Chapter 5

Inference by Variable Elimination

Our purpose in this chapter is to present one of the simplest methods for general inference in Bayesian networks, known as the method of Variable Elimination.

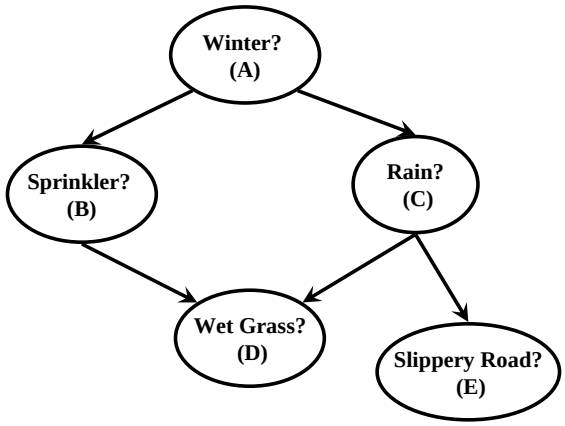
5.1 Introduction

Consider the Bayesian network in Figure 5.1, and let Pr be the probability distribution induced by this network. Our goal in this chapter is to present a general algorithm which can be used to compute posterior marginals for any set of variables in this network and any other Bayesian network. Suppose for example that it is winter and the sprinkler is turned off, $\mathbf{e}: A$=true, B=false, and we are interested in the posterior marginals over variables D and E:

D	E	$\mathsf{Pr}(D, E\|\mathbf{e})$
true	true	0.448
true	false	0.192
false	true	0.112
false	false	0.248

The algorithm we shall present in this chapter will compute such a posterior marginal through a simple procedure, known as *variable elimination.* Intuitively, the algorithm works by *implicitly* constructing the joint probability distribution induced by the Bayesian network, and then summing out variables A, B and C, therefore, constructing a marginal distribution over the variables of interest, D and E.

To explain the process of summing out variables from a probability distribution, consider the joint probability distribution in Figure 5.2 which is induced by the Bayesian network in Figure 5.1. To sum out a variable, say E, from this distribution is to produce a marginal distribution over the remaining variables



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25

A	C	$\Theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.1
false	false	.9

B	C	D	$\Theta_{D\|BC}$
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

C	E	$\Theta_{E\|C}$
true	true	.7
true	false	.3
false	true	0
false	false	1

Figure 5.1: A Bayesian network over five propositional variables.

A, B, C and D. This can be done easily by *merging* all rows that agree on the values of variables A, B, C and D. For example, the first two rows in Figure 5.2:

A	B	C	D	E	Pr(.)
true	true	true	true	true	0.06384
true	true	true	true	false	0.02736

are merged into the row:

A	B	C	D	Pr(.)
true	true	true	true	$0.0912 = 0.06384 + 0.02736$

As we merge rows, we drop reference to the summed–out variable E and add up the probabilities of merged rows. Hence, the result of summing out variable E

A	B	C	D	E	
true	true	true	true	true	0.06384
true	true	true	true	false	0.02736
true	true	true	false	true	0.00336
true	true	true	false	false	0.00144
true	true	false	true	true	0.0
true	true	false	true	false	0.02160
true	true	false	false	true	0.0
true	true	false	false	false	0.00240
true	false	true	true	true	0.21504
true	false	true	true	false	0.09216
true	false	true	false	true	0.05376
true	false	true	false	false	0.02304
true	false	false	true	true	0.0
true	false	false	true	false	0.0
true	false	false	false	true	0.0
true	false	false	false	false	0.09600
false	true	true	true	true	0.01995
false	true	true	true	false	0.00855
false	true	true	false	true	0.00105
false	true	true	false	false	0.00045
false	true	false	true	true	0.0
false	true	false	true	false	0.24300
false	true	false	false	true	0.0
false	true	false	false	false	0.02700
false	false	true	true	true	0.00560
false	false	true	true	false	0.00240
false	false	true	false	true	0.00140
false	false	true	false	false	0.00060
false	false	false	true	true	0.0
false	false	false	true	false	0.0
false	false	false	false	true	0.0
false	false	false	false	false	0.0900

Figure 5.2: A joint probability distribution induced by the Bayesian network in Figure 5.1.

B	C	D	T_1
true	true	true	.95
true	true	false	.05
true	false	true	.9
true	false	false	.1
false	true	true	.8
false	true	false	.2
false	false	true	0
false	false	false	1

D	E	T_2
true	true	0.448
true	false	0.192
false	true	0.112
false	false	0.248

Figure 5.3: Two probability tables (factors).

from the distribution $\Pr$ in Figure 5.2, which has 32 rows, is another distribution $\Pr'$ that does not mention E and that has 16 rows. The important property of summing variables out is that the new distribution is as good as the original one as far as answering queries that do not mention variable E. That is, $\Pr'(\alpha) = \Pr(\alpha)$ for any event α which does not mention variable E. Therefore, if we want to compute the marginal distribution over, say variables B and D, all we have to do then is sum out variables A, C and E from the joint distribution.

This procedure will always work, but its complexity is exponential in the number of variables in the Bayesian network. The key insight underlying the method of variable elimination is that one can sum out variables without having to construct the joint probability distribution explicitly. Instead, variables can be summed out while keeping the original distribution, and all successive distributions, in factored form. This allows the procedure to sometimes escape the exponential complexity of the brute–force method discussed above.

Before we discuss the method of variable elimination though, we need to discuss its central component, known as a probability table.

5.2 Probability Tables

The notion of a *probability table,* also known as a *factor* or *potential,* is a central notion in many of the algorithms we shall discuss, including variable elimination. Figure 5.3 depicts two probability tables, the first of which is over three variables B, C and D. Each row of this table has two components: an instantiation and a number. The instantiation is an assignment of values to variables and the number represents a probability which relates to the instantiation. In some cases, the number represents the probability of the corresponding instantiation, as in Table T_2 of Figure 5.3, which represents a distribution over variables D and E. In other cases, the number represents some conditional probability that relates to the instantiation, as in Table T_1 of Figure 5.3, which represents the conditional probability of D given B and C. Hence, it is important to stress that a probability table does not necessarily represent a probability distribution over the corresponding variables. However, most of the computations we will perform

on tables will start off with tables that represent conditional probabilities and end up with tables that represent marginal probabilities. In the process though, we may have a mixture of tables with different probabilistic interpretations. Following is the formal definition of a probability table.

Definition 1 *A probability table T over variables $\mathbf{X}$ is a function which maps each instantiation $\mathbf{x}$ of variables $\mathbf{X}$ to a non–negative number, denoted $T(\mathbf{x})$.*

We will use $\mathsf{vars}(T)$ to denote the variables over which the table T is defined. We will also write $T(X_1, \ldots, X_n)$ to indicate that $X_1, \ldots, X_n$ are the variables over which table T is defined. Finally, we will allow tables over an empty set of variables. Such tables are called *trivial* as they assign a single number to the empty instantiation.

There are two key operations which are commonly applied to tables. The first is that of *summing out* a variable from a table, and the second is that of *multiplying* two tables. We will next define these operations and discuss their complexity as they represent the building blocks of many algorithms for inference with Bayesian networks, including variable elimination.

We already discussed the summing–out operation informally in the previous section, so here is the formal definition.

Definition 2 *Let T be a probability table over variables $\mathbf{X}$ and let X be a variable in $\mathbf{X}$. The result of summing out variable X from table T is another table over variables $\mathbf{Y} = \mathbf{X} - \{X\}$, which is denoted by $\sum_X T$ and defined as follows:*

$$\left(\sum_X T\right)(\mathbf{y}) \stackrel{def}{=} \sum_x T(\mathbf{y}, x).$$

To visualize this summing–out operation, consider Table T_1 in Figure 5.3. The result of summing out variable D from this table is then:

B	C	$\sum_D T_1$
true	true	1
true	false	1
false	true	1
false	false	1

Note that if we sum out variables B and C from the above table, we get a trivial table which assigns the number 4 to the empty instantiation:

	$\sum_B \sum_C \sum_D T_1$
	4

The summing–out operation is clearly commutative. That is,

$$\sum_Y \sum_X T = \sum_X \sum_Y T.$$

Hence, it makes sense to talk about summing out multiple variables from a table, without fixing the variable order. This also justifies the notation $\sum_{\mathbf{X}} T$, where $\mathbf{X}$ is a set of variables. Summing out variables is also known as the *marginalization* of these variables.

Summing out any number of variables from a probability table T can be done in time which is linear in the size of table T. Appendix A provides pseudocode for implementing this operation. It is important to keep this complexity in mind though, as it is essential for analyzing the complexity of various inference algorithms based on variable elimination.

We now discuss the second operation on tables, which is called multiplication. Consider the two tables, $T_1(B, C, D)$ and $T_2(D, E)$, in Figure 5.3. To multiply these two tables is to construct a table over the union of their variables, B, C, D and E, which is shown partially below:

B	C	D	E	$T_1(B,C,D)T_2(D,E)$
true	true	true	true	$0.4256 = (.95)(.448)$
true	true	true	false	$0.1824 = (.95)(.192)$
true	true	false	true	$0.0056 = (.05)(.112)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
false	false	false	false	$0.2480 = (1)(.248)$

Note that each instantiation b, c, d, e of the resulting table is compatible with exactly one instantiation in Table T_1, b, c, d, and exactly one instantiation in Table T_2, d, e. The number assigned by the new table to instantiation b, c, d, e is then the product of numbers assigned by Tables T_1 and T_2 to these compatible instantiations. Following is the formal definition of table multiplication.

Definition 3 *The result of multiplying tables $T_1(\mathbf{X})$ and $T_2(\mathbf{Y})$ is another table over variables $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$, which is denoted by T_1T_2 and defined as follows:*

$$(T_1T_2)(\mathbf{z}) \stackrel{def}{=} T_1(\mathbf{x})T_2(\mathbf{y}),$$

where $\mathbf{x}$ and $\mathbf{y}$ are consistent with $\mathbf{z}$.

The multiplication of tables is commutative and associative. Hence, it is meaningful to talk about multiplying a number of tables without specifing the order of this multiplication process.

The complexity of table multiplication is linear in the size of resulting table. This is, when multiplying tables $T_1(\mathbf{X}_1), \ldots, T_n(\mathbf{X}_n)$, the complexity is linear in the size of resulting table which must be defined over variables $\mathbf{X} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_n$. But the size of table $T_1 \ldots T_n$ is exponential in the number of variables in $\mathbf{X}$. Hence, it is more common to say that the complexity of table multiplication is $O(c^w)$, where w is the number of variables in $\mathbf{X}$, and c is the largest number of values that any variable in $\mathbf{X}$ can take. Appendix A provides pseudocode for implementing table multiplication. Again, it is important to keep this complexity in mind, as table multiplication is central to many algorithms for inference in Bayesian networks.

5.3 Elimination as a Basis for Inference

Consider again the Bayesian network in Figure 5.1 and suppose that our goal is to compute the joint probability distribution for this network (which is given in Figure 5.2). We can do this in two ways. First, we can use the Chain Rule for Bayesian networks, which allows us to compute the probability of each instantiation a, b, c, d, e as a product of network parameters:

$$\Pr(a, b, c, d, e) = \theta_{e|c}\, \theta_{d|bc}\, \theta_{c|a}\, \theta_{b|a}\, \theta_a.$$

Another more direct method is to multiply the CPTs for this Bayesian network. In particular, it is easy to verify that the table:

$$\Theta_{E|C}\, \Theta_{D|BC}\, \Theta_{C|A}\, \Theta_{B|A}\, \Theta_A.$$

is indeed the joint probability distribution given in Figure 5.2. This shows one of the key applications of table multiplication, as it allows us to express the joint probability distribution of any Bayesian network as a product of its CPTs.

Suppose now that our goal is to compute the marginal distribution over variables D and E in the above Bayesian network. We know that this marginal can be obtained by summing out variables A, B and C from the joint probability distribution. Hence, the marginal we want can be expressed as follows:

$$\Pr(D, E) = \sum_{A,B,C} \Theta_{E|C}\, \Theta_{D|BC}\, \Theta_{C|A}\, \Theta_{B|A}\, \Theta_A.$$

This shows the power of combining the operation for summing out variables with the one for multiplying tables, as they are all we need to compute the marginal for any set of variables. There is still the problem of complexity though, since the multiplication of all CPTs takes time exponential in the number of network variables. Fortunately though, the following result shows that such a multiplication is not necessary in general.

Theorem 1 *If T_1 and T_2 are tables, and if variable X appears only in T_2, then*

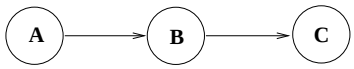
$$\sum_X T_1 T_2 = T_1 \sum_X T_2.$$

Therefore, if $T_1, \ldots, T_n$ are the CPTs of a Bayesian network, and if we want to sum out variable X from the product $T_1 \ldots T_n$, it may not be necessary to multiply these tables first. For example, if variable X appears only in table T_n, then we do not need to multiply any of the tables before we sum out X since

$$\sum_X T_1 \ldots T_n = T_1 \ldots T_{n-1} \sum_X T_n.$$

However, if variable X appears in two tables, say, T_{n-1} and T_n, then we need to multiply these two tables before we can sum out X since

$$\sum_X T_1 \ldots T_n = T_1 \ldots T_{n-2} \sum_X T_{n-1} T_n.$$



A	Θ_A
true	.6
false	.4

A	B	$\Theta_{B\|A}$
true	true	.9
true	false	.1
false	true	.2
false	false	.8

B	C	$\Theta_{C\|B}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

Figure 5.4: A Bayesian network.

In general, to sum out variable X from the product $T_1 \ldots T_n$, all we need to multiply are tables T_k which include X, and then sum out variable X from the resulting table $\prod_k T_k$.

Let us consider an example with respect to the Bayesian network in Figure 5.4. Our goal is to compute the prior marginal on variable C, $\mathsf{Pr}(C)$, by first eliminating variable A and then variable B. There are two tables that contain variable A, Θ_A and $\Theta_{B|A}$. So we must multiply these tables first and then sum out variable A from the resulting table. Multiplying Θ_A and $\Theta_{B|A}$:

A	B	$\Theta_A\Theta_{B\|A}$
true	true	.54
true	false	.06
false	true	.08
false	false	.32

Summing out variable A:

B	$\sum_A \Theta_A\Theta_{B\|A}$
true	.62
false	.38

We we now have two tables, $\sum_A \Theta_A\Theta_{B|A}$ and $\Theta_{C|B}$, and we want to eliminate variable B. Since B appears in both tables, we must multiply them first and then sum out B from the result. Multiplying:

B	C	$\Theta_{C\|B}\sum_A \Theta_A\Theta_{B\|A}$
true	true	.186
true	false	.434
false	true	.190
false	false	.190

Summing out:

C	$\sum_B \Theta_{C\|B}\sum_A \Theta_A\Theta_{B\|A}$
true	.376
false	.624

Algorithm 1 VE-PR-I($\mathcal{N}$: a Bayesian network, $\mathbf{Q}$: some variables in network $\mathcal{N}$, π: an ordering of the n variables not in $\mathbf{Q}$): returns the prior marginal $\mathsf{Pr}(\mathbf{Q})$.

1: $\mathcal{S} \leftarrow$ CPTs of network $\mathcal{N}$
2: **for** $i = 1$ to n **do**
3: $\quad T \leftarrow \prod_k T_k$, where T_k belongs to $\mathcal{S}$ and mentions variable $\pi(i)$
4: $\quad T_i \leftarrow \sum_{\pi(i)} T$
5: $\quad \mathcal{S} \leftarrow \mathcal{S} - \{T_k\} \cup \{T_i\}$
6: Return $\prod_{T \in \mathcal{S}} T$

This table is then the prior marginal for variable C, $\mathsf{Pr}(C)$. Therefore, according to the Bayesian network in Figure 5.4, the probability of C=true is .376 and the probability of C=false is .624.

5.4 Computing Prior Marginals

Algorithm 1 provides pseudocode for computing the marginal for some variables in a Bayesian network, based on the previous elimination method. The algorithm takes as input a Bayesian network $\mathcal{N}$, variables $\mathbf{Q}$, and an elimination order π over network variables other than $\mathbf{Q}$. Here, $\pi(1)$ is the first variable in the order, $\pi(2)$ is the second variable, and so on. The algorithm simply iterates over each variable $\pi(i)$ in the order, identifying all tables T_k that contain variable $\pi(i)$, multiplying them to yield table T, summing out variable $\pi(i)$ from T, and finally replacing tables T_k by table $\sum_{\pi(i)} T$. When all variables in the order π are eliminated, we end up with a set of tables over variables $\mathbf{Q}$. Multiplying these tables is the answer to our query, $\mathsf{Pr}(\mathbf{Q})$. From now on, we will use the phrase *eliminate variable* $\pi(i)$ to denote the multiplication of tables T_k on line 3, followed by summing out variable $\pi(i)$ on line 4.

The question that presents itself now is: How much work does algorithm VE-PR-I actually do? As it turns out, this is an easy question to answer once we observe that the real work done by the algorithm is on line 3, where tables T_k are multiplied together to yield table T, and on line 4, where we sum out variable $\pi(i)$ from table T to yield table T_i. Each of the steps on these lines take time and space which is linear in the size of constructed table T_i. In the example of Figure 5.4, where we eliminated variable A first and then variable B, the biggest table T_i we had to construct had one variable in it. This can be seen in the following formula, which explicates the number of variables appearing in each intermediate table we had to construct:

$$\underbrace{\sum_B \Theta_{C|B} \overbrace{\sum_A \Theta_A \Theta_{B|A}}^{1}}_{1}.$$

Suppose, however, that we eliminate variable B first and then variable A.

Our computation would then be:

$$\underbrace{\sum_A \Theta_A \overbrace{\sum_B \Theta_{B|A}\Theta_{C|B}}^{2}}_{1}$$

which involves constructing an intermediate table with two variables. Therefore, although any order for variable elimination will do, the particular order we use is typically significant computationally. Some orders are better than others in that they lead to constructing smaller intermediate tables on line 4. Therefore, to minimize the resources consumed by VE-PR-I (both time and space) we must choose the "best" order, a subject which we consider in the next section.

5.5 Choosing an Elimination Order

Before we show how to construct elimination orders with good quality, we will first show how to formally measure the quality of a particular elimination order. Specifically, if the largest table ever constructed on line 4 of VE-PR-I (Algorithm 1) has size s, then the complexity of lines 2–5 in VE-PR-I is $O(ns)$. Therefore, we can regard s as a measure of the order quality. It is more customary though to measure the size of a table by the number of variables it contains, instead of the number of rows. Therefore, if the largest table we ever construct has w variables, then the complexity of lines 2–5 is $O(nc^w)$, where c is a bound on the cardinality of variables (that is, the maximum number of values that any variable can assume). In fact, the number w in this case is known the *width* of the order π and is taken to be a measure of its quality. Therefore, we want to choose an order which has the smallest width possible. Note that the total time and space complexity of VE-PR-I is $O(nc^w + c^{|\mathbf{Q}|})$, as we finally construct a table over variables $\mathbf{Q}$ on line 6. The size of this final table and the time to construct it are $O(c^{|\mathbf{Q}|})$. If the number of variables $\mathbf{Q}$ is bounded by a constant, then the complexity of the algorithm drops to $O(nc^w)$. We may also choose to skip the multiplication of tables on line 6 and simply return the tables in $\mathcal{S}$. In this case, we are keeping the marginal for variables $\mathbf{Q}$ in factored form, and the complexity of VE-PR-I is $O(nc^w)$ regardless of variables $\mathbf{Q}$. Finally, note that the above complexity analysis assumes that we can identify tables that mention variable $\pi(i)$ on line 3 in time linear the number of such tables. This can be accomplished using an indexing scheme which we discuss in Section 5.11, leading to a variation on algorithm VE-PR-I known as *bucket elimination.*

Suppose now that we are presented with two orders π_1 and π_2, and we need to choose one of them. We clearly want to choose the one with the smaller width, but how can we compute the width of each order? One straightforward, but inefficient method, is to augment VE-PR-I in order to keep track of the number of variables appearing in table T_i on line 4. To compute the width of a particular order, we simply execute the algorithm on that order and return

the maximum number of variables that any table T_i ever contained. This will clearly work, but we can do much better than this as we shall see next.

Consider the network in Figure 5.1 and suppose that we want to compute the marginal for variable E by eliminating variables according to the order B, C, A, D. The following listing provides a trace of VE-PR-I:

i	$\pi(i)$	$\mathcal{S}$	T_i	w
		$\Theta_A\ \Theta_{B\|A}\ \Theta_{C\|A}\ \Theta_{D\|BC}\ \Theta_{E\|C}$		
1	B	$\Theta_A\ \Theta_{C\|A}\ \Theta_{E\|C}\ T_1(A,C,D)$	$T_1 = \sum_B \Theta_{B\|A}\ \Theta_{D\|BC}$	3
2	C	$\Theta_A\ T_2(A,D,E)$	$T_2 = \sum_C \Theta_{C\|A}\ \Theta_{E\|C}\ T_1(A,C,D)$	3
3	A	$T_3(D,E)$	$T_3 = \sum_A \Theta_A\ T_2(A,D,E)$	2
4	D	$T_4(E)$	$T_4 = \sum_D T_3(D,E)$	1

The second column lists variables according to their elimination order. The third column lists the set of tables $\mathcal{S}$ computed by the algorithm at the end of each iteration i. The fourth column lists the table T_i constructed on line 4 of the algorithm iteration i, and the final column lists the size of this constructed table as measured by the number of its variables. The maximum of these sizes is the width of given order, which is 3 in this case. Note that such an algorithm trace can be constructed without having to execute VE-PR-I. That is, to eliminate variable $\pi(i)$ which appears in tables $T(\mathbf{X}_k)$, we simply replace such tables by a newly constructed table over the variables $\bigcup_k \mathbf{X}_k - \{\pi(i)\}$.

We can also compute the width of an order—and, therefore, compute the size of the biggest table constructed by VE-PR-I under that order—by simply operating on an undirected graph which explicates the interactions between the Bayesian network tables. Such a graph is defined formally below.

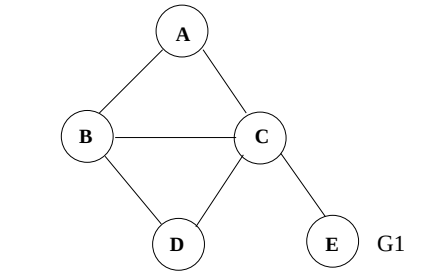
Definition 4 *Let $T_1, \ldots, T_n$ be a set of tables. The interaction graph G of these tables is an undirected graph constructed as follows. The nodes of G are the variables that appear in tables $T_1, \ldots, T_n$. There is an edge between two variables in G iff those variables appear in the same table.*[1]

Figure 5.5 depicts the interaction graph which corresponds to each iteration of the above trace of VE-PR-I. There are two key observations about these interaction graphs:

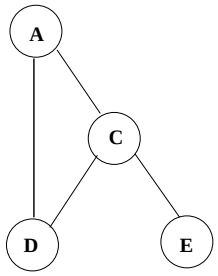
1. If G is the interaction graph of tables $\mathcal{S}$, then eliminating a variable $\pi(i)$ from $\mathcal{S}$ leads to constructing a table over the neighbors of $\pi(i)$ in G. For example, eliminating variable B from the tables $\mathcal{S}_1$ in Figure 5.5 leads to constructing a table over variables A, C, D, which are the neighbors of B in interaction graph G_1.

2. Let $\mathcal{S}'$ be the tables which result from eliminating variable $\pi(i)$ from tables $\mathcal{S}$. If G' and G are the interaction graphs of $\mathcal{S}'$ and $\mathcal{S}$, respectively, then G' can be obtained from G as follows:

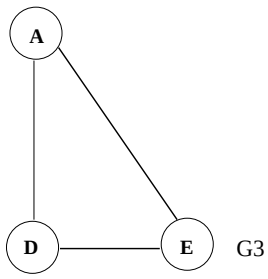
[1] If $T_1, \ldots, T_n$ are the CPTs of a Bayesian network $\mathcal{N}$, the interaction graph is then nothing but the moral graph of the DAG underlying the network $\mathcal{N}$; see Chapter 3.



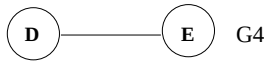
$\mathcal{S}_1 : \ \Theta_A \quad \Theta_{B|A} \quad \Theta_{C|A} \quad \Theta_{D|BC} \quad \Theta_{E|C}$



$\mathcal{S}_2 : \ \Theta_A \quad \Theta_{C|A} \quad \Theta_{E|C} \quad T_1(A, C, D)$



$\mathcal{S}_3 : \ \Theta_A \quad T_2(A, D, E)$



$\mathcal{S}_4 : \ T_3(D, E)$

$\mathcal{S}_5 : \ T_4(E)$

Figure 5.5: Interaction graphs which result from eliminating variables B, C, A, D in that order.

Algorithm 2 OrderWidth($\mathcal{N}$: a Bayesian network, π: an ordering of the n variables in network $\mathcal{N}$): returns the width of elimination order π.

1: $G \leftarrow$ interaction graph of the CPTs in network $\mathcal{N}$
2: $w \leftarrow 0$
3: **for** $i = 1$ to n **do**
4: $\quad w \leftarrow \max(w, d)$, where d is the number of $\pi(i)$'s neighbors in G
5: $\quad$ Connect all neighbors of variable $\pi(i)$ in G
6: $\quad$ Delete variable $\pi(i)$ from G
7: Return w

(a) Add an edge to G between every pair of neighbors of variable $\pi(i)$ that are not already connected by an edge.

(b) Delete variable $\pi(i)$ from G.

In fact, the first step above corresponds to multiplying all tables that contain variable $\pi(i)$ in $\mathcal{S}$, and the second step corresponds to summing out variable $\pi(i)$ from the resulting table.

Algorithm 2 provides pseudocode for computing the width of any order, given that order and a corresponding Bayesian network. It does this by maintaining an interaction graph G for the set of tables $\mathcal{S}$ maintained by VE-PR-I during each iteration. One can use OrderWidth to compute the quality of a particular ordering before using it.

Computing the width of a particular variable order is useful when we have to choose between a small number of orders. When the number of orders is large, however, we need to do better than simply computing the width of each potential order. As it turns out, computing an optimal order is an NP–hard problem. But there are a number of heuristic approaches that tend to generate relatively good orders.

One of the more popular heuristics is also one of the simplest: Always eliminate the variable which leads to constructing the smallest table possible. If we are maintaining the interaction graph as we eliminate variables, this basically means that we always eliminate the variable that has the smallest number of neighbors in the current interaction graph. This heuristic method is given in Algorithm 3, and is known as the *min–degree* heuristic. Using heaps for selecting variables, this algorithm can be implemented in $O(n \lg n)$ time. It is also known that min–degree is optimal when applied to a network that has some elimination order of width ≤ 2.

Another popular heuristic for constructing elimination orders, which is usually more effective than min–degree, is the following. Always eliminate the variable which leads to adding the smallest number of edges on line 4 of MinDegreeOrder. One way to understand the intuition behind this heuristic is to imagine a choice between two variables X and Y, where X has five neighbors but each two of these neighbors are adjacent, i.e., connected by an edge, while variable Y has only two neighbors. Min–degree will choose variable Y in this case since that will construct a smaller table. But since the neighbors of X

Algorithm 3 MinDegreeOrder($\mathcal{N}$: a Bayesian network, $\mathbf{X}$: a set of n variables in $\mathcal{N}$): returns an ordering π of variables $\mathbf{X}$.

1: $G \leftarrow$ interaction graph of the CPTs in network $\mathcal{N}$
2: **for** $i = 1$ to n **do**
3: $\quad \pi(i) \leftarrow$ a variable in $\mathbf{X}$ with smallest number of neighbors in G
4: $\quad$ Add an edge between every pair of non–adjacent neighbors of $\pi(i)$
5: $\quad$ Delete variable $\pi(i)$ from G and from $\mathbf{X}$
6: Return π

Algorithm 4 MinFillOrder($\mathcal{N}$: a Bayesian network, $\mathbf{X}$: a set of n variables in $\mathcal{N}$): returns an ordering π of variables $\mathbf{X}$.

1: $G \leftarrow$ interaction graph of the CPTs in network $\mathcal{N}$
2: **for** $i = 1$ to n **do**
3: $\quad \pi(i) \leftarrow$ a variable in $\mathbf{X}$ that adds the smallest number of edges on line 4
4: $\quad$ Add an edge between every pair of non–adjacent neighbors of $\pi(i)$
5: $\quad$ Delete variable $\pi(i)$ from G and from $\mathbf{X}$
6: Return π

are already pairwise connected, this is an indication that we may already have a table T over X and its neighbors. In this case, the table constructed when eliminating variable X will not be larger than table T. That is, by eliminating variable X, we are not increasing, yet, the width of elimination order being constructed. This heuristic method is given in Algorithm 4, and is known as the *min–fill* heuristic. Again, using heaps, this algorithm can be implemented in $O(n \lg n)$ time.

5.6 Computing Posterior Marginals

We will now present a generalization of VE-PR-I (Algorithm 1) for computing the posterior marginal for any set of variables. For example, if we take $\mathbf{Q} = \{D, E\}$ and $\mathbf{e}$: A=true, B=false in the network of Figure 5.1, then we want to compute the following table:

D	E	$\Pr(\mathbf{Q}\|\mathbf{e})$
true	true	0.448
true	false	0.192
false	true	0.112
false	false	0.248

where the third row above asserts that

$$\Pr(D\text{=false}, E\text{=true} \mid A\text{=true}, B\text{=false}) = .112$$

More generally, given a Bayesian network $\mathcal{N}$, a set of variables $\mathbf{Q}$, and an instantiation $\mathbf{e}$, we want to compute the posterior marginal $\Pr(\mathbf{Q}|\mathbf{e})$ for variables

$\mathbf{Q}$. Recall that prior marginals are a special case of posterior marginals when $\mathbf{e}$ is the empty instantiation.

We will find it more useful though to compute a variation on posterior marginals, called *joint marginals,* $\Pr(\mathbf{Q}, \mathbf{e})$. That is, instead of computing the probability of $\mathbf{q}$ given $\mathbf{e}$, $\Pr(\mathbf{q}|\mathbf{e})$, we compute the probability of $\mathbf{q}$ and $\mathbf{e}$, $\Pr(\mathbf{q}, \mathbf{e})$. If we take $\mathbf{Q} = \{D, E\}$ and $\mathbf{e}$: A=true, B=false in the network of Figure 5.1, the joint marginal is:

D	E	$\Pr(\mathbf{Q}, \mathbf{e})$
true	true	0.21504
true	false	0.09216
false	true	0.05376
false	false	0.11904

For example, the third row above says that

$$\Pr(D\text{=false}, E\text{=true}, A\text{=true}, B\text{=false}) = .05376$$

If we add up the probabilities in the above table, we get .48 which is nothing but the probability of evidence $\mathbf{e}$: A=true, B=false. This is always the case since $\sum_{\mathbf{q}} \Pr(\mathbf{q}, \mathbf{e}) = \mathbf{e}$ by case analysis. Hence, by adding up the probabilities that appear in the joint marginal, we will always get the probability of evidence $\mathbf{e}$. This also means that we can compute the posterior marginal $\Pr(\mathbf{Q}|\mathbf{e})$ by simply normalizing the corresponding joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$. Moreover, the probability of evidence $\mathbf{e}$ is obtained for free.

The method of variable elimination can be easily extended to compute joint marginals if start by *zeroing out* those rows in the joint probability distribution which are inconsistent with evidence $\mathbf{e}$. Specifically, given table $T(\mathbf{X})$ and evidence $\mathbf{e}$, we will define a new table operation which returns a modified table, denoted by $T^{\mathbf{e}}$, by replacing the number $T(\mathbf{x})$ by a zero for every instantiation $\mathbf{x}$ that is inconsistent with evidence $\mathbf{e}$. For example, given the table

D	E	T
true	true	0.448
true	false	0.192
false	true	0.112
false	false	0.248

and evidence $\mathbf{e}$: E=true, we have:

D	E	$T^{\mathbf{e}}$
true	true	0.448
true	false	0
false	true	0.112
false	false	0

We will often omit the zeroed out rows, hence writing:

D	E	$T^{\mathbf{e}}$
true	true	0.448
false	true	0.112

Consider now the network of Figure 5.1 and let $\mathbf{e}$: A=true, B=false. The joint marginal $\mathsf{Pr}(\{D, E\}, \mathbf{e})$ can be computed as follows:

$$\mathsf{Pr}(\{D, E\}, \mathbf{e}) = \sum_{A,B,C} \left(\Theta_{E|C}\, \Theta_{D|BC}\, \Theta_{C|A}\, \Theta_{B|A}\, \Theta_A\right)^{\mathbf{e}}. \tag{5.1}$$

Although this provides a systematic method for computing joint marginals, we still have the problem of complexity as the above equation requires that we multiply all CPTs before we start eliminating variables. Luckily, this is not needed as shown by the following result.

Theorem 2 *If T_1 and T_2 are two tables, and $\mathbf{e}$ is an instantiation, then*

$$(T_1 T_2)^{\mathbf{e}} = {T_1}^{\mathbf{e}} {T_2}^{\mathbf{e}}.$$

Hence, Equation 5.1 reduces to:

$$\mathsf{Pr}(\{D, E\}, \mathbf{e}) = \sum_{A,B,C} {\Theta_{E|C}}^{\mathbf{e}}\, {\Theta_{D|BC}}^{\mathbf{e}}\, {\Theta_{C|A}}^{\mathbf{e}}\, {\Theta_{B|A}}^{\mathbf{e}}\, {\Theta_A}^{\mathbf{e}}$$

which keeps the joint probability distribution in factored form, therefore, allowing us to use VE-PR-I on the reduced CPTs.

Consider now the Bayesian network in Figure 5.4. Let $\mathbf{Q} = \{C\}$, $\mathbf{e}$: A=true, and suppose that we want to compute the joint marginal $\mathsf{Pr}(\mathbf{Q}, \mathbf{e})$ by eliminating variable A first and then variable B. We first need to reduce the network CPTs given evidence $\mathbf{e}$, which gives:

A	${\Theta_A}^{\mathbf{e}}$
true	.6

A	B	${\Theta_{B\|A}}^{\mathbf{e}}$
true	true	.9
true	false	.1

B	C	${\Theta_{C\|B}}^{\mathbf{e}}$
true	true	.3
true	false	.7
false	true	.5
false	false	.5

The formula we need to evaluate is then:

$$\begin{aligned} \mathsf{Pr}(\mathbf{Q}, \mathbf{e}) &= \sum_B \sum_A {\Theta_A}^{\mathbf{e}}\ {\Theta_{B|A}}^{\mathbf{e}}\ {\Theta_{C|B}}^{\mathbf{e}} \\ &= \sum_B {\Theta_{C|B}}^{\mathbf{e}}\ \sum_A {\Theta_A}^{\mathbf{e}}\ {\Theta_{B|A}}^{\mathbf{e}}. \end{aligned}$$

All intermediate tables needed to evaluate the above formula are shown below:

A	B	${\Theta_A}^{\mathbf{e}}{\Theta_{B\|A}}^{\mathbf{e}}$
true	true	.54
true	false	.06

B	$\sum_A {\Theta_A}^{\mathbf{e}}{\Theta_{B\|A}}^{\mathbf{e}}$
true	.54
false	.06

Algorithm 5 VE-PR-II($\mathcal{N}$: a Bayesian network, $\mathbf{Q}$: some variables in network $\mathcal{N}$, $\mathbf{e}$: instantiation of some variables in network $\mathcal{N}$, π: an ordering of the n variables not in $\mathbf{Q}$): returns the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$.

1: $\mathcal{S} \leftarrow \{T^{\mathbf{e}} : T \text{ is a CPT of network } \mathcal{N}\}$
2: **for** $i = 1$ to n **do**
3: $\quad T \leftarrow \prod_k T_k$, where T_k belongs to $\mathcal{S}$ and mentions variable $\pi(i)$
4: $\quad T_i \leftarrow \sum_{\pi(i)} T$
5: $\quad \mathcal{S} \leftarrow \mathcal{S} - \{T_k\} \cup \{T_i\}$
6: Return $\prod_{T \in \mathcal{S}} T$

B	C	$\Theta_{C\|B}{}^{\mathbf{e}} \sum_A \Theta_A{}^{\mathbf{e}} \Theta_{B\|A}{}^{\mathbf{e}}$
true	true	.162
true	false	.378
false	true	.030
false	false	.030

C	$\sum_B \Theta_{C\|B}{}^{\mathbf{e}} \sum_A \Theta_A{}^{\mathbf{e}} \Theta_{B\|A}{}^{\mathbf{e}}$
true	.192
false	.408

Therefore,

$$\begin{aligned} \Pr(C\text{=true}, A\text{=true}) &= .192 \\ \Pr(C\text{=false}, A\text{=true}) &= .408 \\ \Pr(A\text{=true}) &= .600 \end{aligned}$$

To compute the posterior marginal $\Pr(\{C\}|A\text{=true})$, all we have to do is normalize the above table, which gives:

C	$\Pr(\{C\}\|A\text{=true})$
true	.32
false	.68

Therefore, $\Pr(C\text{=true}|A\text{=true}) = .32$ and $\Pr(C\text{=false}|A\text{=true}) = .68$.

Algorithm 5 provides pseudocode for computing the joint marginal for a set of variables $\mathbf{Q}$ with respect to a Bayesian network $\mathcal{N}$ and evidence $\mathbf{e}$. The algorithm is a simple modification of VE-PR-I (Algorithm 1) where we reduce the tables of a Bayesian network before we start eliminating variables. By normalizing the output of this algorithm, we immediately obtain an algorithm for computing posterior marginals, $\Pr(\mathbf{Q}|\mathbf{e})$. Moreover, by adding up the numbers returned by this algorithm, we immediately obtain an algorithm for computing the probability of evidence, $\Pr(\mathbf{e})$. It is not uncommon to run VE-PR-II with $\mathbf{Q}$ being the emptyset. In this case, the algorithm will eliminate all variables in the Bayesian network, therefore returning a trivial table, a number, which represents the probability of evidence $\mathbf{e}$.

5.7 Network Structure and Complexity

Suppose we have two Bayesian networks, each containing, say, a hundred variables. The best elimination order for the first network could have width 3, which

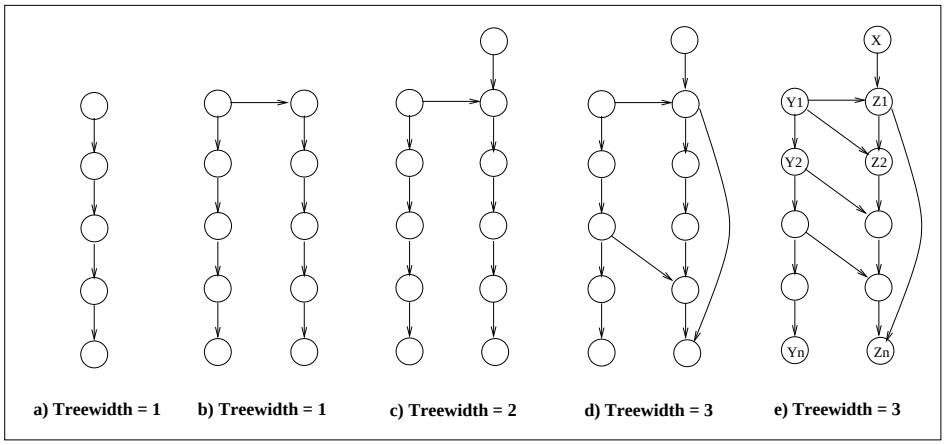


Figure 5.6: Networks with different treewidth.

means that VE-PR-II will do very well on that network. The best elimination order for the second network could have width 20, which means that VE-PR-II will do relatively poorly on this network. Why would VE-PR-II find one of these networks more difficult than the other, even though they have the same number of variables?

The answer to this question lies in the notion of network connectivity. Specifically, the width of the best elimination order for a given network is determined by its connectivity as measured by a formal notion from graph theory known as *treewidth.* Figure 5.6 depicts some networks with their treewidths. In general, the more connected the network is, the higher its treewidth is. In graph theory, treewidth is defined for undirected graphs but we extend the definition to Bayesian networks as follows.

Definition 5 *The treewidth of a Bayesian network is the treewidth of the undirected graph representing its moral graph.*

Recall that the moral graph of a Bayesian network $\mathcal{N}$ is also the interaction graph of its CPTs. Figure 5.7 depicts the moral graphs of the networks in Figure 5.6. The classical definition of treewidth for an undirected graph will be given formally in a later chapter, where it will be discussed at length. For now though, it is sufficient to say that the treewidth of a Bayesian network corresponds to the width of its best elimination order.

When we say an "elimination order" for a Bayesian network, we mean an order which includes all its variables. This assumes that we are eliminating all variables of the network, which corresponds to a query where the variables $\mathbf{Q}$ are empty in VE-PR-II. For this class of queries, which basically computes the probability of instantiation $\mathbf{e}$, the time and space complexity of VE-PR-II is $O(nc^w)$, where n is the number of network variables, c is the maximum number of values per variable, and w is the network treewidth. This complexity result

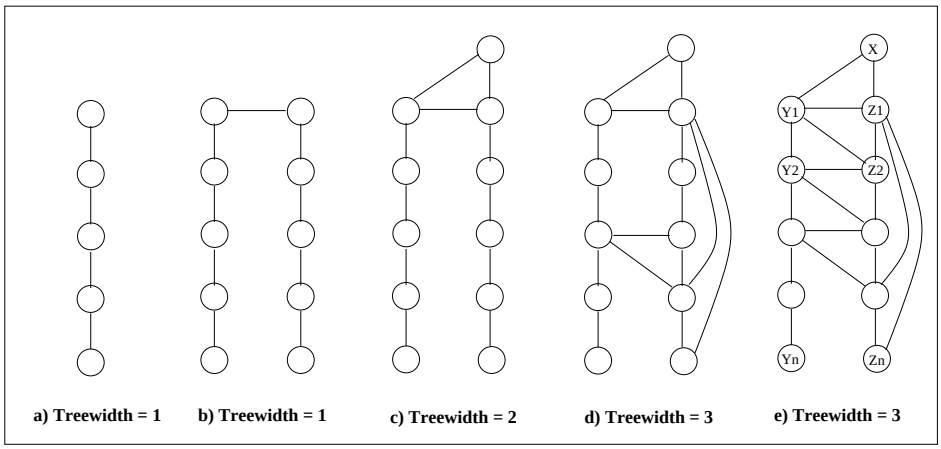


Figure 5.7: Undirected graphs with different treewidth. These graphs are also the moral graphs of the networks that appear in Figure 5.6.

assumes that we always pass the best elimination order to VE-PR-II. Therefore, for this class of queries, the performance of this algorithm depends precisely on the treewidth of the network to which it is applied. If the network has a bounded treewidth, then the algorithm will take linear time and space to execute. A number of observations are in order about treewidth.

The number of nodes has no genuine effect on treewidth. For example, the network in Figure 5.6(a) has treewidth of 1. The network in Figure 5.6(b) is obtained by more than doubling the number of nodes. Yet, treewidth remains to be 1. Therefore, the performance of VE-PR-II will only double when moving from the first to the second network.

The number of parents per node has a direct effect on treewidth. If the number of parents per node is k, treewidth is no less than k. If we have a node with k parents, we must have a table which has $k + 1$ variables in the Bayesian network. Since we must process this table in the course of VE-PR-II, no elimination order will have a width which is less than k. The network in Figure 5.6(c) was obtained from the one in Figure 5.6(b) by adding one node and one edge. But this addition has increased the maximum number of parents per node from 1 to 2, leading also to an increase in the treewidth.

Cycles have a genuine effect on treewidth. In general, more cycles tend to lead to higher treewidth. The network in Figure 5.6(d) was obtained from the one in Figure 5.6(c) by creating more cycles. Note that the number of parents per node did not increase, yet treewidth has increased from 2 to 3, which is due to increasing the connectivity of the network through more cycles.

The number of cycles perse does not have a genuine effect on treewidth. It is the nature of these cycles which does; their interaction in particular. The network in Figure 5.6(e) has more cycles than the one in Figure 5.6(d), yet it has the same treewidth.

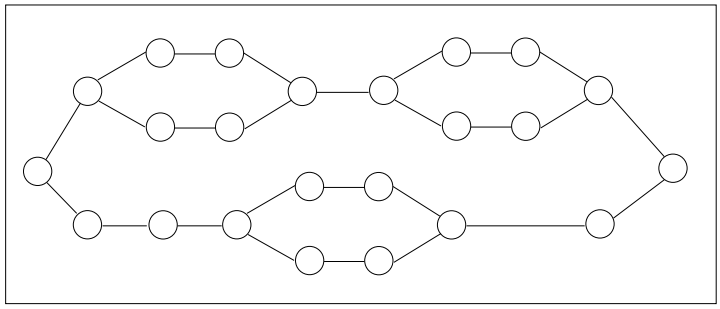


Figure 5.8: A series–parallel graph.

There is a wide class of networks/graphs whose treewidth is bounded by a constant:

Tree networks are ones where each node has at most one parent. The treewidth of such networks is at most 1, no matter what their size is. The networks in Figure 5.6(a)&(b) are trees.

Polytree networks, also known as *singly–connected* networks, are ones where there is at most one (undirected) path between any two nodes. The treewidth of such networks is k, where k is the maximum number of parents that any node may have. The network in Figure 5.6(c) is singly–connected.

Series–parallel graphs have a treewidth which is bounded by 2, independent of their size. A series–parallel graph is shown in Figure 5.8. Each series–parallel graph has a node which acts as *input* and another node which acts as *output.* This class of graphs is defined inductively as follows. A single node N is a series–parallel graph, where N acts as both the graph input and output. If $\mathcal{N}_1$ and $\mathcal{N}_2$ are two series–parallel graphs, with their inputs being I_1, I_2, respectively, and their outputs being O_1, O_2, respectively, then:

1. The two graphs can be connected in *series* as given in Figure 5.9 to yield a new series–parallel graph with input I_1 and output O_2.
2. The two graphs can be connected in *parallel* as given in Figure 5.9 to yield a new series–parallel graph with input I and output O.

Series–parallel graphs can contain many cycles, yet they have a bounded treewidth.

We mentioned earlier that finding an optimal elimination order is NP–hard. The good news though is that finding such an optimal order becomes linear in the network size once the treewidth itself is bounded by a constant. A special case is when the treewidth is no more than 2, where the min–degree heuristic is known to be optimal.

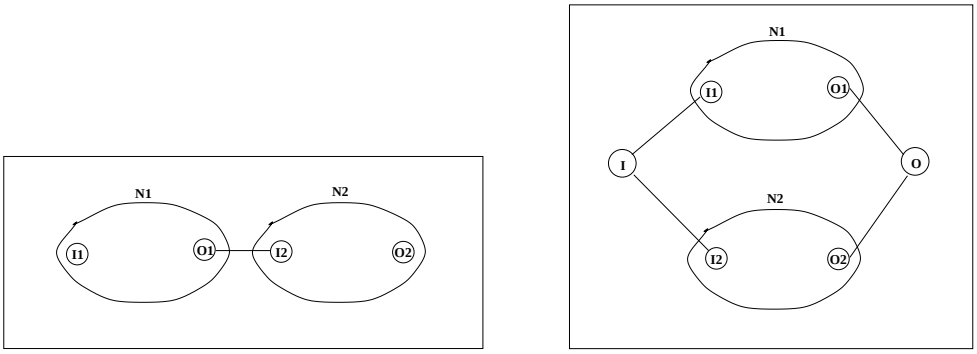


Figure 5.9: On the left, a series connection of two series–parallel graphs. On the right, a parallel connection of two series–parallel graphs.

5.8 Query Structure and Complexity

Network structure has a major impact on the performance of VE-PR-II (Algorithm 5) and on the performance of almost all algorithms we shall discuss later. This is the reason why such algorithms are sometimes known as *structure–based algorithms.* The performance of these algorithms, however, is affected by another important factor: the query structure. In general, a query has two parts:

1. an instantiation $\mathbf{e}$ which is typically known as *evidence,* and
2. a set of variables $\mathbf{Q}$, which is known as the set of *query variables,*

where the goal is to compute the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$. It is possible that we have no evidence, $\mathbf{E} = \emptyset$, in which case we are interested in the prior marginal for variables $\mathbf{Q}$. It is also possible that we have no query variables, $\mathbf{Q} = \emptyset$, in which case we are interested in computing the probability $\Pr(\mathbf{e})$ of evidence $\mathbf{e}$. As shall discuss next, the complexity of inference can be very much affected by the number and location of query and evidence variables within the network structure.

The effect of query structure on the complexity of inference in Bayesian networks is actually independent of any particular algorithm and can be stated precisely without having to assume any such algorithm. For a given query $(\mathbf{Q}, \mathbf{e})$, we will now provide two transformations on a Bayesian network which simplify the network, yet preserve its ability to compute the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$ correctly.

5.8.1 Pruning Nodes

Given a Bayesian network $\mathcal{N}$ and query $(\mathbf{Q}, \mathbf{e})$, one can remove any leaf node (with its CPT) from the network as long as it does not belong to variables $\mathbf{Q} \cup \mathbf{E}$, yet not affect the ability of the network to answer the query correctly. What makes this pruning operation powerful is that it can be applied recursively,

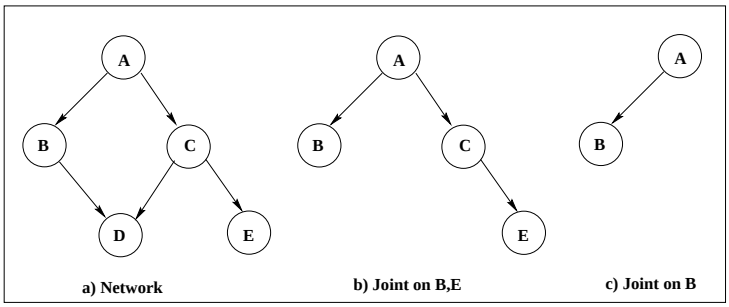


Figure 5.10: Pruning a Bayesian network given two different queries.

possibly leading to the pruning of many network nodes. The result of removing leaf nodes as suggested above will be denoted by $\mathsf{pruneNodes}(\mathcal{N}, \mathbf{Q} \cup \mathbf{E})$.

Theorem 3 *Let $\mathcal{N}$ be a Bayesian network and let $(\mathbf{Q}, \mathbf{e})$ be a corresponding query. If $\mathcal{N}' = \mathsf{pruneNodes}(\mathcal{N}, \mathbf{Q} \cup \mathbf{E})$, then $\mathsf{Pr}(\mathbf{Q}, \mathbf{e}) = \mathsf{Pr}'(\mathbf{Q}, \mathbf{e})$, where Pr and Pr' are the probability distributions induced by networks $\mathcal{N}$ and $\mathcal{N}'$, respectively.*

This follows easily from the fact that for any CPT $\Theta_{X|\mathbf{U}}$, the result of summing out variable X from $\Theta_{X|\mathbf{U}}$ is a table which assigns 1 to each of its instantiations:

$$\left(\sum_X \Theta_{X|\mathbf{U}} \right) (\mathbf{u}) = 1.$$

Multiplying the above table by any other table T that includes variables $\mathbf{U}$ will give table T back. Therefore, if a leaf variable X does not belong to $\mathbf{Q}$ nor to $\mathbf{E}$, we can sum it out first, leading to a useless table. This is why we can always prune such a leaf variable X from the network.

Figure 5.10 depicts a Bayesian network and two of its prunings. In the first case, we are interested in the marginal for variables B and E. Therefore, D is a leaf node that can be pruned. After this pruning, all leaf nodes appear in the query and, therefore, cannot be pruned. In the second case, we are interested in the joint marginal over variable B. Therefore, D and E are leaf nodes that can be pruned. After pruning them, however, node C becomes a leaf node which can also be pruned since it does not appear in the query. Note that the network in Figure 5.10(a) has treewidth 2, yet the pruned networks in Figures 5.10(b)&(c) have treewidth 1.

Network pruning can lead to a significant reduction in a Bayesian network if the query variables appear close to the network roots. In the worst case, all leaf nodes appear in the query and, therefore, no pruning is possible. In the best case, the query contains only root nodes which permits the pruning of every node except for those appearing in the query.

5.8.2 Pruning Edges

Given a Bayesian network $\mathcal{N}$ and a query $(\mathbf{Q}, \mathbf{e})$, one can also eliminate some of the network edges and reduce some of its tables without affecting its ability to compute the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$ correctly. In particular, for each edge $U \longrightarrow X$ which originates from a node U in $\mathbf{E}$, we can

1. Remove the edge $U \longrightarrow X$ from the network.
2. Replace the CPT $\Theta_{X|\mathbf{U}}$ for node X by a smaller CPT, which is obtained from $\Theta_{X|\mathbf{U}}$ by assuming the value of parent U given in evidence $\mathbf{E}$. This new CPT corresponds to $\sum_U \Theta_{X|\mathbf{U}}{}^{\mathbf{e}}$.

The result of the above operation will be denoted by $\mathsf{pruneEdges}(\mathcal{N}, \mathbf{e})$ and we have the following result.

Theorem 4 *Let $\mathcal{N}$ be a Bayesian network and let $\mathbf{e}$ be an instantiation. If $\mathcal{N}' = \mathsf{pruneEdges}(\mathcal{N}, \mathbf{e})$, then $\Pr(\mathbf{Q}, \mathbf{e}) = \Pr'(\mathbf{Q}, \mathbf{e})$, where $\Pr$ and $\Pr'$ are the probability distributions induced by networks $\mathcal{N}$ and $\mathcal{N}'$, respectively.*

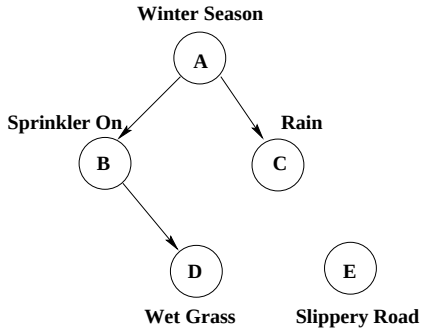
Figure 5.11 depicts the result of pruning edges in the network of Figure 5.1 given evidence C=false. The two edges originating from node C were deleted and tables $\Theta_{D|BC}$ and $\Theta_{E|C}$ were modified. In particular, all rows inconsistent with C=false were removed and the reference to C was dropped from these tables. It is important to stress that the pruned network is only good for answering queries of the form $\Pr(\mathbf{q}, C$=false$)$. If the instantiation C=false does not appear in the query, then the answers returned by the pruned and original networks may disagree.

5.8.3 Network Pruning

The result of pruning nodes and edges for a network will be called *pruning* the network towards query $(\mathbf{Q}, \mathbf{e})$, and denoted by $\mathsf{pruneNetwork}(\mathcal{N}, \mathbf{Q}, \mathbf{e})$. Figure 5.12 depicts a pruning of the Bayesian network in Figure 5.1 given the query $\mathbf{Q} = \{D\}$ and $\mathbf{e} : A$=true$, C$=false. Pruning leads to removing node E and the edges originating from nodes A and C, and modifying the tables $\Theta_{B|A}$, $\Theta_{C|A}$ and $\Theta_{D|BC}$. Note that the pruned network in Figure 5.12 has a treewidth of 1, where the original network had a treewidth of 2. In general, network pruning may lead to a significant reduction in the treewidth, which suggests the following definition.

Definition 6 *The effective treewidth for a Bayesian network with respect to query $(\mathbf{Q}, \mathbf{e})$ is the treewidth of its pruning given query $(\mathbf{Q}, \mathbf{e})$.*

It is important to realize that pruning a Bayesian network can be accomplished in time which is linear in the size of the network (the size of its tables). Therefore, it is usually worthwhile pruning a Bayesian network before answering queries.



A	Θ_A
true	.6
false	.4

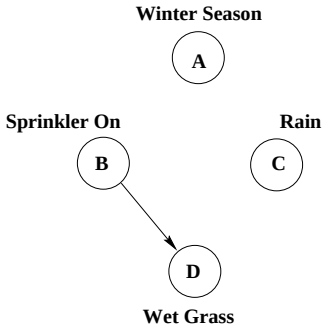
A	B	$\Theta_{B\|A}$
true	true	.2
true	false	.8
false	true	.75
false	false	.25

A	C	$\Theta_{C\|A}$
true	true	.8
true	false	.2
false	true	.1
false	false	.9

B	D	$\sum_C \Theta_{D\|BC}{}^{\mathbf{e}}$
true	true	.9
true	false	.1
false	true	0
false	false	1

E	$\sum_C \Theta_{E\|C}{}^{\mathbf{e}}$
true	0
false	1

Figure 5.11: Pruning edges in the network of Figure 5.1, where $\mathbf{e} : C$=false.



A	Θ_A
true	.6
false	.4

B	$\Theta'_B = \sum_A \Theta_{B\|A}{}^{\mathbf{e}}$
true	.2
false	.8

C	$\Theta'_C = \sum_A \Theta_{C\|A}{}^{\mathbf{e}}$
true	.8
false	.2

B	D	$\Theta'_{D\|B} = \sum_C \Theta_{D\|BC}{}^{\mathbf{e}}$
true	true	.9
true	false	.1
false	true	0
false	false	1

Figure 5.12: Pruning the Bayesian network in Figure 5.1 given the query $\mathbf{Q} = \{D\}$ and $\mathbf{e} : A$=true, C=false.

Algorithm 6 VE-PR($\mathcal{N}$: a Bayesian network, $\mathbf{Q}$: some variables in network $\mathcal{N}$, $\mathbf{e}$: instantiation of some variables in network $\mathcal{N}$): returns the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$.

1: $\mathcal{N}' \leftarrow$ pruneNetwork$(\mathcal{N}, \mathbf{Q}, \mathbf{e})$
2: $\pi \leftarrow$ MinDegreeOrder$(\mathcal{N}', \mathbf{R})$, where $\mathbf{R}$ are network variables not in $\mathbf{Q}$
3: $\mathcal{S} \leftarrow \{T^{\mathbf{e}} : T$ is a CPT of network $\mathcal{N}'\}$
4: **for** $i = 1$ to $|\mathbf{R}|$ **do**
5: $\quad T \leftarrow \prod_k T_k$, where T_k belongs to $\mathcal{S}$ and mentions variable $\pi(i)$
6: $\quad T_i \leftarrow \sum_{\pi(i)} T$
7: $\quad \mathcal{S} \leftarrow \mathcal{S} - \{T_k\} \cup \{T_i\}$
8: Return $\prod_{T \in \mathcal{S}} T$

Algorithm 6 depicts the most general variable elimination algorithm of this chapter, VE-PR. Given a query $(\mathbf{Q}, \mathbf{e})$, the algorithm prunes the Bayesian network before it starts eliminating variables. The elimination order used by the algorithm is computed based on the pruned network, not the original one. This is very important! VE-PR uses MinDegreeOrder for computing a variable ordering, but other algorithms can be used for this purpose as well.

5.8.4 Computing Marginals after Pruning: An Example

We will now apply VE-PR to the network in Figure 5.1 with $\mathbf{Q} = \{D\}$ and $\mathbf{e} : A$=true, C=false. The pruning of this network given the query is depicted in Figure 5.12. The eliminating order $\pi = A, C, B$ is consistent with the min–degree heuristic. Reducing the network CPTs given evidence $\mathbf{e}$ leads to the following tables:

A	$\Theta_A{}^{\mathbf{e}}$
true	.6

B	$\Theta'_B{}^{\mathbf{e}}$
true	.2
false	.8

C	$\Theta'_C{}^{\mathbf{e}}$
false	.2

B	D	$\Theta'_{D\|B}{}^{\mathbf{e}}$
true	true	.9
true	false	.1
false	true	0
false	false	1

VE-PR will then evaluate the following formula:

$$\begin{aligned}\Pr(D, A{=}\text{true}, C{=}\text{false}) &= \sum_B \sum_C \sum_A \Theta_A{}^{\mathbf{e}} \;\; \Theta'_B{}^{\mathbf{e}} \;\; \Theta'_C{}^{\mathbf{e}} \;\; \Theta'_{D|B}{}^{\mathbf{e}} \\ &= \left(\sum_A \Theta_A{}^{\mathbf{e}}\right) \left(\sum_B \Theta'_B{}^{\mathbf{e}} \;\; \Theta'_{D|B}{}^{\mathbf{e}}\right) \left(\sum_C \Theta'_C{}^{\mathbf{e}}\right).\end{aligned}$$

All intermediate tables constructed during this process are shown below:

B	D	$\Theta'_B{}^{\mathbf{e}}\Theta'_{D\|B}{}^{\mathbf{e}}$
true	true	.18
true	false	.02
false	true	0
false	false	.80

$\sum_A \Theta_A{}^{\mathbf{e}}$
.6

D	$\sum_B \Theta'_B{}^{\mathbf{e}} \Theta'_{D\|B}{}^{\mathbf{e}}$
true	.18
false	.82

$\sum_C \Theta'_C{}^{\mathbf{e}}$
.2

The final table returned by VE-PR is then:

D	$\left(\sum_A \Theta_A{}^{\mathbf{e}}\right)\left(\sum_B \Theta'_B{}^{\mathbf{e}} \Theta'_{D\|B}{}^{\mathbf{e}}\right)\left(\sum_C \Theta'_C{}^{\mathbf{e}}\right)$
true	$.0216 = (.6)(.18)(.2)$
false	$.0984 = (.6)(.82)(.2)$

which the joint marginal $\Pr(D, A{=}\text{true}, C{=}\text{false})$. From this table, we conclude that the probability of evidence, $\Pr(A{=}\text{true}, C{=}\text{false})$, is .12. Note how trivial tables act as scaling constants when multiplied by other tables.

5.9 Computing MPEs

We will now present a variation on the variable elimination algorithm for computing the most probable explanation (MPE) given some evidence $\mathbf{e}$. That is, if $\mathbf{Q}$ are all the variables in a Bayesian network, then our goal is to find an instantiation $\mathbf{q}$ of $\mathbf{Q}$ such that $\Pr(\mathbf{q}|\mathbf{e})$ is maximal, which is the same as maximizing $\Pr(\mathbf{q}, \mathbf{e})$ since $\Pr(\mathbf{q}|\mathbf{e}) = \Pr(\mathbf{q}, \mathbf{e})/\Pr(\mathbf{e})$. The method of variable elimination can be used directly for this purpose as we shall illustrate next.

Consider the joint probability table in Figure 5.2, which has one MPE:

A	B	C	D	E	
false	true	false	true	false	0.24300

Suppose now that our goal is not to compute the MPE,

$$A{=}\text{false},\ B{=}\text{true},\ C{=}\text{false},\ D{=}\text{true},\ E{=}\text{false},$$

but its probability of 0.24300. We can do this using the method of variable elimination, but when eliminating a variable, we need to *maximize* over that variable instead of summing it out. To maximize over a variable, say E, is to produce another table over remaining variables A, B, C and D, by merging all rows that agree on the values of these remaining variables. For example, the first two rows in Figure 5.2:

A	B	C	D	E	
true	true	true	true	true	0.06384
true	true	true	true	false	0.02736

are merged into the row:

A	B	C	D	
true	true	true	true	$0.06384 = \max(0.06384, 0.02736)$

As we merge rows, we drop reference to the maximized variable E, and choose the *maximum* probability assigned to the merged rows.

The result of maximizing over variable E in table T of Figure 5.2 is another table which we shall denote by $\max_E T$. Note that table $\max_E T$ does not mention variable E, and has 16 rows (table T has 32 rows). One property of the new table $\max_E T$ is that it agrees with the old table T on the probability of most likely instantiation. Hence, $\max_E T$ is as good as T for computing this probability. This means that we can continue to maximize over variables in $\max_E T$, until we are left with a trivial table. The probability assigned by that table is then the probability of MPE. We will show later how this method can be easily extended so that it returns an MPE, in addition to computing its probability. But first, we need to discuss the properties of the new operation of maximization, which allows us to use an algorithm such VE-PR for computing an MPE and its probability.

First, the formal definition of maximization.

Definition 7 *The maximization of table $T(\mathbf{X})$ over variable X in $\mathbf{X}$ is a new table over variables $\mathbf{Y} = \mathbf{X} - \{X\}$, denoted by $\max_X T$ and defined as follows:*

$$\left(\max_X T\right)(\mathbf{y}) \stackrel{def}{=} \max_x T(x, \mathbf{y}).$$

Like summing out, maximization is commutative, which allows us to talk about maximizing over a set of variables without having to specify the order in which we maximize. Moreover, maximization behaves well when combined with table multiplication:

Theorem 5 *If T_1 and T_2 are tables, and if variable X appears only in T_2, then*

$$\max_X T_1 T_2 = T_1 \max_X T_2.$$

The above results justify the use of Algorithm 7 for computing the probability of an MPE. VE-MPE is exactly like VE-PR, except that (1) it prunes network edges only, since every node in the network is relevant to the result; (2) it eliminates all variables from the network, leading to a trivial table; and (3) it maximizes over variables instead of summing them out. VE-MPE has the same complexity as VE-PR. That is, if we use an elimination order with width w, and if variable cardinalities are bounded by c, the time and space complexity of VE-MPE are then $O(nc^w)$.

5.9.1 Recovering an MPE

VE-MPE can be easily modified to compute a particular MPE, in addition to its probability. The basic idea is that each table assigns both a *number* and an *instantiation* to each row. Consider the following table for an example:

Y	O	T	
true	false	.000095	I=true, X=true
false	false	.460845	I=false, X=false

Algorithm 7 VE-MPE($\mathcal{N}$: a Bayesian network, $\mathbf{e}$: instantiation of some variables in network $\mathcal{N}$): returns $\max_{\mathbf{q}} \Pr(\mathbf{q}, \mathbf{e})$ where $\mathbf{Q}$ are all variables in network $\mathcal{N}$.

1: $\mathcal{N}' \leftarrow$ pruneEdges($\mathcal{N}, \mathbf{e}$)
2: $\mathbf{Q} \leftarrow$ variables in network $\mathcal{N}'$
3: $\pi \leftarrow$ MinDegreeOrder($\mathcal{N}', \mathbf{Q}$)
4: $\mathcal{S} \leftarrow \{T^{\mathbf{e}} : T$ is a CPT of network $\mathcal{N}'\}$
5: **for** $i = 1$ to $|\mathbf{Q}|$ **do**
6: $\quad T \leftarrow \prod_k T_k$, where T_k belongs to $\mathcal{S}$ and mentions variable $\pi(i)$
7: $\quad T_i \leftarrow \max_{\pi(i)} T$
8: $\quad \mathcal{S} \leftarrow \mathcal{S} - \{T_k\} \cup \{T_i\}$
9: Return $T(\emptyset)$, where T is the product of tables $\mathcal{S}$

This table assigns the number .000095 to row Y=true, O=false. But it also assigns the instantiation I=true, X=true to that row. We will use $T[\mathbf{x}]$ to denote the *instantiation* assigned by table T to row $\mathbf{x}$, as opposed to $T(\mathbf{x})$ which is the *number* assigned by T to row $\mathbf{x}$.

The instantiation $T[\mathbf{x}]$ associated with a row $\mathbf{x}$ will now be used to record the MPE as it is being constructed. Consider the following table for an example:

X	Y	O	T	
true	true	false	.0095	I=true
true	false	false	.0205	I=true
false	true	false	.0055	I=false
false	false	false	.4655	I=false

Maximizing over variable X will now give us the following table:

Y	O	$\max_X T$	
true	false	.0095	I=true, X=true
false	false	.4655	I=false, X=false

Note how we recorded the value true of X in the first row since this is the value of X that corresponds to the maximized probability .0095. Similarly, we recorded the value false of X in the second row for the same reason. There are situations where multiple values of the maximized variable will lead to a maximal probability. Recording any of these values will do in such a case.

In general, if table $T_2(\mathbf{Y})$ was obtained by maximizing over variable X in table $T_1(\mathbf{X})$, then

$$T_2[\mathbf{y}] = \mathsf{append}(x, T_1[\mathbf{y}x]),$$

where x is a value of variable X which maximizes $T_1(\mathbf{y}, x)$. We also need to modify the operation of table multiplication so it updates the instantiation associated with each row. Specifically, if $T(\mathbf{Z})$ is the multiplication of tables $T_1(\mathbf{X})$ and $T_2(\mathbf{Y})$, then

$$T[\mathbf{z}] = \mathsf{append}(T_1[\mathbf{x}], T_2[\mathbf{y}]),$$

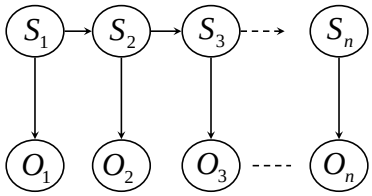


Figure 5.13: A directed acyclic graph known as a Hidden Markov Model.

where $\mathbf{x}$ and $\mathbf{y}$ are compatible with $\mathbf{z}$. Note that append can be implemented as a constant–time operation, which simply returns an object that points to the two instantiations being appended. Hence, associating an instantiation with each row does not change the complexity of table operations.

Given these more refined tables and their corresponding operations, we can use VE-MPE to compute an MPE in addition to its probability. Specifically, we start the algorithm with each CPT assigning the empty instantiation to each of its rows. When the algorithm terminates on line 9, the trivial table T will contain both an MPE, $T[\emptyset]$, and its probability, $T(\emptyset)$.

Before we consider an example of VE-MPE, consider again the class of Bayesian networks known as Hidden Markov Models; see Figure 5.13. If we compute MPE for an HMM using variable elimination, evidence $o_1, \ldots, o_n$, and variable order $\pi = O_1, S_1, O_2, S_2, \ldots, O_n, S_n$, we obtain an algorithm which corresponds to the *Viterbi Algorithm.* This is a specialized algorithm proposed specifically for HMMs, where the MPE it computes is known as a *most probable state path.* Moreover, if we compute the probability of evidence $o_1, \ldots, o_n$ using variable elimination and the previous order, we obtain an algorithm which corresponds to the *Forward Algorithm.* This is also a specialized algorithm proposed specifically for reasoning with HMMs, where the probability it computes is known as the *sequence probability.*

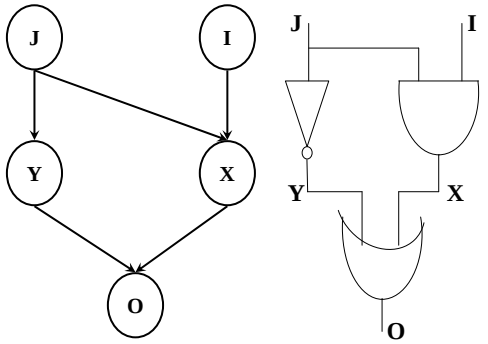
5.9.2 An Example of Computing MPE

Let us now consider an example of using VE-MPE to compute an MPE and its probability for the Bayesian network in Figure 5.14 and evidence J=true, O=false. We first prune edges in the network, leading to the network in Figure 5.15. We then compute the elimination order J, I, X, Y, O according to min–degree. Reducing network CPTs with the given evidence leads to the following set of tables:

I	$\Theta_I{}^{\mathbf{e}}$
true	.5
false	.5

J	$\Theta_J{}^{\mathbf{e}}$
true	.5

Y	$\Theta_Y{}^{\mathbf{e}}$
true	.01
false	.99



I	Θ_I
true	.5
false	.5

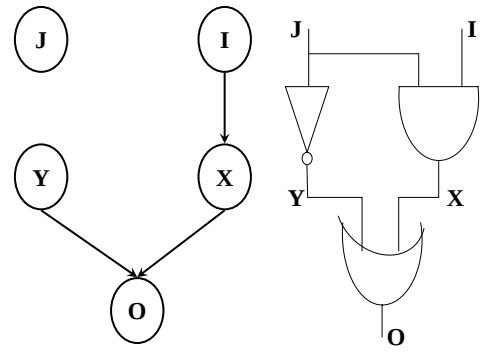
J	Θ_J
true	.5
false	.5

J	Y	$\Theta_{Y\|J}$
true	true	.01
true	false	.99
false	true	.99
false	false	.01

I	J	X	$\Theta_{X\|IJ}$
true	true	true	.95
true	true	false	.05
true	false	true	.05
true	false	false	.95
false	true	true	.05
false	true	false	.95
false	false	true	.05
false	false	false	.95

X	Y	O	$\Theta_{O\|XY}$
true	true	true	.98
true	true	false	.02
true	false	true	.98
true	false	false	.02
false	true	true	.98
false	true	false	.02
false	false	true	.02
false	false	false	.98

Figure 5.14: A Bayesian network modeling the behavior of a digital circuit.



I	Θ_I
true	.5
false	.5

J	Θ_J
true	.5
false	.5

Y	Θ_Y
true	.01
false	.99

I	X	$\Theta_{X\|I}$
true	true	.95
true	false	.05
false	true	.05
false	false	.95

X	Y	O	$\Theta_{O\|XY}$
true	true	true	.98
true	true	false	.02
true	false	true	.98
true	false	false	.02
false	true	true	.98
false	true	false	.02
false	false	true	.02
false	false	false	.98

Figure 5.15: Pruning edges in network of Figure 5.14 given J=true, O=false.

I	X	$\Theta_{X\|I}{}^{\mathbf{e}}$
true	true	.95
true	false	.05
false	true	.05
false	false	.95

X	Y	O	$\Theta_{O\|XY}{}^{\mathbf{e}}$
true	true	false	.02
true	false	false	.02
false	true	false	.02
false	false	false	.98

Note that each of the above tables assigns the empty instantiation to each of its rows. VE-MPE will then evaluate the following expression:

$$\max_{J,I,X,Y,O} \Theta_I{}^{\mathbf{e}}\ \Theta_J{}^{\mathbf{e}}\ \Theta_Y{}^{\mathbf{e}}\ \Theta_{X|I}{}^{\mathbf{e}}\ \Theta_{O|XY}{}^{\mathbf{e}}$$
$$= \left(\max_J \Theta_J{}^{\mathbf{e}}\right) \max_O \left(\max_Y \left(\max_X \left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X|I}{}^{\mathbf{e}}\right)\Theta_{O|XY}{}^{\mathbf{e}}\right)\Theta_Y{}^{\mathbf{e}}\right)$$

All intermediate tables constructed during this evaluation are shown below.

$\max_J \Theta_J{}^{\mathbf{e}}$	
.5	J=true

I	X	$\Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}$
true	true	.475
true	false	.025
false	true	.025
false	false	.475

X	$\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}$	
true	.475	I=true
false	.475	I=false

X	Y	O	$\left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}\right)\Theta_{O\|XY}{}^{\mathbf{e}}$	
true	true	false	.0095	I=true
true	false	false	.0095	I=true
false	true	false	.0095	I=false
false	false	false	.4655	I=false

Y	O	$\max_X \left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}\right)\Theta_{O\|XY}{}^{\mathbf{e}}$	
true	false	.0095	I=true, X=true
false	false	.4655	I=false, X=false

Y	O	$\left(\max_X \left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}\right)\Theta_{O\|XY}{}^{\mathbf{e}}\right)\Theta_Y{}^{\mathbf{e}}$	
true	false	.000095	I=true, X=true
false	false	.460845	I=false, X=false

O	$\max_Y \left(\max_X \left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}\right)\Theta_{O\|XY}{}^{\mathbf{e}}\right)\Theta_Y{}^{\mathbf{e}}$	
false	.460845	I=false, X=false, Y=false

$\max_O \left(\max_Y \left(\max_X \left(\max_I \Theta_I{}^{\mathbf{e}}\Theta_{X\|I}{}^{\mathbf{e}}\right)\Theta_{O\|XY}{}^{\mathbf{e}}\right)\Theta_Y{}^{\mathbf{e}}\right)$	
.460845	I=false, X=false, Y=false, O=false

Algorithm 8 VE-MAP($\mathcal{N}$: a Bayesian network, $\mathbf{Q}$: a set of variables in network $\mathcal{N}$, $\mathbf{e}$: instantiation of some variables in network $\mathcal{N}$): returns an instantiation $\mathbf{q}$ which maximizes $\Pr(\mathbf{q}, \mathbf{e})$, together with the probability $\Pr(\mathbf{q}, \mathbf{e})$.

1: $\mathcal{S} \leftarrow$ VE-PR-F($\mathcal{N}, \mathbf{Q}, \mathbf{e}$)
2: $\pi \leftarrow$ variable order for $\mathbf{Q}$ based on the interaction graph of tables in $\mathcal{S}$
3: **for** $i = 1$ to $|\mathbf{Q}|$ **do**
4: $\quad T \leftarrow \prod_k T_k$, where T_k belongs to $\mathcal{S}$ and mentions variable $\pi(i)$
5: $\quad T_i \leftarrow \max_{\pi(i)} T$
6: $\quad \mathcal{S} \leftarrow \mathcal{S} - \{T_k\} \cup \{T_i\}$
7: Return $T[\emptyset]$ and $T(\emptyset)$, where T is the product of tables $\mathcal{S}$

$\left(\max_J \Theta_J{}^{\mathbf{e}}\right) \max_O \left(\max_Y \left(\max_X \left(\max_I \Theta_I{}^{\mathbf{e}} \Theta_{X|I}{}^{\mathbf{e}}\right) \Theta_{O|XY}{}^{\mathbf{e}}\right) \Theta_Y{}^{\mathbf{e}}\right)$	
.2304225	J=true, I=false, X=false, Y=false, O=false

Therefore, an MPE given evidence $\mathbf{e}$: J=true, O=false is:

$$I\text{=false}, J\text{=true}, X\text{=false}, Y\text{=false}, O\text{=false}.$$

Moreover, the probability of this MPE is .2304225.

5.10 Computing MAPs

To compute MAP for a set of variables $\mathbf{Q}$ and evidence $\mathbf{e}$ is to find an instantiation $\mathbf{q}$ of variables $\mathbf{Q}$ which maximizes the probability $\Pr(\mathbf{q}|\mathbf{e})$. Note that MPE is a special case of MAP, when $\mathbf{Q}$ contains all variables in the Bayesian network, which turns out to be much easier from a complexity viewpoint as we show in Section 5.12.

One can compute MAP using a combination of VE-PR and VE-MPE. Specifically, let $\mathbf{R}$ be the set of all variables not in $\mathbf{Q}$. We can first sum out variables $\mathbf{R}$ using VE-PR, which leads to the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$. We can then run VE-MPE on this marginal in order to return an instantiation $\mathbf{q}$ that maximizes the marginal $\Pr(\mathbf{Q}, \mathbf{e})$. When running VE-PR, we will return the tables on line 6 without multiplying them, which has the effect of keeping the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$ in factored form. This variation on VE-PR will be called VE-PR-F. Algorithm 8 provides pseudocode for computing a MAP and its probability, given a Bayesian network $\mathcal{N}$, a set of variables $\mathbf{Q}$, and some evidence $\mathbf{e}$.

5.10.1 MAP and Constrained Width

The complexity of VE-MAP is similar to that of VE-MPE. That is, given a Bayesian network with n variables, a variable order with width w, and a bound c on variable cardinality, the time and space complexity of VE-MAP are $O(nc^w)$. There is one key difference with VE-MPE though. The variable orders that we

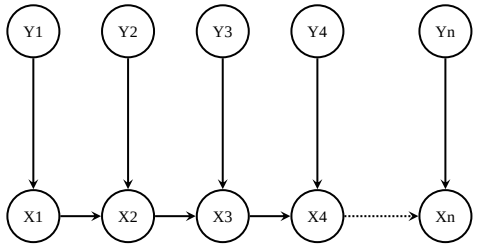


Figure 5.16: A polytree structure.

can use in VE-MAP are *constrained,* as we must eliminate all variables in **R** first, followed by all variables in **Q**. What this means is that we may not be able to use a good ordering (one with low width), simply because it may interleave the elimination of variables in **R** with variables in **Q**. Consider the network in Figure 5.16, which has a polytree structure. The treewidth of this network is 2 since we have at most two parents per node. Suppose now that we want to compute MAP for variables $\mathbf{Q} = \{Y_1, \ldots, Y_n\}$ with respect to this network. Any order in which variables **Q** come last must have width $\geq n$, even though we do have an unconstrained variable order with width 2. Hence, VE-MPE requires linear time in this case, while VE-MAP requires exponential time.

In general, we cannot arbitrarily interleave variables that we are summing out with those that we are maximizing over, because maximization does not commute with summing out. There are restricted cases where interleaving maximization and summation is possible, but this does not help computationally. To state this more formally, we need the following definitions.

Definition 8 *A variable order π is* **Q***–constrained iff variables* **Q** *appear last in the order π. The* **Q***–constrained treewidth of a Bayesian network is the width of its best* **Q***–constrained variable order.*

As it turns out, even though we may be able to correctly use variables elimination orders that are not **Q**–constrained for computing MAP, these orders will not be better than some **Q**–constrained order.

Theorem 6 *Every variable order π which can be used to compute MAP for variables* **Q** *and Bayesian network $\mathcal{N}$ must have a width which is no less than the* **Q***–constrained treewidth of network $\mathcal{N}$.*

For example, considering the polytree in Figure 5.16, even if we are able to identify a valid variable order in which the maximization variables $Y_1, \ldots, Y_n$ are interleaved with the summation variables $X_1, \ldots, X_n$, the width of that order cannot be less than n.

This all suggests that MAP is more difficult than MPE in the context of variable elimination. An interesting question then is whether this difference in complexity is genuine, or whether it is an idiosyncrasy of the variable elimination framework. We shed more light on this question when we discuss the complexity of probabilistic inference in Section 5.12

5.10.2 An Example of Computing MAP

Let us now consider an example of using VE-MAP to compute a MAP for the Bayesian network of Figure 5.14, with $\mathbf{Q} = \{I, J\}$ and $\mathbf{e} : O$=true. We will first compute a factored representation of the joint marginal $\Pr(\mathbf{Q}, \mathbf{e})$ using VE-PR-F, and then find an MPE for $\mathbf{Q}$ using the resulting marginal. If we run VE-PR-F using the elimination order O, Y, X, we obtain the following set of tables, which represent a factored representation of the joint marginal $\Pr(I, J, O$=true$)$:

I	J	T_1
true	true	.93248
true	false	.97088
false	true	.07712
false	false	.97088

I	T_2
true	.5
false	.5

J	T_3
true	.5
false	.5

These tables are the output of line 1 in VE-MAP. We will now trace the rest of VE-MAP using the elimination order $\pi : I, J$. To eliminate variable I, we multiply its tables and then maximize:

I	J	$T_1\ T_2$
true	true	.466240
true	false	.485440
false	true	.038560
false	false	.485440

J	$\max_I T_1\ T_2$	
true	.466240	I=true
false	.485440	I=true

To eliminate variable J, we multiply its tables and then maximize:

J	$\left(\max_I T_1\ T_2\right)\ T_3$	
true	.233120	I=true
false	.242720	I=true

$\max_J \left(\max_I T_1\ T_2\right)\ T_3$	
.242720	I=true, J=false

Therefore, the instantiation I=true, J=false is a MAP in this case, and the probability of I=false, J=false, O=true is .242720.

5.11 Bucket Elimination

The complexity analysis of VE-PR (Algorithm 6) assumes that we can identify all tables T_k that mention a particular variable $\pi(i)$ in time linear in the number of such tables. One method for achieving this is to arrange the tables maintained by the algorithm into *buckets,* where we have one bucket for each network variable. Consider the network in Figure 5.1 for an example, and suppose that we want to eliminate variables according to the order E, B, C, D, A. We can do this by

constructing the following buckets, which are initially populated with network CPTs:

Bucket Label	Bucket Tables
E	$\Theta_{E\|C}$
B	$\Theta_{B\|A},\ \Theta_{D\|BC}$
C	$\Theta_{C\|A}$
D	
A	Θ_A

That is, each CPT is placed in the first bucket whose label appears in the CPT. For example CPT $\Theta_{C|A}$ is placed in the bucket with label C because this is the first bucket whose label C appears in $\Theta_{C|A}$. The only other bucket whose label appears in $\Theta_{C|A}$ is the one for variable A, but that comes later in the order.

Given these buckets, we eliminate variables by processing buckets from top to bottom. When processing the bucket corresponding to some variable $\pi(i)$, we are guaranteed that the tables appearing in that bucket are exactly the tables that mention variable $\pi(i)$. This is true initially, and remains true after processing each bucket for the following reason. When processing the bucket of variable $\pi(i)$, we multiply all tables in that bucket, sum out variable $\pi(i)$, and then place the resulting table T in the first next bucket whose label appears in T. For example, after processing the bucket for variable E above, the resulting table $\sum_E \Theta_{E|C}$ is placed in the bucket for variable C:

Bucket Label	Bucket Tables
E	
B	$\Theta_{B\|A},\ \Theta_{D\|BC}$
C	$\Theta_{C\|A},\ \sum_E \Theta_{E\|C}$
D	
A	Θ_A

If our goal is to obtain the marginal for variables D and A, $\mathsf{Pr}(D, A)$, then we only process the first three buckets, E, B, and C. After such processing, the buckets for D and A will contain a factored representation of the marginal for these two variables. Again, we can either multiply these tables, or simply keep them in factored form.

Another variation on VE-PR is in how evidence is handled. In particular, given evidence, say $\mathbf{e}$: B=true, E=false, we do not reduce CPTs explicitly. Instead, we create two new tables:

D	$\lambda(B$=true$)$
true	1
false	0

E	$\lambda(E$=false$)$
true	0
false	1

and then add these tables to their corresponding buckets:

Bucket Label	Bucket Tables
E	$\Theta_{E\|C},\ \lambda(E{=}\mathsf{false})$
B	$\Theta_{B\|A},\ \Theta_{D\|BC},\ \lambda(B{=}\mathsf{true})$
C	$\Theta_{C\|A}$
D	
A	Θ_A

If we process buckets E, B and C, the last two buckets for D and A will then contain the joint marginal for these two variables, $\mathsf{Pr}(D, A, \mathbf{e})$. Again, we can either multiply the tables or keep them in factored form.

The tables $\lambda(B{=}\mathsf{true})$ and $\lambda(E{=}\mathsf{false})$ above are known as *evidence indicators.* Moreover, the above variation on VE-PR is known as the method of *bucket elimination.*

5.12 The Complexity of Probabilistic Reasoning

We will discuss in this section the complexity of probabilistic queries we considered so far: joint marginals, MPE and MAP. For joint marginals, we will consider the simpler query of computing the probability of evidence as this is sufficient to compute joint marginals. That is, to get the joint marginal for variables $\mathbf{Q}$ and evidence $\mathbf{e}$, it is sufficient to compute the probability of extended evidence $\mathbf{q}, \mathbf{e}$ for each instantiation $\mathbf{q}$. Note that the number of these simpler queries is exponential in $|\mathbf{Q}|$, but this is also the size of answer returned in this case, i.e., the size of joint marginal table.

As we show next, the above queries fall into three different complexity classes that are strongly believed to be distinct. To simplify the presentation, we will focus on the *decision problems* corresponding to these queries:

D-MPE: Given a rational number p, evidence $\mathbf{e}$, and the set of network variables $\mathbf{X}$, is there an instantiation $\mathbf{x}$ such that $\mathsf{Pr}(\mathbf{x}, \mathbf{e}) > p$?

D-PR: Given a rational number p and evidence $\mathbf{e}$, is $\mathsf{Pr}(\mathbf{e}) > p$?

D-MAP: Given a rational number p, evidence $\mathbf{e}$, and some set of variables $\mathbf{Q}$, is there an instantiation $\mathbf{q}$ such that $\mathsf{Pr}(\mathbf{q}, \mathbf{e}) > p$?

All of the above problems assume that the network parameters are rational numbers.

As it turns out, **D-MPE** is NP–complete, **D-PR** is PP–complete, and **D-MAP** is $\mathrm{NP}^{\mathrm{PP}}$–complete. We will not formally define these complexity classes here, but we will provide some intuition on the kind of problems they include and how they relate to one another.

5.12.1 Complexity Classes

Consider a propositional sentence α over binary variables $X_1, \ldots, X_n$. The following three problems are representatives of the above complexity classes:

SAT: Is there a world that satisfies α? This problem is NP–complete.

MAJSAT: Do the majority of worlds satisfy α? This problem is PP–complete.

E-MAJSAT: Is there an instantiation of variables $X_1,\ldots,X_k$, $1 \leq k \leq n$, under which the majority of worlds satisfy α? This problem is NP^{PP}–complete.

Intuitively, to solve an NP–complete problem we have to *search* for a solution among an exponential number of candidates, where it is easy to decide whether a given candidate constitutes a solution. For example, in **SAT**, we are searching for a world that satisfies a sentence (testing whether a world satisfies a sentence can be done in time linear in the sentence size). Similarly, in **D-MPE**, we are searching for a *complete* variable instantiation that is compatible with the evidence and that has a probability greater than some threshold (computing the probability of a complete variable instantiation can be done in time linear in the Bayesian network size using the Chain Rule for Bayesian networks).

To solve a PP–complete problem, we have to *add up* the weights of solutions, where it is easy to decide whether a particular candidate constitutes a solution and it is also easy to compute the weight of a solution. For example, in **MAJSAT**, a solution is a world that satisfies the sentence and the weight of a solution is 1. And in **D-PR**, a solution is a complete variable instantiation that is compatible with evidence and the weight of a solution is its probability.

Finally, to solve an NP^{PP}–complete problem, we have to search for a solution among an exponential number of candidates, but we also need to solve a PP–complete problem in order to decide whether a particular candidate constitutes a solution. For example, in **E-MAJSAT**, we are searching for an instantiation $x_1, \ldots, x_k$, but to test whether an instantiation satisfies the condition we want, we must solve an **MAJSAT** problem. Moreover, in **D-MAP**, we are searching for a *partial* variable instantiation $x_1, \ldots, x_k$ that is compatible with evidence, and that has a probability which exceeds a certain threshold. But to compute the probability of a partial instantiation, we need to solve a **D-PR** problem.

The class NP is included in the class PP, which is also included in the class NP^{PP}. Moreover, these classes are strongly believed to be distinct. This distinction implies that **D-MPE** is strictly easier than **D-PR**, which is then strictly easier than **D-MAP**. This also suggests that the use of variable elimination to solve **D-MPE** is actually an overkill, since the amount of work that variable elimination does is sufficient to solve a harder problem, **D-PR**. It also suggests that the additional penalty that we incur when using variable elimination to solve **D-MAP** is due to the intrinsic difficulty of **D-MAP**, as opposed to an idiosyncrasy of the variable elimination method.

5.12.2 Reductions

We will not formally prove the completeness of **D-MPE**, **D-PR**, and **D-MAP** for their corresponding classes here. Instead, we will illustrate how the corresponding queries can be used to solve the **SAT**, **MAJSAT**, and **E-MAJSAT** problems stated above. Our illustration will indeed constitute the core of a formal proof of completeness.

Suppose now that we have the following propositional sentence:

$$\alpha : (X_1 \vee X_2 \vee \neg X_3) \wedge ((X_3 \wedge X_4) \vee \neg X_5). \tag{5.2}$$

We can solve **SAT**, **MAJSAT**, and **E-MAJSAT** for this and similar sentences by constructing a corresponding Bayesian network, and reducing the previous queries to **D-MPE**, **D-PR**, and **D-MAP** queries on the network, respectively. The Bayesian network for a sentence α is constructed as follows:

1. We include a root node in the network for each variable X_i in the sentence.

2. We include a non–root node C for each logical connective in the sentence.

3. Each variable or connective node N will now represent a sub–sentence in α. We add an edge from each one of these sub–sentences N to the connective C that embodies N.

Figure 5.17 depicts the Bayesian network corresponding to the sentence α given in 5.2. Note that the resulting network will have only one leaf node, denoted by S, which represents the original sentence α. Moreover, the network will generally be multiply–connected since each root node may have multiple children. Finally, the number of parents per node in the resulting structure depends on the arity of logical connectives in sentence α.

Every node in the constructed network will have two values, true and false, where the network parameters are as follows. For each root node X_i, $\Theta_{x_i} = 1/2$ and $\Theta_{\overline{x}_i} = 1/2$. The CPTs for connectives are based on the usual interpretation of these connectives. Given this parameterization, the joint probability distribution induced by the network is such that:

$$\Pr(x_1, \ldots, x_n, S{=}\mathsf{true}) = \begin{cases} 0, & \text{if the instantiation } x_1, \ldots, x_n \text{ contradicts } \alpha; \\ 1/2^n & \text{if the instantiation } x_1, \ldots, x_n \text{ satisfies } \alpha. \end{cases} \tag{5.3}$$

We will not prove this equation here, but it can be used to show the following reductions.

Reducing SAT to D-MPE *There is a world that satisfies α iff there is an MPE for evidence S=*true *which has a non–zero probability.* This follows immediately from Equation 5.3.

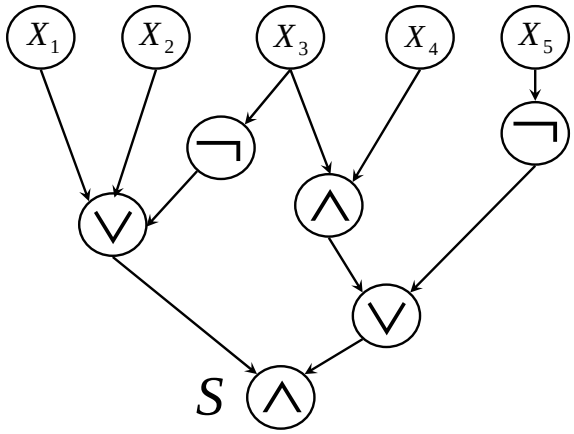


Figure 5.17: A Bayesian network representing a propositional sentence.

Reducing MAJSAT to D-PR *The majority of worlds satisfy α iff the probability of evidence S=true is greater than $1/2$.*

To prove this, note that there are 2^n worlds which correspond to the instantiations $x_1, \ldots, x_n$. If c is the number of instantiations that satisfy α, then the majority of instantiations satisfy α iff $c > 2^n/2$. Moreover,

$$\begin{aligned}
&\Pr(S{=}\mathsf{true}) \\
&= \sum_{x_1,\ldots,x_n \models \alpha} \Pr(x_1, \ldots, x_n, S{=}\mathsf{true}) + \sum_{x_1,\ldots,x_n \not\models \alpha} \Pr(x_1, \ldots, x_n, S{=}\mathsf{true}) \\
&= \frac{1}{2^n}\, c + 0 \quad \text{by Equation 5.3}
\end{aligned}$$

Hence, $c > 2^n/2$ iff $\Pr(S{=}\mathsf{true}) > 1/2$.

Reducing E-MAJSAT to D-MAP *There is an instantiation of variables $X_1, \ldots, X_k$ under which the majority of worlds satisfy α iff there is a MAP $x_1, \ldots, x_k$ for evidence S=true such that $\Pr(x_1, \ldots, x_k, S{=}\mathsf{true}) > 1/2^{k+1}$.*

To prove this, note that there are 2^{n-k} worlds which correspond to the instantiations $x_{k+1}, \ldots, x_n$. If c is the number of such instantiations such that $x_1, \ldots, x_n$ satisfy α, for a given $x_1, \ldots, x_k$, then the majority of these instantiations satisfy α iff $c > 2^{n-k}/2$. Moreover,

$$\begin{aligned}
&\Pr(x_1, \ldots, x_k, S{=}\mathsf{true}) \\
&= \sum_{x_1,\ldots,x_n \models \alpha} \Pr(x_1, \ldots, x_n, S{=}\mathsf{true}) + \sum_{x_1,\ldots,x_n \not\models \alpha} \Pr(x_1, \ldots, x_n, S{=}\mathsf{true}) \\
&= \frac{1}{2^n}\, c + 0 \quad \text{by Equation 5.3}
\end{aligned}$$

Hence, $c > 2^{n-k}/2$ iff $\Pr(x_1, \ldots, x_k, S{=}\mathsf{true}) > 1/2^{k+1}$.

5.12.3 Complexity of Networks with Bounded Treewidth

Suppose now that we have a Bayesian network with a treewidth w that is bounded by a constant. We know that we can obtain an optimal variable order for this network in linear time. Hence, both **D-MPE** and **D-PR** can now be solved in linear time using variable elimination. Unfortunately, **D-MAP** remains intractable for this class of Bayesian networks since **D-MAP** is known to be NP–complete even on polytrees which have at most two parents node, i.e., their treewidth is no more than 2 (see Figure 5.16). Specifically, it is known that using **D-MAP** on such polytrees, we are able to solve an NP–complete problem known as **MAXSAT**: *Given a set of clauses $\alpha_1, \ldots, \alpha_m$, and an integer k, is there a world that satisfies more than k of these clauses?* Therefore, even though the complexity of **D-MAP** is improved for polytrees (NP–complete instead of NP^{PP}–complete), the problem remains intractable for this class of networks.

Appendix A: Implementing Table Operations

Algorithm 9 MultiplyTables($T_1(\mathbf{X}_1), \ldots, T_n(\mathbf{X}_n)$): returns the table which results from multiplying Tables T_1,...,T_n.

```
1: Z←⋃_{i=1}^{n} X_i
2: T← a table over variables Z where T(z) = 1 for all z
3: for each instantiation z do
4:    for i = 1 to n do
5:       x_i← instantiation of variables X_i consistent with z
6:       T(z)←T(z)T_i(x_i)
7: return T
```

Algorithm 9 provides pseudocode for multiplying a number of probability tables $T_1, \ldots, T_n$ over variables $\mathbf{X}_1, \ldots, \mathbf{X}_n$, respectively. This multiplication process takes $O(c^w)$ time and space, where c is the maximum number of values that any variable in $\mathbf{Z} = \mathbf{X}_1 \cup \ldots \cup \mathbf{X}_n$ takes, and w is the number of variables in $\mathbf{Z}$. It should be noted that the step on line 5 can be implemented in time which is linear in the number of variables in $\mathbf{X}_i$, which explains the above complexity result for multiplication.

Algorithm 10 SumOutVars($T(\mathbf{X}), \mathbf{Z}$): returns a table which results from elimination variables $\mathbf{Z}$ from Table T.

```
1: Y←X − Z
2: T′← a table over variables Y where T(y) = 0 for all y
3: for each instantiation y do
4:    for each instantiation z do
5:       T′(y)←T′(y) + T(yz)
6: return T′
```

Algorithm 10 provides pseudocode for summing out a set of variables $\mathbf{Z}$ from Table $T(\mathbf{X})$. This process takes $O(c^w)$ time and space, where c is a bound on variable cardinality, and w is the number of variables in $\mathbf{X}$.