values. The structure of this computation is shown in Figure 14.8. Using the numbers from Figure 14.2, we obtain $P(b \mid j, m) = \alpha \times 0.00059224$. The corresponding computation for $\neg b$ yields $\alpha \times 0.0014919$; hence,

$$\mathbf{P}(B \mid j, m) = \alpha \langle 0.00059224, 0.0014919 \rangle \approx \langle 0.284, 0.716 \rangle .$$

That is, the chance of a burglary, given calls from both neighbors, is about 28%.

The evaluation process for the expression in Equation (14.4) is shown as an expression tree in Figure 14.8. The ENUMERATION-ASK algorithm in Figure 14.9 evaluates such trees using depth-first recursion. The algorithm is very similar in structure to the backtracking algorithm for solving CSPs (Figure 6.5) and the DPLL algorithm for satisfiability (Figure 7.17).

The space complexity of ENUMERATION-ASK is only linear in the number of variables: the algorithm sums over the full joint distribution without ever constructing it explicitly. Unfortunately, its time complexity for a network with n Boolean variables is always $O(2^n)$—better than the $O(n\,2^n)$ for the simple approach described earlier, but still rather grim.

Note that the tree in Figure 14.8 makes explicit the *repeated subexpressions* evaluated by the algorithm. The products $P(j \mid a)P(m \mid a)$ and $P(j \mid \neg a)P(m \mid \neg a)$ are computed twice, once for each value of e. The next section describes a general method that avoids such wasted computations.

14.4.2 The variable elimination algorithm

The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in Figure 14.8. The idea is simple: do the calculation once and save the results for later use. This is a form of dynamic programming. There are several versions of this approach; we present the **variable elimination** algorithm, which is the simplest. Variable elimination works by evaluating expressions such as Equation (14.4) in *right-to-left* order (that is, *bottom up* in Figure 14.8). Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.

VARIABLE ELIMINATION

Let us illustrate this process for the burglary network. We evaluate the expression

$$\mathbf{P}(B \mid j, m) = \alpha \underbrace{\mathbf{P}(B)}_{\mathbf{f}_1(B)} \sum_e \underbrace{P(e)}_{\mathbf{f}_2(E)} \sum_a \underbrace{\mathbf{P}(a \mid B, e)}_{\mathbf{f}_3(A,B,E)} \underbrace{P(j \mid a)}_{\mathbf{f}_4(A)} \underbrace{P(m \mid a)}_{\mathbf{f}_5(A)} .$$

Notice that we have annotated each part of the expression with the name of the corresponding **factor**; each factor is a matrix indexed by the values of its argument variables. For example, the factors $\mathbf{f}_4(A)$ and $\mathbf{f}_5(A)$ corresponding to $P(j \mid a)$ and $P(m \mid a)$ depend just on A because J and M are fixed by the query. They are therefore two-element vectors:

FACTOR

$$\mathbf{f}_4(A) = \begin{pmatrix} P(j \mid a) \\ P(j \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix} \qquad \mathbf{f}_5(A) = \begin{pmatrix} P(m \mid a) \\ P(m \mid \neg a) \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.01 \end{pmatrix} .$$

$\mathbf{f}_3(A, B, E)$ will be a $2 \times 2 \times 2$ matrix, which is hard to show on the printed page. (The "first" element is given by $P(a \mid b, e) = 0.95$ and the "last" by $P(\neg a \mid \neg b, \neg e) = 0.999$.) In terms of factors, the query expression is written as

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A)$$

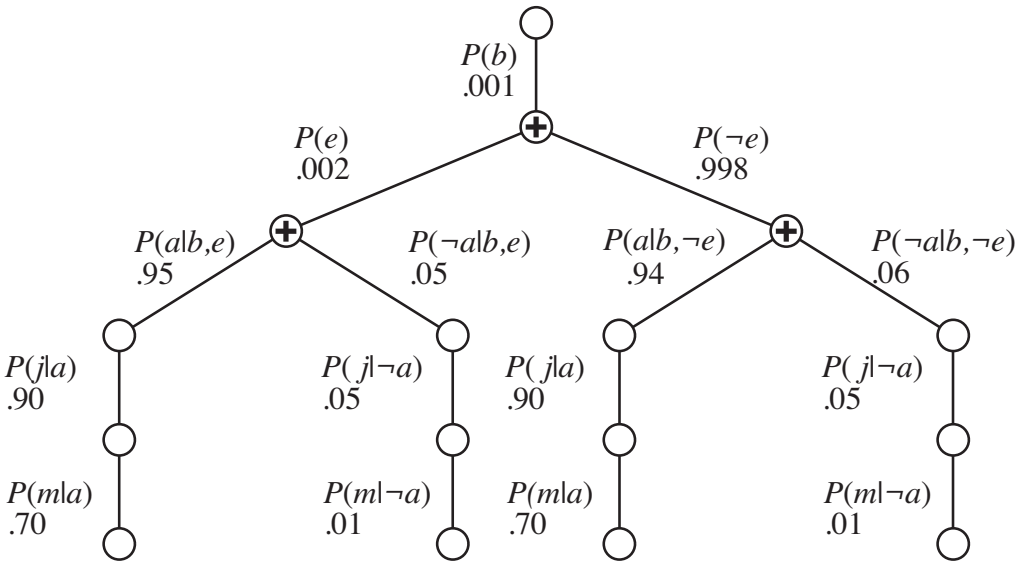

Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

function ENUMERATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayes net with variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$ /* **Y** = *hidden variables* */

  $\mathbf{Q}(X) \leftarrow$ a distribution over X, initially empty
  for each value x_i of X **do**
    $\mathbf{Q}(x_i) \leftarrow$ ENUMERATE-ALL(bn.VARS, $\mathbf{e}_{x_i}$)
      where $\mathbf{e}_{x_i}$ is **e** extended with $X = x_i$
  return NORMALIZE($\mathbf{Q}(X)$)

---

function ENUMERATE-ALL($vars$, **e**) **returns** a real number
  if EMPTY?($vars$) **then return** 1.0
  $Y \leftarrow$ FIRST($vars$)
  if Y has value y in **e**
    then return $P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), **e**)
    else return $\sum_y P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_y$)
      where $\mathbf{e}_y$ is **e** extended with $Y = y$

Figure 14.9 The enumeration algorithm for answering queries on Bayesian networks.

POINTWISE PRODUCT

where the "$\times$" operator is not ordinary matrix multiplication but instead the **pointwise product** operation, to be described shortly.

The process of evaluation is a process of summing out variables (right to left) from pointwise products of factors to produce new factors, eventually yielding a factor that is the solution, i.e., the posterior distribution over the query variable. The steps are as follows:

- First, we sum out A from the product of $\mathbf{f}_3$, $\mathbf{f}_4$, and $\mathbf{f}_5$. This gives us a new 2×2 factor $\mathbf{f}_6(B, E)$ whose indices range over just B and E:

$$\begin{aligned}\mathbf{f}_6(B, E) &= \sum_a \mathbf{f}_3(A, B, E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) \\ &= (\mathbf{f}_3(a, B, E) \times \mathbf{f}_4(a) \times \mathbf{f}_5(a)) + (\mathbf{f}_3(\neg a, B, E) \times \mathbf{f}_4(\neg a) \times \mathbf{f}_5(\neg a)) \,.\end{aligned}$$

  Now we are left with the expression

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \,.$$

- Next, we sum out E from the product of $\mathbf{f}_2$ and $\mathbf{f}_6$:

$$\begin{aligned}\mathbf{f}_7(B) &= \sum_e \mathbf{f}_2(E) \times \mathbf{f}_6(B, E) \\ &= \mathbf{f}_2(e) \times \mathbf{f}_6(B, e) + \mathbf{f}_2(\neg e) \times \mathbf{f}_6(B, \neg e) \,.\end{aligned}$$

  This leaves the expression

$$\mathbf{P}(B \mid j, m) = \alpha\, \mathbf{f}_1(B) \times \mathbf{f}_7(B)$$

  which can be evaluated by taking the pointwise product and normalizing the result.

Examining this sequence, we see that two basic computational operations are required: pointwise product of a pair of factors, and summing out a variable from a product of factors. The next section describes each of these operations.

Operations on factors

The pointwise product of two factors $\mathbf{f}_1$ and $\mathbf{f}_2$ yields a new factor $\mathbf{f}$ whose variables are the *union* of the variables in $\mathbf{f}_1$ and $\mathbf{f}_2$ and whose elements are given by the product of the corresponding elements in the two factors. Suppose the two factors have variables $Y_1, \ldots, Y_k$ in common. Then we have

$$\mathbf{f}(X_1 \ldots X_j, Y_1 \ldots Y_k, Z_1 \ldots Z_l) = \mathbf{f}_1(X_1 \ldots X_j, Y_1 \ldots Y_k)\, \mathbf{f}_2(Y_1 \ldots Y_k, Z, \ldots Z_l).$$

If all the variables are binary, then $\mathbf{f}_1$ and $\mathbf{f}_2$ have 2^{j+k} and 2^{k+l} entries, respectively, and the pointwise product has 2^{j+k+l} entries. For example, given two factors $\mathbf{f}_1(A, B)$ and $\mathbf{f}_2(B, C)$, the pointwise product $\mathbf{f}_1 \times \mathbf{f}_2 = \mathbf{f}_3(A, B, C)$ has $2^{1+1+1} = 8$ entries, as illustrated in Figure 14.10. Notice that the factor resulting from a pointwise product can contain more variables than any of the factors being multiplied and that the size of a factor is exponential in the number of variables. This is where both space and time complexity arise in the variable elimination algorithm.

A	B	$\mathbf{f}_1(A,B)$	B	C	$\mathbf{f}_2(B,C)$	A	B	C	$\mathbf{f}_3(A,B,C)$
T	T	.3	T	T	.2	T	T	T	$.3 \times .2 = .06$
T	F	.7	T	F	.8	T	T	F	$.3 \times .8 = .24$
F	T	.9	F	T	.6	T	F	T	$.7 \times .6 = .42$
F	F	.1	F	F	.4	T	F	F	$.7 \times .4 = .28$
						F	T	T	$.9 \times .2 = .18$
						F	T	F	$.9 \times .8 = .72$
						F	F	T	$.1 \times .6 = .06$
						F	F	F	$.1 \times .4 = .04$

Figure 14.10 Illustrating pointwise multiplication: $\mathbf{f}_1(A,B) \times \mathbf{f}_2(B,C) = \mathbf{f}_3(A,B,C)$.

Summing out a variable from a product of factors is done by adding up the submatrices formed by fixing the variable to each of its values in turn. For example, to sum out A from $\mathbf{f}_3(A,B,C)$, we write

$$\begin{aligned}\mathbf{f}(B,C) &= \sum_a \mathbf{f}_3(A,B,C) = \mathbf{f}_3(a,B,C) + \mathbf{f}_3(\neg a,B,C) \\ &= \begin{pmatrix} .06 & .24 \\ .42 & .28 \end{pmatrix} + \begin{pmatrix} .18 & .72 \\ .06 & .04 \end{pmatrix} = \begin{pmatrix} .24 & .96 \\ .48 & .32 \end{pmatrix} .\end{aligned}$$

The only trick is to notice that any factor that does *not* depend on the variable to be summed out can be moved outside the summation. For example, if we were to sum out E first in the burglary network, the relevant part of the expression would be

$$\sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) \times \mathbf{f}_4(A) \times \mathbf{f}_5(A) = \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) .$$

Now the pointwise product inside the summation is computed, and the variable is summed out of the resulting matrix.

Notice that matrices are *not* multiplied until we need to sum out a variable from the accumulated product. At that point, we multiply just those matrices that include the variable to be summed out. Given functions for pointwise product and summing out, the variable elimination algorithm itself can be written quite simply, as shown in Figure 14.11.

Variable ordering and variable relevance

The algorithm in Figure 14.11 includes an unspecified ORDER function to choose an ordering for the variables. Every choice of ordering yields a valid algorithm, but different orderings cause different intermediate factors to be generated during the calculation. For example, in the calculation shown previously, we eliminated A before E; if we do it the other way, the calculation becomes

$$\mathbf{P}(B \mid j,m) = \alpha\, \mathbf{f}_1(B) \times \sum_a \mathbf{f}_4(A) \times \mathbf{f}_5(A) \times \sum_e \mathbf{f}_2(E) \times \mathbf{f}_3(A,B,E) ,$$

during which a new factor $\mathbf{f}_6(A,B)$ will be generated.

In general, the time and space requirements of variable elimination are dominated by the size of the largest factor constructed during the operation of the algorithm. This in turn

```
function ELIMINATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayesian network specifying joint distribution P(X_1, ..., X_n)

  factors ← [ ]
  for each var in ORDER(bn.VARS) do
      factors ← [MAKE-FACTOR(var, e)|factors]
      if var is a hidden variable then factors ← SUM-OUT(var, factors)
  return NORMALIZE(POINTWISE-PRODUCT(factors))
```

Figure 14.11 The variable elimination algorithm for inference in Bayesian networks.

is determined by the order of elimination of variables and by the structure of the network. It turns out to be intractable to determine the optimal ordering, but several good heuristics are available. One fairly effective method is a greedy one: eliminate whichever variable minimizes the size of the next factor to be constructed.

Let us consider one more query: $\mathbf{P}(JohnCalls \mid Burglary = true)$. As usual, the first step is to write out the nested summation:

$$\mathbf{P}(J \mid b) = \alpha\, P(b) \sum_e P(e) \sum_a P(a \mid b, e) \mathbf{P}(J \mid a) \sum_m P(m \mid a)\ .$$

Evaluating this expression from right to left, we notice something interesting: $\sum_m P(m \mid a)$ is equal to 1 by definition! Hence, there was no need to include it in the first place; the variable M is *irrelevant* to this query. Another way of saying this is that the result of the query $P(JohnCalls \mid Burglary = true)$ is unchanged if we remove $MaryCalls$ from the network altogether. In general, we can remove any leaf node that is not a query variable or an evidence variable. After its removal, there may be some more leaf nodes, and these too may be irrelevant. Continuing this process, we eventually find that *every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.* A variable elimination algorithm can therefore remove all these variables before evaluating the query.

14.4.3 The complexity of exact inference

The complexity of exact inference in Bayesian networks depends strongly on the structure of the network. The burglary network of Figure 14.2 belongs to the family of networks in which there is at most one undirected path between any two nodes in the network. These are called

SINGLY CONNECTED

POLYTREE

singly connected networks or **polytrees**, and they have a particularly nice property: *The time and space complexity of exact inference in polytrees is linear in the size of the network.* Here, the size is defined as the number of CPT entries; if the number of parents of each node is bounded by a constant, then the complexity will also be linear in the number of nodes.

MULTIPLY CONNECTED

For **multiply connected** networks, such as that of Figure 14.12(a), variable elimination can have exponential time and space complexity in the worst case, even when the number of parents per node is bounded. This is not surprising when one considers that *because it*