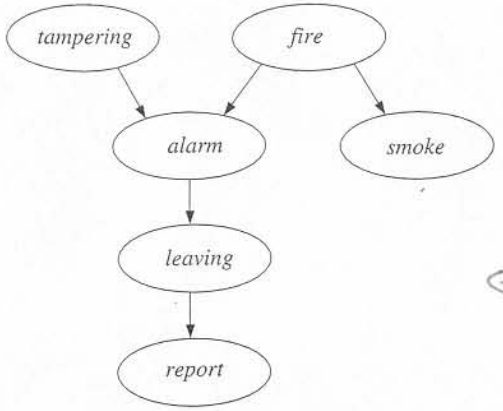

Figure 10.2: Belief network for report of leaving of Example 10.16

applicable, even if all of the parents of a are also part of e. If e specifies a value for a—for example, contains $a = v_1$—then that value is all that is needed to determine the probability of a. Otherwise, suppose e is a conjunction $e_1 \wedge e_2$ where e_1 involves only descendants of a, and e_2 contains no descendants of a. Bayes' rule can be used to compute $P(a|e)$:

$$P(a|e_1 \wedge e_2) = \frac{P(e_1|a \wedge e_2) \times P(a|e_2)}{P(e_1|e_2)}.$$

The right-hand side of this equation has no instance of conditioning on a descendant: Neither a nor e_2 involves a descendant of e_1, and e_2 doesn't involve a descendant of a.

Example 10.16

Suppose you want to use the diagnostic assistant to diagnose whether there is a fire in a building, based on noisy sensor information and possibly conflicting explanations of what could be going on. The agent receives a report about whether everyone is leaving the building. Suppose the report sensor is noisy: It sometimes reports leaving when there is no exodus, a false positive, and sometimes doesn't report when everyone is leaving, a false negative. Suppose the fire alarm going off can cause the leaving. Either tampering or fire could cause the alarm. Fire also causes smoke to rise from the building. The belief network of Figure 10.2 can express such knowledge.

The variable *report* denotes the sensor report that people are leaving. This variable is introduced to allow conditioning on unreliable sensor data. The infobot

knows what the sensor reports, but it only has unreliable evidence about people leaving the building. It can condition only on what it knows—in this case, on what the sensor reports.

As well as the graph depicted in Figure 10.2, you also need to specify the domain of each variable and the conditional probabilities of each variable given each assignment of values to its parent. For this example, assume that the variables are all Boolean, with the following probabilities:

$P(alarm|fire \wedge tampering) = 0.5$
$P(alarm|fire \wedge \neg tampering) = 0.99$
$P(alarm|\neg fire \wedge tampering) = 0.85$
$P(alarm|\neg fire \wedge \neg tampering) = 0.0001$
$P(smoke|fire) = 0.9$
$P(smoke|\neg fire) = 0.01$
$P(leaving|alarm) = 0.88$
$P(leaving|\neg alarm) = 0.001$
$P(report|leaving) = 0.75$
$P(report|\neg leaving) = 0.01$
$P(fire) = 0.01$
$P(tampering) = 0.02$

This is the network used in the example on the next page

The probabilities of a variable given nondescendants can be computed using the "reasoning by cases" rule (page 352). The probability of people leaving the building, given there is smoke, can be derived using

$$\begin{aligned} & P(leaving|smoke) \\ = \; & P(leaving|alarm \wedge smoke) \times P(alarm|smoke) \\ & + P(leaving|\neg alarm \wedge smoke) \times (1 - P(alarm|smoke)) \\ = \; & P(leaving|alarm) \times P(alarm|smoke) \\ & + P(leaving|\neg alarm) \times (1 - P(alarm|smoke)). \end{aligned}$$

The probabilities $P(leaving|alarm)$ and $P(leaving|\neg alarm)$ are provided as part of the belief network.

It remains to compute $P(alarm|smoke)$. A case analysis on $P(alarm|smoke)$ gives

$$\begin{aligned} & P(alarm|smoke) \\ = \; & P(alarm|fire \wedge tampering) \times P(fire \wedge tampering|smoke) \\ & + P(alarm|fire \wedge \neg tampering) \times P(fire \wedge \neg tampering|smoke) \\ & + P(alarm|\neg fire \wedge tampering) \times P(\neg fire \wedge tampering|smoke) \end{aligned}$$

is to add an extra node that indicates that the model is appropriate. Arcs from this node would lead to each variable representing power in a wire and to each light. When the model is appropriate, you can use the probabilities of Example 10.15 (page 368). When the model is inappropriate, you can, for example, specify that each wire and light works at random. When there are weird observations that don't fit in with the original model—they are impossible or extremely unlikely given the model—the probability that the model is inappropriate will increase.

This network can be used in a number of ways, for example:

- By conditioning on the knowledge that the switches and circuit breakers are ok, and on the values of the outside power and the position of the switches, you can use this network to simulate how the lighting should work.
- Given values of the outside power and the position of the switches, you can determine how likely any outcome is—for example, how likely it is that l_1 is lit.
- Given values for the switches and whether the lights are lit, you can determine the posterior probability that each switch or circuit breaker is in any particular state.
- Given some observations, you can use the network to reason backwards to determine the most likely position of switches.
- Given some switch positions and some outputs and some intermediate values, you can determine the probability of any other variable in the network.

Implementing Belief Networks

The problem of determining posterior distributions—the problem of computing conditional probabilities given the evidence—is one that has been widely researched.

The problem of estimating the posterior probability in a belief network within an absolute error (of less than 0.5), or within a constant factor, is NP-hard, so general efficient implementations will not be available.

Three main approaches are taken to implement belief networks:

- Exploiting the structure of the network. This approach is typified by the *clique tree propagation method*, where the network is transformed into a tree with nodes labeled with sets of variables. Evidential reasoning is carried out by passing messages between the nodes in the tree. The values passed between the nodes are the distributions on the variables in common between the nodes. These distributions on a subset of the variables are called **marginal distributions**. One piece of evidence (an observation) can be entered with time complexity linear in the size of the tree. The size of the tree may, however, be exponential in the size of the belief network, but it's small when there are few multiple paths between nodes the belief networks. A related approach that exploits structure is detailed below.
- Search-based approaches. You enumerate some of the possible worlds, and you estimate posterior probabilities from the worlds generated. You can bound the probability mass of the worlds not considered, and you use this bound to estimate the error in the posterior probability. This approach works well when the distributions are extreme (all probabilities are close to zero or close to one).
- Stochastic simulation. In these approaches, random cases are generated according to the probability distributions. By treating these random cases as a set of samples, you can estimate the marginal distribution on any combination of variables.

The following gives a general algorithm for exploiting structure. Many of the efficient methods can be seen as optimizations of this algorithm.

An Algorithm For Evaluating Belief Networks

This section gives an algorithm for finding the posterior distribution for a variable in an arbitrarily structured belief network. The algorithm is based on the notion that a belief network specifies a factorization of the joint probability distribution (page 368). A Prolog implementation of this algorithm is given on page 521.

Before we give the algorithm, we need to define factors and the operations that will be performed on them.

A **factor** is a representation of a function from a tuple of random variables into a number. We will write factor f on variables $x_1, \ldots, x_j$ as $f(x_1, \ldots, x_j)$. The variables $x_1, \ldots, x_j$ are the variables **of** the factor f, and f is a factor **on** $x_1, \ldots, x_j$.

Suppose $f(x_1, \ldots, x_j)$ is a factor and each v_i is an element of the domain of x_i. $f(x_1 = v_1, x_2 = v_2, \ldots, x_j = v_j)$ is a number that is the value of f when each x_i has value v_i. You can assign some of the variables of a factor and make a new factor. For example, $f(x_1 = v_1, x_2, \ldots, x_j)$, sometimes written as $f(x_1, x_2, \ldots, x_j)_{x_1 = v_1}$, where v_1 is an element of the domain of variable x_1, is a factor on $x_2, \ldots, x_j$.

You can multiply factors together. Suppose f_1 and f_2 are factors, where f_1 is a factor that contains variables $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$, and f_2 is a factor with variables $y_1, \ldots, y_j$ and $z_1, \ldots, z_k$, where $y_1, \ldots, y_j$ are the variables in common to f_1 and f_2. The **product** of f_1 and f_2 is a factor on the union of the variables, namely $x_1, \ldots, x_i, y_1, \ldots, y_j, z_1, \ldots, z_k$, defined by:

$$(f_1 \times f_2)(x_1, \ldots, x_i, y_1, \ldots, y_j, z_1, \ldots, z_k) = f_1(x_1, \ldots, x_i, y_1, \ldots, y_j) f_2(y_1, \ldots, y_j, z_1, \ldots, z_k).$$

You can sum out a variable in a factor. Given factor $f(x_1, \ldots, x_j)$, summing out a variable, say x_1, results in a factor on $x_2, \ldots, x_j$ defined by:

$$(\sum_{x_1} f)(x_2, \ldots, x_j) = f(x_1 = v_1, \ldots, x_j) + \cdots + f(x_1 = v_k, \ldots, x_j)$$

where $\{v_1, \ldots, v_k\}$ is the set of possible values of variable x_1.

Given this definition, a conditional probability distribution $P(x|y_1, \ldots, y_j)$ can be seen as a factor f on $x, y_1, \ldots, y_j$, where

$$f(x = u, y_1 = v_1, \ldots, y_j = v_j) = P(x = u | y_1 = v_1 \wedge \cdots \wedge y_j = v_j).$$

Usually we will use the $P(\cdot|\cdot)$ notation rather than use f. The fact that it's a probability means that for all values u and $v_1, \ldots, v_j$,

$$P(x = u | y_1 = v_1, \ldots, y_j = v_j) \geq 0 \text{ and}$$
$$\sum_x P(x | y_1 = v_1, \ldots, y_j = v_j) = 1.$$

The **belief network inference problem** is the problem of computing the posterior distribution of a variable given some evidence.

The problem of computing posterior probabilities can be reduced to the problem of computing the probability of conjunctions. Given evidence $y_1 = v_1, \ldots, y_j = v_j$, and query variable z:

$$\begin{aligned} & P(z | y_1 = v_1, \ldots, y_j = v_j) \\ &= \frac{P(z, y_1 = v_1, \ldots, y_j = v_j)}{P(y_1 = v_1, \ldots, y_j = v_j)} \\ &= \frac{P(z, y_1 = v_1, \ldots, y_j = v_j)}{\sum_z P(z, y_1 = v_1, \ldots, y_j = v_j)}. \end{aligned}$$

So all you need to do is compute the factor $P(z, y_1 = v_1, \ldots, y_j = v_j)$ and normalize. Note that this is a factor only of z; given a value for z, this returns a number that is the probability of the conjunction of the propositions.

Suppose the variables of the belief network are $x_1, \ldots, x_n$. To compute the factor $P(z, y_1 = v_1, \ldots, y_j = v_j)$, you can sum out the other variables from the joint distribution. Suppose $z_1, \ldots, z_k$ is an enumeration of the other variables in the belief network—that is,

$$\{z_1, \ldots, z_k\} = \{x_1, \ldots, x_n\} - \{z\} - \{y_1, \ldots, y_j\}.$$

You can construct the desired factor by summing out the z_i. The order of the z_i is an **elimination ordering**.

$$P(z, y_1 = v_1, \ldots, y_j = v_j) = \sum_{z_k} \cdots \sum_{z_1} P(x_1, \ldots, x_n)_{y_1 = v_1, \ldots, y_j = v_j}.$$

Note how this is related to the semantics of probability (page 349): There is a possible world for each assignment of a value to each variable. The **joint probability distribution**, $P(x_1, \ldots, x_n)$ gives the probability (or measure) for each possible world. All you are doing is selecting the worlds with the observed values for the y_i's and summing over the possible worlds with the same value for z. This is the definition of conditional probability (page 353).

By the rule for conjunction of probabilities and the definition of a belief network,

$$P(x_1, \ldots, x_n) = P(x_1 | \pi_{x_1}) \times \cdots \times P(x_n | \pi_{x_n}),$$

where π_{x_i} is the set of parents of variable x_i.

We have now reduced the belief network inference problem to a problem of summing out a set of variables from a product of factors. To compute the posterior distribution of a query variable given observations:

1. Construct the joint probability distribution in terms of a product of factors.
2. Set the observed variables to their observed values.
3. Sum out each of the other variables (the $\{z_1, \ldots, z_k\}$).
4. Multiply the remaining factors and normalize.

To sum out a variable z from a product $f_1, \ldots, f_k$ of factors, you first partition the factors into those that don't contain z, say $f_1, \ldots, f_i$, and those that contain $z, f_{i+1}, \ldots, f_k$; then

$$\sum_z f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \times \left(\sum_z f_{i+1} \times \cdots \times f_k \right).$$

You explicitly construct a representation (in terms of a multidimensional array, a tree, or a set of rules) of the rightmost factor. The factor's size is exponential in the number of variables of the factor.

Example 10.19

Consider Example 10.16 (page 370). Here two-letter abbreviations are used for the variables. The joint probability distribution is given by

$$\begin{aligned} & P(ta, fi, sm, al, le, re) \\ &= P(ta) \times P(fi) \times P(sm|fi) \times P(al|ta, fi) \times P(le|al) \times P(re|le). \end{aligned}$$

Suppose that each variable has possible values $\{yes, no\}$. Given the query

$$P(ta | sm = yes \wedge re = yes)$$

and the elimination ordering fi, al, le, use the following equation:

$$\begin{aligned} & P(ta \wedge sm = yes \wedge re = yes) \\ &= \sum_{le} \sum_{al} \sum_{fi} P(ta) \times P(fi) \times P(sm = yes|fi) \times \\ & \quad P(al|ta, fi) \times P(le|al) \times P(re = yes|le) \end{aligned}$$

You first sum out fi:

$$\begin{aligned}
&\sum_{fi} P(ta) \times P(fi) \times P(sm = yes|fi) \times \\
&\quad P(al|ta, fi) \times P(le|al) \times P(re = yes|le) \\
&= P(ta) \times P(le|al) \times P(re = yes|le) \times \\
&\quad \sum_{fi} P(fi) \times P(sm = yes|fi) \times P(al|ta, fi) \\
&= P(ta) \times P(le|al) \times P(re = yes|le) \times f_1(al, ta).
\end{aligned}$$

f_1 is a newly created factor. f_1 only depends on al and ta. For each combination of values of these variables there is a number obtained by summing the product $P(fi) \times P(sm = yes|fi) \times P(al|ta, fi)$ for each value of fi. Note how the factor's size depends on how many variables are connected to the summed-out variable.

Next sum out al:

$$\begin{aligned}
&\sum_{al} P(ta) \times P(le|al) \times P(re = yes|le) \times f_1(al, ta) \\
&= P(ta) \times P(re = yes|le) \times \sum_{al} P(le|al) \times f_1(al, ta) \\
&= P(ta) \times P(re = yes|le) \times f_2(le, ta),
\end{aligned}$$

where f_2 is a newly created factor that depends on le and ta.

You next sum out le:

$$\begin{aligned}
&\sum_{le} P(ta) \times P(re = yes|le) \times f_2(le, ta) \\
&= P(ta) \times \sum_{le} P(re = yes|le) \times f_2(le, ta) \\
&= P(ta) \times f_3(ta),
\end{aligned}$$

where f_3 is a newly created factor that depends on ta.

The posterior distribution on ta can be computed by multiplying these factors and normalizing.

Modern exact algorithms use what is essentially this method, and they speed it up by preprocessing as much as possible into a secondary structure before any evidence arrives. This is appropriate when, for example, the same belief network may be used for many different cases. They save intermediate results so that evidence can be incrementally added and so that each variable's probability can be derived after each addition of evidence.

This algorithm can be speeded up by pruning the irrelevant nodes from the network before the query starts. An approximation to what is relevant can be stated as follows: The query node is relevant, ancestors of relevant nodes are relevant, and observed descendants of relevant nodes are relevant. All other nodes are irrelevant. This will not remove relevant nodes but misses some irrelevant nodes. For example, if le were observed and the query variable were re, all other variables really are irrelevant. A more detailed specification is left as an exercise.

Unfortunately, extensive preprocessing, allowing arbitrary sequences of observations and deriving the posterior on each variable, precludes pruning the network. So for each application you need to choose whether you will save more by pruning irrelevant variables for each query and observation or by preprocessing before you have any observations.

up to here.

10.4 Making Decisions Under Uncertainty

The main reason you need probabilities is to make decisions under uncertainty. An agent's decision on what to do depends on two things:

- *What the agent believes.* You may be tempted to say "what is true in the world," but when an agent doesn't know what is true in the world, it can act based only on its beliefs. Sensing the world updates the agent's beliefs by conditioning on what is sensed.
- *The agent's goals.* When an agent has to reason under uncertainty, it has to consider not only what will most likely happen but also what may happen. Some possible outcomes may have much worse consequences than others. The notion of a "goal" here needs to be richer than the goals considered in Chapter 8 because you must specify the tradeoffs between different outcomes. For example, if some action results in a good outcome most of the time, but sometimes results in a disastrous outcome, it needs to be compared with doing an alternative action that results in the good outcome less often and the disastrous outcome less often. Decision theory specifies how to trade off the desirability of outcomes with the probabilities.

Example 10.20

Consider the problem of the delivery robot when there is uncertainty in the outcome of its actions. In particular, consider the problem of going from position $o109$ in Figure 8.1 (page 285) to the *mail* position, where there is a chance that the robot will slip off course and fall down the stairs. Suppose that you can get pads for the robot that won't change the probability of an accident, but will make it less severe. Unfortunately, the pads add extra weight. The robot could also go the long way around, which would reduce the probability of an accident but make the trip much slower.

Figure 10.4 shows a **decision tree** that depicts the different choices and outcomes. To read the decision tree, you start from the left. From each node one of