sequence of distributions Q_ϵ such that $Q_\epsilon(x) = \epsilon$, then $\lim_{\epsilon \to 0} \mathbf{D}(P\|Q_\epsilon) = \infty$.

It is natural ask whether the relative entropy defines a distance measure over distributions. Proposition A.5 shows that the relative entropy satisfies the positivity property specified above. Unfortunately, **positivity is the only property of distances that relative entropy satisfies; it satisfies neither symmetry nor the triangle inequality.** Given how natural these properties are, one might wonder why relative entropy is used at all. Aside from the fact that it arises very naturally in many settings, it also has a variety of other useful properties, that often make up for the lack of symmetry and the triangle inequality.

A.1.3.2 Conditional Relative Entropy

As with entropies, we can define a notion of conditional relative entropy.

Definition A.6 conditional relative entropy

Let P and Q be two distributions over random variables X, Y. The conditional relative entropy *of P and Q, is*

$$\mathbf{D}(P(X \mid Y) \| Q(X \mid Y)) = \mathbf{E}_P \left[\log \frac{P(X \mid Y)}{Q(X \mid Y)} \right].$$

■

We can think of the conditional relative entropy $\mathbf{D}(P(X \mid Y) \| Q(X \mid Y))$ as the weighted sum of the relative entropies between the conditional distributions given different values of y

$$\mathbf{D}(P(X \mid Y) \| Q(X \mid Y)) = \sum_y P(y) \mathbf{D}(P(X \mid y) \| Q(X \mid y)).$$

relative entropy chain rule

Using the conditional relative entropy, we can write the *chain rule of relative entropy*:

Proposition A.6

Let P and Q be distributions over $X_1, \ldots, X_n$, then

$$\begin{aligned} \mathbf{D}(P \| Q) &= \mathbf{D}(P(X_1) \| Q(X_1)) + \\ &\quad \mathbf{D}(P(X_2 \mid X_1) \| Q(X_2 \mid X_1)) + \ldots + \\ &\quad \mathbf{D}(P(X_n \mid X_1, \ldots, X_{n-1}) \| Q(X_n \mid X_1, \ldots, X_{n-1})). \end{aligned}$$

Using the chain rule, we can prove additional properties of the relative entropy. First, using the chain rule and the fact that $\mathbf{D}(P(Y \mid X) \| Q(Y \mid X)) \geq 0$, we can get the following property.

Proposition A.7

$$\mathbf{D}(P(X) \| Q(X)) \leq \mathbf{D}(P(X, Y) \| Q(X, Y)).$$

That is, the relative entropy of a marginal distributions is upper-bounded by the relative entropy of the joint distributions. This observation generalizes to situations where we consider sets of variables. That is,

$$\mathbf{D}(P(X_1, \ldots, X_k) \| Q(X_1, \ldots, X_k)) \leq \mathbf{D}(P(X_1, \ldots, X_n) \| Q(X_1, \ldots, X_n))$$

for $k \leq n$.

Suppose that X and Y are independent in both P and Q. Then, we have that $P(Y \mid X) = P(Y)$, and similarly, $Q(Y \mid X) = Q(Y)$. Thus, we conclude that $\mathbf{D}(P(Y \mid X) \| Q(Y \mid X)) = \mathbf{D}(P(Y) \| Q(Y))$. Combining this observation with the chain rule, we can prove an additional property.

Proposition A.8

If both P and Q satisfy $(X \perp Y)$, then

$$\mathbf{D}(P(X, Y) \| Q(X, Y)) = \mathbf{D}(P(X) \| Q(X)) + \mathbf{D}(P(Y) \| Q(Y)).$$

A.1.3.3 Other Distance Measures

There are several different metric distances between distributions that we may consider. Several simply treat a probability distribution as a vector in $\mathbb{R}^N$ (where N is the dimension of our probability space), and use standard distance metrics for Euclidean spaces. More precisely, let P and Q be two distributions over $X_1, \ldots, X_n$. The three most commonly used distance metrics of this type are:

- The L_1 distance: $\|P - Q\|_1 = \sum_{x_1,\ldots,x_n} |P(x_1, \ldots, x_n) - Q(x_1, \ldots, x_n)|$.
- The L_2 distance: $\|P - Q\|_2 = \left(\sum_{x_1,\ldots,x_n} (P(x_1, \ldots, x_n) - Q(x_1, \ldots, x_n))^2\right)^{\frac{1}{2}}$.
- The L_∞ distance: $\|P - Q\|_\infty = \max_{x_1,\ldots,x_n} |P(x_1, \ldots, x_n) - Q(x_1, \ldots, x_n)|$.

variational distance

An apparently different distance measure is the *variational distance*, which seems more specifically tailored to probability distributions, rather than to general real-valued vectors. It is defined as the maximal difference in the probability that two distributions assign to *any* event that can be described by the distribution. For two distributions P, Q over an event space $\mathcal{S}$, we define:

$$\mathbb{D}_{var}(P; Q) = \max_{\alpha \in \mathcal{S}} |P(\alpha) - Q(\alpha)|. \tag{A.1}$$

Interestingly, this distance turns out to be exactly half the L_1 distance:

Proposition A.9 *Let P and Q be two distributions over $\mathcal{S}$. Then*

$$\mathbb{D}_{\text{var}}(P; Q) = \frac{1}{2}\|P - Q\|_1.$$

These distance metrics are all useful in the analysis of approximations, but, unlike the relative entropy, they do not decompose by a chain-rule-like construction, often making the analytical analysis of such distances harder. However, we can often use an analysis in terms of relative entropy to provide bounds on the L_1 distance, and hence also on the variational distance:

Theorem A.1 *For any two distribution P and Q, we have that*

$$\|P - Q\|_1 \leq \left((2 \ln 2)\mathbb{D}(P\|Q)\right)^{1/2}.$$

A.2 Convergence Bounds

In many situations that we cover in this book, we are given a set of samples generated from a distribution, and we wish to estimate certain properties of the generating distribution from the samples. We now review some properties of random variables that are useful for this task. The derivation of these convergence bounds is central to many aspects of probability theory, statistics, and randomized algorithms. Motwani and Raghavan (1995) provide one good introduction on this topic and its applications to the analysis of randomized algorithms.

Specifically, suppose we have a biased coin that has an unknown probability p of landing heads. We can estimate the value of p by tossing the coin several times and counting the

frequency of heads. More precisely, assume we have a data set $\mathcal{D}$ consisting of M coin tosses, that is, M trials from a Bernoulli distribution. The m'th coin toss is represented by a binary variable $X[m]$ that has value 1 if the coin lands heads, and 0 otherwise. Since each toss is separate from the previous one, we are assuming that all these random variables are independent. Thus, these variables are *independence and identically distribution*, or *IID*. It is easy to compute the expectation and variance of each $X[m]$:

IID

- $\mathbf{E}[X[m]] = p$.
- $\mathbf{Var}[X[m]] = p(1-p)$.

A.2.1 Central Limit Theorem

We are interested in the sum of all the variables $S_{\mathcal{D}} = X[1] + \ldots + X[M]$ and in the fraction of successful trials $T_{\mathcal{D}} = \frac{1}{M} S_{\mathcal{D}}$. Note that $S_{\mathcal{D}}$ and $T_{\mathcal{D}}$ are functions of the data set $\mathcal{D}$. As $\mathcal{D}$ is chosen randomly, they can be viewed as random variables over the probability space defined by different possible data sets $\mathcal{D}$. Using properties of expectation and variance, we can analyze the properties of these random variables.

- $\mathbf{E}[S_{\mathcal{D}}] = M \cdot p$, by linearity of expectation.
- $\mathbf{Var}[S_{\mathcal{D}}] = M \cdot p(1-p)$, since all the all the $X[i]$'s are independent.
- $\mathbf{E}[T_{\mathcal{D}}] = p$.
- $\mathbf{Var}[T_{\mathcal{D}}] = \frac{1}{M} p(1-p)$, since $\mathbf{Var}\left[\frac{1}{M} S_{\mathcal{D}}\right] = \frac{1}{M^2} \mathbf{Var}[S_{\mathcal{D}}]$.

The fact that $\mathbf{Var}[T_{\mathcal{D}}] \to 0$ as $M \to \infty$ suggests that for sufficiently large M the distribution of $T_{\mathcal{D}}$ is concentrated around p. In fact, a general result in probability theory allows us to conclude that this distribution has a particular form:

Theorem A.2 central limit theorem

(Central Limit Theorem) *Let $X[1], X[2], \ldots$ be a series of IID random variables, where each $X[m]$ is sampled from a distribution such that $\mathbf{E}[X[m]] = \mu$, and variance $\mathbf{Var}[X[m]] = \sigma^2$ $(0 < \sigma < \infty)$. Then*

$$\lim_{M \to \infty} P\left(\frac{\sum_m (X[m] - \mu)}{\sqrt{M}\sigma} < r \right) = \Phi(r),$$

where $\Phi(r) = P(Z < r)$ for a Gaussian variable Z with distribution $\mathcal{N}(0; 1)$.

Thus, if we collect a large number of repeated samples from the same distribution, then the distribution of the random variable $(S_{\mathcal{D}} - \mathbf{E}[S_{\mathcal{D}}])/\sqrt{\mathbf{Var}[S_{\mathcal{D}}]}$ is roughly *Gaussian*. In other words, the distribution of $S_{\mathcal{D}}$ is, at the limit, close to a Gaussian with the appropriate expectation and variance: $\mathcal{N}(\mathbf{E}[S_{\mathcal{D}}]; \mathbf{Var}[S_{\mathcal{D}}])$.

Gaussian

There are variants of the central limit theorem for the case where each $X[m]$ has a different distribution. These require additional technical conditions that we do not go into here. However, the general conclusion is similar — the sum of many independent random variables has a distribution that is approximately Gaussian. This is often a justification for using a Gaussian distribution in modeling quantities that are the cumulative effect of many independent (or almost independent) factors.

estimator

The quantity $T_{\mathcal{D}}$ is an *estimator* for the mean μ: a statistical function that we can use to estimate the value of μ. The mean and variance of an estimator are the two key quantities for evaluating it. The mean of the estimator tells us the value around which its values are going to be concentrated. When the mean of the estimator is the target value μ, it is called an *unbiased estimator* for the quantity μ — an estimator whose mean is precisely the desired value. In general, lack of bias is a desirable property in an estimator: it tells us that, although they are noisy, at least the values obtained by the estimator are centered around the right value. The variance of the estimator tells us the "spread" of values we obtain from it. Estimators with high variance are not very reliable, as their value is likely to be far away from their mean.

unbiased estimator

Applying the central limit theorem to our problem, we see that, for sufficiently large M, the variable $T_{\mathcal{D}}$ has a roughly Gaussian distribution with mean p and variance $\frac{p(1-p)}{M}$.

A.2.2 Convergence Bounds

In many situations, we are interested not only in the asymptotic distribution of $T_{\mathcal{D}}$, but also in the probability that $T_{\mathcal{D}}$ is close to p for a concrete choice of M. We can bound this probability in several ways. One of the simplest is by using Chebyshev's inequality; see exercise 12.1. This bound, however, is quite loose, as it assumes quadratic decay in the distance $|T_{\mathcal{D}} - p|$. Other, more refined bounds, can be used to prove an exponential rate of decay in this distance. There are many variants of these bounds, of which we describe two.

Hoeffding bound

The first, called *Hoeffding bound*, measures error in terms of the absolute distance $|T_{\mathcal{D}} - p|$.

Theorem A.3

Let $\mathcal{D} = \{X[1], \ldots, X[M]\}$ be a sequence of M independent Bernoulli trials with probability of success p. Let $T_{\mathcal{D}} = \frac{1}{M}\sum_m X[m]$. Then

$$\begin{aligned} P_{\mathcal{D}}(T_{\mathcal{D}} > p + \epsilon) &\leq e^{-2M\epsilon^2} \\ P_{\mathcal{D}}(T_{\mathcal{D}} < p - \epsilon) &\leq e^{-2M\epsilon^2}. \end{aligned}$$

The bound asserts that, with very high probability, $T_{\mathcal{D}}$ is within an additive error ϵ of the true probability p. The probability here is taken relative to possible data sets $\mathcal{D}$. Intuitively, we might end up with really unlikely choices of $\mathcal{D}$, for example, ones where we get the same value all the time; these choices will clearly give wrong results, but they are very unlikely to arise as a result of a random sampling process. Thus, the bound tells us that, **for most data sets $\mathcal{D}$ that we generate at random, we obtain a good estimate. Furthermore, the fraction of "bad" sample sets $\mathcal{D}$, those for which the estimate is more than ϵ from the true value, diminishes exponentially as the number of samples M grows.**

Chernoff bound

The second bound, called the *Chernoff bound*, measures error in terms of the relative size of this distance to the size of p.

Theorem A.4

Let $\mathcal{D} = \{X[1], \ldots, X[M]\}$ be a sequence of M independent Bernoulli trials with probability of success p. Let $T_{\mathcal{D}} = \frac{1}{M}\sum_m X[m]$, then

$$\begin{aligned} P_{\mathcal{D}}(T_{\mathcal{D}} > p(1+\epsilon)) &\leq e^{-Mp\epsilon^2/3} \\ P_{\mathcal{D}}(T_{\mathcal{D}} < p(1-\epsilon)) &\leq e^{-Mp\epsilon^2/2}. \end{aligned}$$

Let $\sigma_M = \sqrt{\mathbf{Var}[T_{\mathcal{D}}]}$ be the standard deviation of $T_{\mathcal{D}}$ for $\mathcal{D}$ of size M. Using the multiplicative Chernoff bound, we can show that

$$P_{\mathcal{D}}(|T_{\mathcal{D}} - p| \geq k\sigma) \leq 2e^{-k^2/6}. \tag{A.2}$$

This inequality should be contrasted with the Chebyshev inequality. The big difference owes to the fact that the Chernoff bound exploits the particular properties of the distribution of $T_{\mathcal{D}}$.

A.3 Algorithms and Algorithmic Complexity

In this section, we briefly review relevant algorithms and notions from algorithmic complexity. Cormen et al. (2001) is a good source for learning about algorithms, data structures, graph algorithms, and algorithmic complexity; Papadimitriou (1993) and Sipser (2005) provide a good introduction to the key concepts in computational complexity.

A.3.1 Basic Graph Algorithms

Given a graph structure, there are many useful operations that we might want to perform. For example, we might want to determine whether there is a certain type of path between two nodes. In this section, we survey algorithms for performing two key tasks that will be of use in several places throughout this book. Additional algorithms, for more specific tasks, are presented as they become relevant.

Algorithm A.1 Topological sort of a graph

```
    Procedure Topological-Sort (
      G = (X, E)    // A directed graph
    )
1     Set all nodes to be unmarked
2     for i = 1, ..., n
3       Select any unmarked node X all of whose parents are marked
4       d(X) ← i
5       Mark X
6     return (d⃗)
```

topological ordering

One algorithm, shown in algorithm A.1, finds a *topological ordering* of the nodes in the graph, as defined in definition 2.19.

maximum weight spanning tree

Another useful algorithm is one that finds, in a weighted undirected graph $\mathcal{H}$ with nonnegative edge weights, a *maximum weight spanning tree*. More precisely, a subgraph is said to be a *spanning tree* if it is a tree and it spans all vertices in the graph. Similarly, a *spanning forest* is a forest that spans all vertices in the graph. A maximum weight spanning tree (or forest) is the tree (forest) whose edge-weight sum is largest among all spanning trees (forests).

Algorithm A.2 Maximum weight spanning tree in an undirected graph

Procedure Max-Weight-Spanning-Tree (
$\mathcal{H} = (\mathcal{N}, \mathcal{E})$
$\{w_{ij} \ : \ (X_i, X_j) \in \mathcal{E}\}$
)
$\mathcal{N}_T \leftarrow \{X_1\}$
$\mathcal{E}_T \leftarrow \emptyset$
while $\mathcal{N}_T \neq \mathcal{X}$
$\mathcal{E}' \leftarrow \{(i,j) \in \mathcal{E} \ : \ X_i \in \mathcal{N}_T, X_j \notin \mathcal{N}_T\}$
$(X_i, X_j) \leftarrow \arg\max_{(X_i,X_j)\in\mathcal{E}'} w_{ij}$
// (X_i, X_j) is the highest-weight edge between a node in T and a node out of T
$\mathcal{N}_T \leftarrow \mathcal{N}_T \cup \{X_j\}$
$\mathcal{E}_T \leftarrow \mathcal{E}_T \cup \{(X_i, X_j)\}$
return $(\mathcal{E}_T)$

A.3.2 Analysis of Algorithmic Complexity

A key step in evaluating the usefulness of an algorithm is to analyze its computational cost: the amount of time it takes to complete the computation and the amount of space (memory) required. To evaluate the algorithm, we are usually not interested in the cost for a particular input, but rather in the algorithm's performance over a set of inputs. Of course, we would expect most algorithms to run longer when applied to larger problems. Thus, the complexity of an algorithm is usually measured in terms of its performance, as a function of the size of the input given to it. Of course, to determine the precise cost of the algorithm, we need to know exactly how it is implemented and even which machine it will be run on. However, we can often determine the scalability of an algorithm at a more abstract level, without worrying about the details of its implementation. We now provide a high-level overview of some of the basic concepts underlying such analysis.

Consider an algorithm that takes a list of n numbers and adds them together to compute their sum. Assuming the algorithm simply traverses the list and computes the sum as it goes along, it has to perform some fixed number of basic operations for each element in the list. The precise operations depend on the implementation: we might follow a pointer in a linked list, or simply increment a counter in an array. Thus, the precise cost might vary based on the implementation. But, the total number of operations per list element is some fixed constant factor. Thus, for any reasonable implementation, the running time of the algorithm will be bounded by $C \cdot n$ for some constant C. In this case, we say that the *asymptotic complexity* of the algorithm is $O(n)$, where the $O()$ notation makes implicit the precise nature of the constant factor, which can vary from one implementation to another. This idea only makes sense if we consider the running time as a function of n. For any fixed problem size, say up to 100, we can always find a constant C (for instance, a million years) such that the algorithm takes time no more than C. However, even if we are not interested in problems of unbounded size, evaluating the way in which the running time varies as a function of the problem size is the first step to understanding how well it will

asymptotic complexity

scale to large problems.

To take a more relevant example, consider the maximum weight spanning tree procedure of algorithm A.2. A (very) naive implementation of this algorithm traverses all of the edges in the graph every time a node is added to the spanning tree; the resulting cost is $O(mn)$ where m is the number of edges and n the number of nodes. A more careful implementation of the data structures, however, maintains the edges in a sorted data structure known as a heap, and the list of edges adjacent to a node in an *adjacency list*. In this case, the complexity of the algorithm can be $O(m \log n)$ or (with a yet more sophisticated data structure) $O(m + n \log n)$. Surprisingly, even more sophisticated implementations exist whose complexity is very close to linear time in m.

More generally, we can provide the following definition:

Definition A.7 *Consider an algorithm $\mathcal{A}$ that takes as input problems Π from a particular class, and returns an output. Assume that the size of each possible input problem Π is measured using some set of parameters $n_1, \ldots, n_k$. We say that the running time of $\mathcal{A}$ is $O(f(n_1, \ldots, n_k))$ for some function f (called "big O of f"), if, for $n_1, \ldots, n_k$ sufficiently large, there exists a constant C such that, for any possible input problem Π, the running time of $\mathcal{A}$ on Π is at most $C \cdot f(n_1, \ldots, n_k)$.* ■

In our example, each problem Π is a graph, and its size is defined by two parameters: the number of nodes n and the number of edges m. The function $f(n, m)$ is simply $n + m$.

running time

When the function f is linear in each of the input size parameters, we say that the *running time* of the algorithm is linear, or that the algorithm has *linear time*. We can similarly define notions of *polynomial time* and *exponential time*. It may be useful to distinguish different rates of growth in the different parameters. For example, if we have a function that has the form $f(n, m) = n^2 + 2^m$, we might say that the function is polynomial in n but exponential in m.

polynomial time

exponential time

Although one can find algorithms at various levels of complexity, the key cutoff between feasible and infeasible computations is typically set between algorithms whose complexity is polynomial and those whose complexity is exponential. Intuitively, an algorithm whose complexity is exponential allows virtually no useful scalability to larger problems. For example, assume we have an algorithm whose complexity is $O(2^n)$, and that we can now solve instances whose size is N. If we wait a few years and get a computer that is twice as fast as the one we have now, we will be able to solve only instances whose size is $N + 1$, a negligible improvement.

We can also see this phenomenon by comparing the growth curves for various cost functions, as in figure A.1. We see that the constant factors in front of the polynomial functions have some impact on very small problem sizes, but even for moderate problem sizes, such as 20, the exponential function quickly dominates and grows to the point of infeasibility. Thus, a major distinction is made between algorithms that run in polynomial time and those whose running time is exponential. **While the exponential-polynomial distinction is a critical one, there is also a tendency to view polynomial-time algorithms as tractable. This view, unfortunately, is overly simplified: an algorithm whose running time is $O(n^3)$ is not generally tractable for problems where n is in the thousands.**

Algorithmic theory offers a suite of tools for constructing efficient algorithms for certain types of problems. One such tool, which we shall use many times throughout the book, is *dynamic programming*, which we describe in more detail in appendix A.3.3. Unfortunately, not all problems are not amenable to these techniques, and a broad class of highly important problems fall into a category for which polynomial-time algorithms are extremely unlikely to

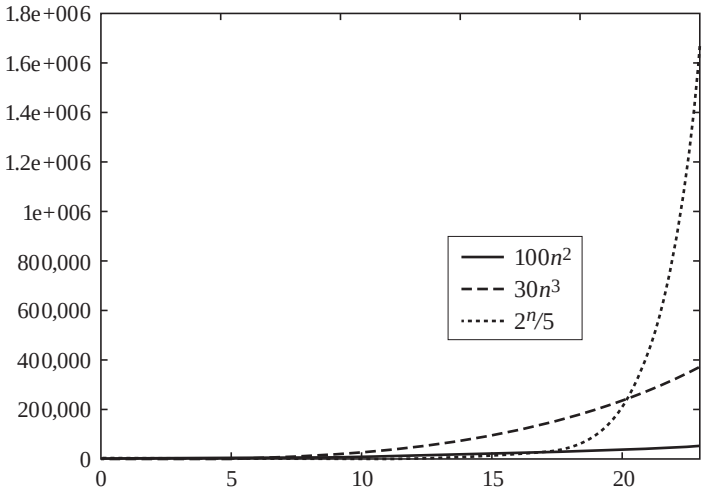


Figure A.1 Illustration of asymptotic complexity. The growth curve of three functions: The solid line is $100n^2$, the dashed line is $30n^3$, and the dotted line is $2^n/5$.

exist; see appendix A.3.4.

A.3.3 Dynamic Programming

As we discussed earlier, several techniques can be used to provide efficient solutions to apparently challenging computational problems. **One important tool is *dynamic programming*, a general method that we can apply when the solution to a problem requires that we solve many smaller subproblems that recur many times. In this case, we are often better off precomputing the solution to the subproblems, storing them, and using them to compute the values to larger problems.**

dynamic programming

Perhaps the simplest application of dynamic programming is the problem of computing *Fibonacci numbers*, defined via the recursive equations:

$$\begin{aligned} F_0 &= 1 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2}. \end{aligned}$$

Thus, we have that $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, and so on.

One simple algorithm to compute *Fibonacci(n)* is to use the recursive definition directly, as shown in algorithm A.3. Unrolling the computation, we see that the first of these recursive calls, Fibonacci($n-1$), calls Fibonacci($n-2$) and Fibonacci($n-3$). Thus, we are already have two calls to Fibonacci($n-2$). Similarly, Fibonacci($n-2$) also calls Fibonacci($n-3$), another redundant computation. If we carry through the entire recursive analysis, we can show that the running time of the algorithm is exponential in n.

On the other hand, we can compute "bottom up", as in algorithm A.4. Here, we start with F_0

Algorithm A.3 Recursive algorithm for computing Fibonacci numbers

```
    Procedure Fibonacci (
      n
    )
1     if (n = 0 or n = 1) then
2       return (1)
3     return (Fibonacci(n − 1) + Fibonacci(n − 2))
```

Algorithm A.4 Dynamic programming algorithm for computing Fibonacci numbers

```
    Procedure Fibonacci (
      n
    )
1     F_0 ← 1
2     F_1 ← 1
3     for i = 2, ..., n
4       F_i ← F_{i−1} + F_{i−2}
5     return (F_n)
```

and F_1, compute F_2 from F_0 and F_1, compute F_3 from F_1 and F_2, and so forth. Clearly, this process computes F_n in time $O(n)$. We can view this alternative algorithm as precomputing and then caching (or storing) the results of the intermediate computations performed on the way to each F_i, so that each only has to be performed once.

More generally, if we can define the set of intermediate computations required and how they depend on each other, we can often use this caching idea to avoid redundant computation and provide significant savings. This idea underlies most of the exact inference algorithms for graphical models.

A.3.4 Complexity Theory

In appendix A.3.3, we saw how the same problem might be solvable by two algorithms that have radically different complexities. Examples like this raise an important issue regarding the algorithm design process: If we come up with an algorithm for a problem, how do we know whether its computational complexity is the best we can achieve? In general, unfortunately, we cannot tell. There are very few classes of problems for which we can give nontrivial lower bounds on the amount of computation required for solving them.

complexity theory

However, there are certain types of problems for which we can provide, not a guarantee, but at least a certain expectation regarding the best achievable performance. *Complexity theory* has defined classes of problems that are, in a sense, equivalent to each other in terms of their computational cost. In other words, we can show that an algorithm for solving one problem can be converted into an algorithm that solves another problem. Thus, if we have an efficient algorithm for solving the first problem, it can also be used to solve the second efficiently.

The most prominent such class of problems is that of $\mathcal{NP}$-complete problems; this class

contains many problems for which researchers have unsuccessfully tried, for decades, to find efficient algorithms. Thus, by proving that a problem is $\mathcal{NP}$-complete, we are essentially showing that it is "as easy" as all other $\mathcal{NP}$-complete problems. Finding an efficient (polynomial time) algorithm for this problem would therefore give rise to efficient algorithms for all $\mathcal{NP}$-complete problems, an extremely unlikely event. In other words, by showing that a problem is $\mathcal{NP}$-complete, we are essentially showing that it is extremely unlikely to have an efficient solution. We now provide some of the formal basis for this type of discussion.

A.3.4.1 Decision Problems

A *decision problem* Π is a task that has the following form: The program must accept an input ω and decide whether it satisfies a certain condition or not. A prototypical decision problem is the *SAT* problem, which is defined as the problem of taking as input a formula in propositional logic, and returning *true* if the formula has a satisfying assignment and *false* if it does not. For example, an algorithm for the SAT problem should return *true* for the formula

$$(q_1 \vee \neg q_2 \vee q_3) \wedge (\neg q_1 \vee q_2 \vee \neg q_3), \tag{A.3}$$

which has (among others) the satisfying assignment $q_1 = \textit{true}; q_2 = \textit{true}; q_3 = \textit{true}$. It would return *false* for the formula

$$(\neg q_1 \vee \neg q_2) \wedge (q_2 \vee q_3) \wedge (\neg q_1 \vee \neg q_3), \tag{A.4}$$

which has no satisfying assignments.

We often use a somewhat restricted version of the SAT problem, called *3-SAT*.

Definition A.8 3-SAT

A formula ϕ is said to be a 3-SAT *formula over the Boolean (binary-valued) variables $q_1, \ldots, q_n$ if it has the following form: ϕ is a conjunction: $\phi = C_1 \wedge \ldots \wedge C_m$. Each C_i is a* clause *of the form $\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$. Each $\ell_{i,j}$ ($i = 1, \ldots, m$; $j = 1, 2, 3$) is a* literal*, which is either q_k or $\neg q_k$ for some $k = 1, \ldots, n$.* ■

A decision problem Π is associated with a language $\mathcal{L}_\Pi$ that defines the precise set of instances for which a correct algorithm for Π must return *true*. In the case of 3-SAT, $\mathcal{L}_{3SAT}$ is the set of all correct encodings of propositional 3-SAT formulas that are satisfiable.

A.3.4.2 $\mathcal{P}$ and $\mathcal{NP}$

A decision problem is said to be in the class $\mathcal{P}$ if there exists a deterministic algorithm that takes an instance ω and determines whether or not $\omega \in \mathcal{L}_\Pi$, in polynomial time in the size of the input ω. In SAT, for example, the input is the formula, and its size is simply its length.

We can also define a significantly more powerful type of computation that allows us to provide a formal foundation for a very rich class of problems. Consider again our SAT algorithm. The naive algorithm for determining whether a formula is satisfiable enumerates all of the assignments, and returns *true* if one of them satisfies the formula. Imagine that we allow the algorithm a notion of a "lucky guess": the algorithm is allowed to guess an assignment, and then verify whether it satisfies the formula. The algorithm can determine if the formula is satisfiable simply by having one guess that works out. In other words, we assume that the

algorithm asserts that the formula is in $\mathcal{L}_{3SAT}$ if there is some guess that works out. This type of computation is called a *nondeterministic computation*. A fully formal definition requires that we introduce a range of concepts (such as Turing Machines) that are outside the scope of this book. Roughly speaking, a *nondeterministic decision algorithm* has the following form. The first stage is a guessing stage, where the algorithm nondeterministically produces some guess γ. The second stage is a deterministic verifying stage that either accepts its input ω based on γ or not. The algorithm as a whole is said to accept ω if it accepts γ using any one of its guesses. A decision problem Π is in the class $\mathcal{NP}$ if there exists a nondeterministic algorithm that accepts an instance ω if and only if $\omega \in \mathcal{L}_\Pi$, and if the verification stage can be executed in polynomial time in the length of ω. Clearly, SAT is in $\mathcal{NP}$: the guesses γ are possible assignments, and they are verified in polynomial time simply by testing whether the assignment γ satisfies the input formula ϕ.

Because deterministic computations are a special case of nondeterministic ones, we have that $\mathcal{P} \subseteq \mathcal{NP}$. The converse of this inclusion is the biggest open problem in computational complexity. In other words, can every problem that can be solved in polynomial time using a lucky guess also be solved in polynomial time without guessing?

As stated, it seems impossible to get a handle on this problem: The number of problems in $\mathcal{NP}$ is potentially unlimited, and even if we find an efficient algorithm for one problem, what does that tell us about the class in general? The notion of $\mathcal{NP}$-complete problems gives us a tool for reducing this unmanageable question into a much more compact one. Roughly speaking, the class $\mathcal{NP}$ has a set of problems that are the "hardest problems in $\mathcal{NP}$": if we can solve them in polynomial time, we can provably solve any problem in $\mathcal{NP}$ in polynomial time. These problems are known as $\mathcal{NP}$-complete problems.

$\mathcal{NP}$-hard

More formally, we say that a decision problem Π is *$\mathcal{NP}$-hard* if for every decision problem Π' in $\mathcal{NP}$, there is a polynomial-time transformation of inputs such that an input for Π' belongs to $\mathcal{L}_{\Pi'}$ if and only if the transformed instance belongs to $\mathcal{L}_\Pi$. This type of transformation is called a *reduction* of one problem to another. When we have such a reduction, any algorithm $\mathcal{A}$ that solves the decision problem Π can be used to solve Π': We simply convert each instance of Π' to the corresponding instance of Π, and apply $\mathcal{A}$. An $\mathcal{NP}$-hard problem can be used in this way for any problem in $\mathcal{NP}$. Thus, it provides a universal solution for any $\mathcal{NP}$-problem. It is possible to show that the SAT problem is $\mathcal{NP}$-hard. A problem Π is said to be $\mathcal{NP}$-complete if it is both $\mathcal{NP}$-hard and in $\mathcal{NP}$. The 3-SAT problem is $\mathcal{NP}$-complete, as are many other important problems. For example, the *Max-Clique Problem* of deciding whether an undirected graph has a clique of size at least K (where K is a parameter to the algorithm) is also $\mathcal{NP}$-hard.

reduction

Max-Clique Problem

At the moment, it is not yet known whether $\mathcal{P} = \mathcal{NP}$. Much work has been devoted to investigating both sides of this conjecture. In particular, decades of research have been spent on failed attempts to find polynomial-time algorithms for many $\mathcal{NP}$-complete problems, such as SAT or Max-Clique. The lack of success suggests that probably no such algorithm exists for any $\mathcal{NP}$-hard problem, and that $\mathcal{P} \neq \mathcal{NP}$. Thus, a standard way of showing that a particular problem Π probably is unlikely to have a polynomial time algorithm is to show that it is $\mathcal{NP}$-hard. In other words, we try to find a reduction from some known $\mathcal{NP}$-hard problem, such as SAT, to the problem of interest. If we construct such a reduction, then we have shown the following: If we find a polynomial-time algorithm for Π, we have also provided a polynomial-time algorithm for all $\mathcal{NP}$-complete problems, and shown that $\mathcal{NP} = \mathcal{P}$. Although this is not impossible, it is currently believed to be highly unlikely.

Thus, if we show that a problem is $\mathcal{NP}$-hard, we should probably resign ourselves to algorithms that are exponential-time in the worst case. However, as we will see, there are many cases where algorithms can be exponential-time in the worst case, yet achieve significantly better performance in practice. Because many of the problems we encounter are $\mathcal{NP}$-hard, finding tractable cases and providing algorithms for them is where most of the interesting work takes place.

A.3.4.3 Other Complexity Classes

The classes $\mathcal{P}$ and $\mathcal{NP}$ are the most important and commonly used classes used to describe the computational complexity of problems, but they are only part of a rich framework used for classifying problems based on their time or space complexity. In particular, the class $\mathcal{NP}$ is only the first level in an infinite hierarchy of increasingly larger classes. Classes higher in the hierarchy might or might not be harder than the lower classes; this problem also is a major open problem in complexity theory.

A different dimension along which complexity can vary relates to the existential nature of the definition of the class $\mathcal{NP}$. A problem is in $\mathcal{NP}$ if there is some guess on which a polynomial time computation succeeds (returns *true*). In our SAT example, the guesses were different assignments that could satisfy the formula ϕ defined in the problem instance. However, we might want to know what fraction of the computations succeed. In our SAT example, we may want to compute the exact number (or fraction) of assignments satisfying ϕ. This problem is no longer a decision problem, but rather a counting problem that returns a numeric output.

$\#\mathcal{P}$

The class $\#\mathcal{P}$ is defined precisely for problems that return a numerical value. Such a problem is in $\#\mathcal{P}$ if the number can be computed as the number of accepting guesses of a nondeterministic polynomial time algorithm. The problem of counting the number of satisfying assignments to a 3-SAT formula is clearly in $\#\mathcal{P}$. Like the class of $\mathcal{NP}$-hard problems, there are problems that are at least as hard as any problem in $\#\mathcal{P}$. The problem of counting satisfying assignments is the canonical $\#\mathcal{P}$-hard problem. This problem is clearly $\mathcal{NP}$-hard: if we can solve it, we can immediately solve the 3-SAT decision problem. For trivial reasons, it is not in $\mathcal{NP}$, because it is a counting problem, not a decision problem. However, it is generally believed that the counting version of the 3-SAT problem is inherently more difficult than the original decision problem, in that we can use them to solve problems that are "harder" than $\mathcal{NP}$.

Finally, another, quite different, complexity class is the class of *randomized polynomial time algorithms* — those that can be solved using a polynomial time algorithm that makes random guesses. There are several ways of defining when a randomized algorithm accepts a particular input; we provide one of them. A decision problem Π is in the class $\mathcal{RP}$ if there exists a randomized algorithm that makes a guess probabilistically, and then processes it in polynomial time, such that the following holds: The algorithm always returns *false* for an input not in $\mathcal{L}_\Pi$; for an input in $\mathcal{L}_\Pi$, the algorithm returns *true* with probability greater than $1/2$. Thus, the algorithm only has to get the "right" answer in half of its guesses; this requirement is much more stringent than that of nondeterministic polynomial time, where the algorithm only had to get one guess right. Thus, many problems are known to be in $\mathcal{NP}$ but are not known to be in $\mathcal{RP}$. Whether $\mathcal{NP} = \mathcal{RP}$ is another important open question, where the common belief is also that the answer is no.

$\mathcal{RP}$

A.4 Combinatorial Optimization and Search

A.4.1 Optimization Problems

optimization problem

objective function

Many of the problems we address in this book and in other settings can be formulated as an *optimization problem*. Here, we are given a *solution space* Σ of possible solutions σ, and an *objective function* $f_{\text{obj}} : \Sigma \mapsto \mathbb{R}$ that allows us to evaluate the "quality" of each candidate solution. Our aim is then to find the solution that achieves the maximum score:

$$\sigma^* = \arg\max_{\sigma \in \Sigma} f(\sigma).$$

This optimization task is a maximization problem; we can similarly define a minimization problem, where our goal is to minimize a *loss function*. One can easily convert one problem to another (by negating the objective), and so, without loss of generality, we focus on maximization problems.

Optimization problems can be discrete, where the solution space Σ consists of a certain (finite) number of discrete hypotheses. In most such cases, this space is (at least) exponentially large in the size of the problem, and hence, for reasonably sized problems, it cannot simply be enumerated to find the optimal solution. In other problems the solution space is continuous, so that enumeration is not even an option.

The available tools for solving an optimization problem depend both on the form of the solution space Σ and on the form of the objective. For some classes of problems, we can identify the optimum in terms of a closed-form expression; for others, there exist algorithms that can provably find the optimum efficiently (in polynomial time), even when the solution space is large (or infinite); others are $\mathcal{NP}$-hard; and yet others do not (yet) have any theoretical analysis of their complexity. Throughout this book, multiple optimization problems arise, and we will see examples of all of these cases.

A.4.2 Local Search

local search

search space

search state

Many optimization problems do not appear to admit tractable solution algorithms exist, and we are forced to fall back on heuristic methods that have no guarantees of actually finding the optimal solution. One such class of methods that are in common use is the class of *local search* methods. Such search procedures operate over a *search space*. A search space is a collection of candidate solutions, often called *search states*. Each search state is associated with a score and a set of neighboring states. A search procedure is a procedure that, starting from one state, explores search space in attempt to find a high-scoring state.

search operators

Local search algorithms keep track of a "current" state. At each iteration they consider several states that are "similar" to the current one, and therefore are viewed as adjacent to it in the search space. These states are often generated by a set of *search operators*, each of which takes a state and makes a small modification to it. They select one of these neighboring states and make it the current candidate. These iterations are repeated until some termination condition. These local search procedures can be thought of as moving around in the solution space by taking small steps. Generally, these steps are taken in a direction that tends to improve the objective. If we assume that "similar" solutions tend to have similar values, this approach is likely to move toward better regions of the space.

MAP assignment

This approach can be applied to a broad range of problems. For example, we can use it to find a *MAP assignment* relative to a distribution P: the space of solutions is the set of assignments ξ to a set of random variables $\mathcal{X}$; the objective function is $P(\xi)$; and the search operators take one assignment $\boldsymbol{x}$ and change the value of one variable X_i from x_i to x_i'.

structure search

As we discuss in section 18.4, it can also be used to perform *structure search* over the space of Bayesian network structures to find one that optimizes a certain "goodness" function: the search space is the set of network structures, and the search operators make small changes to the current structure, such as adding or deleting an edge.

Algorithm A.5 Greedy local search algorithm with search operators

```
    Procedure Greedy-Local-Search (
      σ_0,    // initial candidate solution
      score,  // Score function
      O,      // Set of search operators
    )
1     σ_best ← σ_0
2     do
3       σ ← σ_best
4       Progress ← false
5       for each operator o ∈ O
6         σ_o ← o(σ)    // Result of applying o on σ
7         if σ_o is legal solution then
8           if score(σ_o) > score(σ_best) then
9             σ_best ← σ_o
10            Progress ← true
11    while Progress
12
13    return σ_best
```

A.4.2.1 Local Hill Climbing

greedy hill-climbing

One of the simplest, and often used, search procedures is the *greedy hill-climbing* procedure. As the name suggests, at each step we take the step that leads to the largest improvement in the score. This is the search analogue of a continuous gradient-ascent method; see appendix A.5.2. The actual details of the procedure are shown in algorithm A.5. We initialize the search with some solution σ_0. Then we repeatedly execute the following steps: We consider all of the solutions that are neighbors of the current one, and we compute their score. We then select the neighbor that leads to the best improvement in the score. We continue this process until no modification improves the score. One issue with this algorithm is that the number of operators that can be applied may be quite large.

first-ascent hill climbing

A slight variant of this algorithm, called *first-ascent hill climbing*, samples operators from $\mathcal{O}$ and evaluates them one at a time. Once it finds one that leads to better scoring network, it applies it without considering other operators. In the initial stages of the search, this procedure requires relatively few random trials before it finds such an

operator. As we get closer to the local maximum, most operators hurt the score, and more trials are needed before an upward step is found (if any).

What can we say about the solution returned by Greedy-Local-Search? From our stopping criterion, it follows that the score of this solution is no lower than that of its neighbors. This implies that we are in one of two situations. We might have reached a *local maximum* from which all changes are score-reducing. **Except in rare cases, there is no guarantee that the local maximum we find via local search is actually the global optimum σ^*. Indeed, it may be a very poor solution.** The other option is that we have reached a *plateau*: a large set of neighboring solutions that have the same score. By design, greedy hill-climbing procedure cannot "navigate" through a plateau, since it relies on improvement in score to guide it to better solutions. Once again, we have no guarantee that this plateau achieves the highest possible score.

local maximum

plateau

There are many modifications to this basic algorithm, mostly intended to address this problem. We now discuss some basic ideas that are applicable to all local search algorithms. We defer to the main text any detailed discussion of algorithms specific to problems of interest to us.

A.4.2.2 Forcing Exploration in New Directions

One common approach is to try to escape a suboptimal convergence point by systematically exploring the region around that point with the hope of finding an "outlet" that leads to a new direction to climb up. This can be done if we are willing to record all networks we "visited" during the search. Then, instead of choosing the best operator in Line 7 of Greedy-Local-Search, we select the best operator that leads to a solution we have not visited. We then allow the search to continue even when the score does not improve (by changing the termination condition). This variant can take steps that explore new territories even if they do not improve the score. Since it is greedy in nature, it will try to choose the best network that was not visited before. To understand the behavior of this method, visualize it climbing to the hilltop. Once there, the procedure starts pacing parts of the hill that were not visited before. As a result, it will start circling the hilltop in circles that grow wider and wider until it finds a ridge that leads to a new hill. (This procedure is often called *basin flooding* in the context of minimization problems.)

basin flooding

In this variant, even when no further progress can be made, the algorithm keeps moving, trying to find new directions. One possible termination condition is to stop when no progress has been made for some number of steps. Clearly, the final solution produced should not necessarily be the one at which the algorithm stops, but rather the best solution found anywhere during the search. Unfortunately, the computational cost of this algorithm can be quite high, since it needs to keep track of all solutions that have been visited in the past.

tabu search

Tabu search is a much improved variant of this general idea utilizes the fact that the steps in our search space take the form of local modifications to the current solution. In tabu search, we keep a list not of solutions that have been found, but rather of operators that we have recently applied. In each step, we do not consider operators that reverse the effect of operators applied within a history window of some predetermined length L. Thus, if we flip a variable X_i from x_i to x_i', we cannot flip it back in the next L steps. These restrictions force the search procedure to explore new directions in the search space, instead of tweaking with the same parts of the solution. The size L determines the amount of memory retained by the search.

The tabu search procedure is shown in algorithm A.6. The "tabu list" is the list of operators

Algorithm A.6 Local search with tabu list

```
    Procedure LegalOp (
      o,     // Search operator to check
      TABU   // List of recently applied operators

    )
1     if exists o' ∈ TABU such that o reverses o' then return false
2     else return true
3

  Procedure Tabu-Structure-Search (
      σ_0,   // initial candidate solution
      score, // Score
      𝒪,     // A set of search operators
      L,     // Size of tabu list
      N,     // Stopping criterion
    )
1     σ_best ← σ_0
2     σ ← σ_best
3     t ← 1
4     LastImprovement ← 0
5     while LastImprovement < N
6       o^(t) ← ϵ    // Set current operator to be uninitialized
7       for each operator o ∈ 𝒪    // Search for best allowed operator
8         if LegalOp(o, {o^(t−L), ..., o^(t−1)}) then
9           σ_o ← o(σ)
10          if σ_o is legal solution then
11            if o^(t) = ϵ or score(σ_o) > score(σ_{o^t}) then
12              o^(t) ← o
13      σ ← σ_{o^t}
14      if score(σ) > score(σ_best) then
15        σ_best ← σ_o
16        LastImprovement ← 0
17      else
18        LastImprovement ← LastImprovement + 1
19      t ← t + 1
20
21    return σ_best
```

applied in the last L steps. The procedure LegalOp checks if a new operator is legal given the current tabu list. The implementation of this procedure depends on the exact nature of operators we use. As in the basin-flooding approach, tabu search does not stop when it reaches a solution that cannot be improved, but rather continues the search with the hope of reaching a better structure. If this does not happen after a prespecified number of steps, we decide to abandon the search.

Algorithm A.7 Beam search

```
    Procedure Beam-Search (
      σ_0,    // initial candidate solution
      score,  // Score
      O,      // A set of search operators
      K,      // Beam width
    )
1     Beam ← {σ_0}
2     while not terminated
3       H ← ∅    // Current successors
4       for each σ ∈ L and each o ∈ O
5         Add o(σ) to H
6       Beam ← K-Best(score, H, K)
7     σ_best ← K-Best(score, H, 1)
8     return (σ_best)
```

beam search

Another variant that forces a more systematic search of the space is *beam search*. In beam search, we conduct a hill-climbing search, but we keep track of a certain fixed number K of states. The value K is called the *beam width*. At each step in the search, we take all of the current states and generate and evaluate all of their successors. The best K are kept, and the algorithm repeats. The algorithm is shown in algorithm A.7. Note that with a beam width of 1, beam search reduces to greedy hill-climbing search, and with an infinite beam width, it reduces to breadth-first search. Note that this version of beam search assumes that the (best) steps taken during the search always improve the score. If that is not the case, we would also have to compare the current states in our beam *Beam* to the new candidates in H in order to determine the next set of states to put in the beam. The termination condition can be an upper bound on the number of steps or on the improvement achieved in the last iteration.

A.4.2.3 Randomization in Search

randomization

Another approach that can help in reducing the impact of local maxima is *randomization*. Here, multiple approaches exist. We note that most randomization procedures can be applied as a wrapper to a variety of local search algorithm, including both hill climbing and tabu search. Most simply, we can initialize the algorithm at different random starting points, and then use a hill-climbing algorithm from each one. Another strategy is to interleave random steps and hill-climbing steps. Here, many strategies are possible. In one approach, we can "revitalize" the search by taking the best network found so far and applying several randomly chosen operators

Algorithm A.8 Greedy hill-climbing search with random restarts

```
    Procedure Search-with-Restarts (
      σ_0,    // initial candidate solution
      score,    // Score
      O,    // A set of search operators
      Search,    // Search procedure
      l,    // random restart length
      k    // number of random restarts
    )
1     σ_best ← Search(σ_0, score, O)
2     for i = 1, ..., k
3       σ ← σ_best
4         // Perform random walk
5       j ← 1
6       while j < l
7         sample o from O
8         if o(σ) is a legal network then
9           σ ← o(σ)
10          j ← j + 1
11      σ ← Search(σ, score, O)
12      if score(σ) > score(σ_best) then
13        σ_best ← σ
14
15    return σ_best
```

random restart

to get a network that is fairly similar, yet perturbed. We then restart our search procedure from the new network. If we are lucky, this *random restart* step moves us to a network that belongs to a better "basin of attraction," and thus the search will converge to a better structure. A simple random restart procedure is shown in algorithm A.8; it can be applied as wrapper to plain hill climbing, tabu search, or any other search algorithm.

This approach can be effective in escaping from fairly local maxima (which can be thought of as small bumps on the slope of a larger hill). However, it is unlikely to move from one wide hill to another. There are different choices in applying random restart, the most important one is how many random "steps" to take. If we take too few, we are unlikely to escape the local maxima. If we take too many, than we move too far off from the region of high scoring network. One possible strategy is to applying random restarts of growing magnitude. That is, each successive random restart applies more random operations.

To make this method concrete, we need a way of determining how to apply random restarts, and how to interleave hill-climbing steps and randomized moves. A general framework for doing is *simulated annealing*. The basic idea of simulated annealing is similar to Metropolis-Hastings MCMC methods that we discuss in section 12.3, and so we only briefly touch it.

simulated annealing

In broad outline, the simulated annealing procedure attempts to mix hill-climbing steps with

temperature parameter

moves that can decrease the score. This mixture is controlled by a so-called *temperature parameter*. When the temperature is "hot," the search tries many moves that decrease the score. As the search is annealed (the temperature is slowly reduced) it starts to focus only on moves that improve the score. The intuition is that during the "hot" phase the search explores the space and eventually gets trapped in a region of high scores. As the temperature reduces it is able to distinguish between finer details of the score "landscape" and eventually converge to a good maximum.

proposal distribution

To carry out this intuition, a simulated annealing procedure uses a *proposal distribution* over operators to propose candidate operators to apply in the search. At each step, the algorithm selects an operator o using this distribution, and evaluates $\delta(o)$ — the change in score incurred by applying o at the current state. The search accepts this move with probability $\min(1, e^{\frac{\delta(o)}{\tau}})$, where τ is the current temperature. Note that, if $\delta(o) > 0$, the move is automatically accepted. If $\delta(o) < 0$, the move is accepted with probability that depends both on the decrease in score and on the temperature τ. For large value of τ (hot) all moves are applied with probability close to 1. For small values of τ (cold), all moves that decrease the score are applied with small probability. The search procedure anneals τ every fixed number of move attempts. There are various strategies for annealing; the simplest one is simply to have τ decay exponentially. One can actually show that, if the temperature is annealed sufficiently slowly, simulated annealing converges to the globally optimal solution with high probability. However, in practice, this "guaranteed" annealing schedule is both unknown and much too slow to be useful in practice. **In practice, the success of simulated annealing depends heavily on the design of the proposal distribution and annealing schedule.**

A.4.3 Branch and Bound Search

Here we discussed one class of solutions to discrete optimization problems: the class of local hill-climbing search. Those methods are very broadly useful, since they apply to any discrete optimization problem for which we can define a set of search operators. In some cases, however, we may know some additional structure within the problem, allowing more informed methods to be applied. One useful type of information is a mechanism that allows us to evaluate a partial assignment $\boldsymbol{y}_{1...i}$, and to place a bound $\text{bound}(\boldsymbol{y}_{1...i})$ on the best score of any complete assignment that extends $\boldsymbol{y}_{1...i}$. In this case, we can use an algorithm called *branch and bound search*, shown in algorithm A.9 for the case of a maximization problem.

branch and bound

Roughly speaking, branch and bound searches the space of partial assignments, beginning with the empty assignment, and assigning the variables $X_1, \ldots, X_n$, one at a time (in some order), using depth-first search. At each point, when considering the current partial assignment $\boldsymbol{y}_{1...i}$, the algorithm evaluates it using $\text{bound}(\boldsymbol{y}_{1...i})$ and compares it to the best full assignment ξ found so far. If $\text{score}(\xi)$ is better than the best score that can possibly be achieved starting from $\boldsymbol{y}_{1...i}$, then there is no point continuing to explore any of those assignments, and the algorithm backtracks to try a different partial assignment. Because the bound is correct, it is not difficult to show that the assignments that were *pruned* without being searched cannot possibly be optimal. When the bound is reasonably tight, this algorithm can be very effective, pruning large parts of the space without searching it.

The algorithm shows the simplest variant of the branch-and-bound procedure, but many extensions exist. One heuristic is to perform the search so as to try and find good assignments

Algorithm A.9 Branch and bound algorithm

```
    Procedure Branch-and-Bound (
      score,    // Score function
      bound,    // Upper bound function
      σ_best,   // Best full assignment so far
      score_best,   // Best score so far
      i,    // Variable to be assigned next
      y_{1...i-1},   // Current partial assignment
    )   // Recursive algorithm, called initially with the following arguments: some arbitrary full assignment σ_best, score_best = score(σ_best), i = 1, and the empty assignment.
1     for each x_i ∈ Val(X_i)
2       y_{1...i} ← (y_{1...i-1}, x_i)    // Extend the assignment
3       if i = n and score(y_{1...n}) > score_best then
4         (σ_best, score_best) ← (y_{1...n}, score(y_{1...n}))
5           // Found a better full assignment
6       else if bound(y_{1...i}) > score_best then
7         (σ_best, score_best) ← Branch-and-Bound(score, bound, σ_best, score_best, i+1, y_{1...i})
8           // If bound is better than current solution, try current partial
              assignment; otherwise, prune and move on
9     return (σ_best, score_best)
```

early. The better the current assignment σ_{best}, the better we can prune suboptimal trajectories. Other heuristics intelligently select, at each point in the search, which variable to assign next, allowing this choice to vary across different points in the search. When available, one can also use a lower bound as well as an upper bound, allowing pruning to take place based on partial (not just full) trajectories. Many other extensions exist, but are outside the scope of this book.

A.5 Continuous Optimization

In the preceding section, we discussed the problem of optimizing an objective over a discrete space. In this section we briefly review methods for solving optimization problems over a *continuous* space. See Avriel (2003); Bertsekas (1999) for more thorough discussion of nonlinear optimization, and see Boyd and Vandenberghe (2004) for an excellent overview of convex optimization methods.

A.5.1 Characterizing Optima of a Continuous Function

At several points in this book we deal with maximization (or minimization) problems. In these problems, we have a function $f_{\text{obj}}(\theta_1, \ldots, \theta_n)$ for several *parameters*, and we wish to find joint values of the parameters that maximizes the value of f_{obj}.

Formally, we face the following problem:

Find values $\theta_1, \ldots, \theta_n$ such that $f_{\text{obj}}(\theta_1, \ldots, \theta_n) = \max_{\theta'_1, \ldots, \theta'_n} f_{\text{obj}}(\theta'_1, \ldots, \theta'_n)$.

Example A.2

Assume we are given a set of points $(x[1], y[1]), \ldots, (x[m], y[m])$*. Our goal is to find the "centroid" of these points, defined as a point* (θ_x, θ_y) *that minimizes the square distance to all of the points. We can formulate this problem into a maximization problem by considering the negative of the sum of squared distances:*

$$f_{\text{obj}}(\theta_x, \theta_y) = -\sum_i \left((x[i] - \theta_x)^2 + (y[i] - \theta_y)^2\right).$$

■

gradient

One way of finding the maximum of a function is to use the fact that, at the maximum, the *gradient* of the function is 0. Recall that the gradient of a function $f_{\text{obj}}(\theta_1, \ldots, \theta_n)$ is the vector of partial derivatives

$$\nabla f = \left\langle \frac{\partial f}{\partial \theta_1}, \cdots \frac{\partial f}{\partial \theta_n} \right\rangle.$$

Theorem A.5

If $\langle \theta_1, \ldots, \theta_n \rangle$ *is an interior maximum point of* f_{obj}*, then*

$$\nabla f(\theta_1, \ldots, \theta_n) = 0.$$

stationary point

This property, however, does not characterize maximum points. Formally, a point $\langle \theta_1, \ldots, \theta_n \rangle$ where $\nabla f_{\text{obj}}(\theta_1, \ldots, \theta_n) = 0$ is a *stationary point* of f_{obj}. Such a point can be either a local maximum, a local minimum, or a saddle point. However, finding such a point can often be the first step toward finding a maximum.

To satisfy the requirement that $\nabla f = 0$ we need to solve the set of equations

$$\frac{\partial}{\partial \theta_k} f_{\text{obj}}(\theta_1, \ldots, \theta_n) = 0 \qquad k = 1, \ldots, n$$

Example A.3

Consider the task of example A.2. We can easily verify that:

$$\frac{\partial}{\partial \theta_x} f_{\text{obj}}(\theta_x, \theta_y) = 2 \sum_i (x[i] - \theta_x).$$

Equating this term to 0 *and performing simple arithmetic manipulations, we get the equation:*

$$\theta_x = \frac{1}{m} \sum_i x[i].$$

The exact same reasoning allows us to solve for θ_y*.* ■

In this example, we conclude that f_{obj} has a unique stationary point. We next need to verify that this point is a maximum point (rather than a minimum or a saddle point). In our example, we can check that, for any sequence that extends from the origin to infinity (that is, $\theta_x^2 + \theta_y^2 \to \infty$), we have $f_{\text{obj}} \to -\infty$. Thus, the single stationary point is a maximum.

In general, to verify that the stationary point is a maximum, we can check that the second derivative is negative. To see this, recall the multivariate Taylor expansion of degree two:

$$f_{\text{obj}}(\vec{\theta}) = f_{\text{obj}}(\vec{\theta}_0) + (\vec{\theta} - \vec{\theta}_0)^T \nabla f_{\text{obj}}(\vec{\theta}_0) + \frac{1}{2} \left[\vec{\theta} - \vec{\theta}_0\right]^T A(\vec{\theta}_0) \left[\vec{\theta} - \vec{\theta}_0\right],$$

Hessian

where $A(\vec{\theta}_0)$ is the *Hessian* — the matrix of second derivatives at $\vec{\theta}_0$. If we use this expansion around a stationary point, then the gradient is 0, and we only need to examine the term $[\vec{\theta} - \vec{\theta}_0]^T A(\vec{\theta}_0)[\vec{\theta} - \vec{\theta}_0]$. In the univariate case, we can verify that a point is a local maximum by testing that the second derivative is negative. The analogue to this condition in the multivariate

negative definite

case is that $A(\vec{\theta}_0)$ is *negative definite* at the point $\vec{\theta}_0$, that is, that $\vec{\theta}^T A(\vec{\theta}_0)\vec{\theta} < 0$ for all $\vec{\theta}$.

Theorem A.6

Suppose $\vec{\theta} = \langle \theta_1, \ldots, \theta_n \rangle$ is an interior point of f_{obj} with $\nabla \vec{\theta} = 0$. Let $A(\vec{\theta}) = \{\frac{\partial^2}{\partial \theta_i \partial \theta_j} f_{\text{obj}}(\vec{\theta})\}$ be the Hessian *matrix of second derivatives of f_{obj} at $\vec{\theta}$. $A(\vec{\theta})$ is negative definite at θ if and only if $\vec{\theta}$ is a local maximum of f_{obj}.*

Example A.4

Continuing example A.2, we can verify that

$$\begin{aligned}
\frac{\partial^2}{\partial^2_{\theta_x}} f_{\text{obj}}(\theta_x, \theta_y) &= -2m \\
\frac{\partial^2}{\partial^2_{\theta_y}} f_{\text{obj}}(\theta_x, \theta_y) &= -2m \\
\frac{\partial^2}{\partial_{\theta_y} \partial_{\theta_x}} f_{\text{obj}}(\theta_x, \theta_y) &= 0 \\
\frac{\partial^2}{\partial_{\theta_x} \partial_{\theta_y}} f_{\text{obj}}(\theta_x, \theta_y) &= 0.
\end{aligned}$$

Thus, the Hessian matrix is simply

$$A = \begin{pmatrix} -2m & 0 \\ 0 & -2m \end{pmatrix}.$$

It is easy to verify that this matrix is negative definite. ■

A.5.2 Gradient Ascent Methods

The characterization of appendix A.5.1 allows us to provide closed-form solutions for certain continuous optimization problems. However, there are many problems for which such solutions cannot be found. In these problems, the equations posed by $\nabla f_{\text{obj}}(\boldsymbol{\theta}) = 0$ do not have an analytical solution. Moreover, in many practical problems, there are multiple local maxima, and then this set of equations does not even have a unique solution.

One approach for dealing with problems that do not yield analytical solutions is to *search* for a (local) maximum. The idea is very analogous to the discrete local search of appendix A.4.2: We begin with an initial point $\boldsymbol{\theta}^0$, which can be an arbitrary choice, a random guess, or an approximation of the solution based on other considerations. From this starting point, we want to "climb" to a maximum. A great many techniques roughly follow along these lines. In this section, we survey some of the most common ones.

A.5.2.1 Gradient Ascent

gradient ascent

The simplest approach is *gradient ascent*, an approach directly analogous to the hill-climbing