Probabilistic Reasoning

By
S.Komal Kaur,
Assistant Professor, CSE Dept.,
VCE

Outline

- Representing Knowledge in an Uncertain Domain
- The Semantics of Bayesian Networks
- Efficient Representation of Conditional Distributions
- Exact Inference in Bayesian Networks.
- Approximate Inference in Bayesian Networks.

Bayesian Network

- Introduces a data structure called a Bayesian network to represent the dependencies among variables.
- Bayesian networks can represent essentially any full joint probability distribution and in many cases can do so very concisely.
- A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:

Bayesian Network

1. Each node corresponds to a random variable, which may be discrete or continuous.
2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y , X is said to be a parent of Y. The graph has no directed cycles (and hence is a directed acyclic graph, or DAG.
3. Each node Xi has a conditional probability distribution P(Xi |Parents(Xi)) that quantifies the effect of the parents on the node.

A simple Bayesian Network

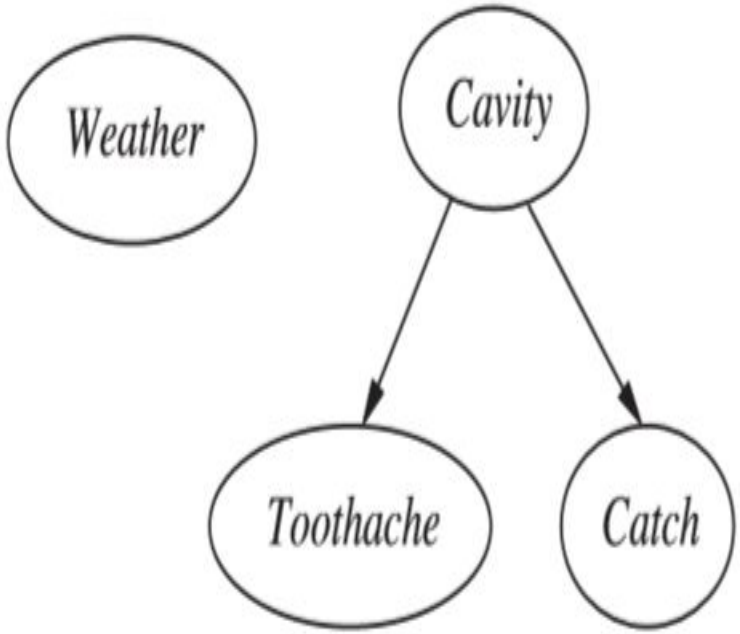


Figure 14.1 A simple Bayesian network in which *Weather* is independent of the other three variables and *Toothache* and *Catch* are conditionally independent, given *Cavity*.

Bayesian Network

- **Example:**
- You have a new burglar alarm installed at home.
- It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes.
- You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm.
- John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too.
- Mary, on the other hand, likes rather loud music and often misses the alarm altogether.

A Complex Bayesian Network

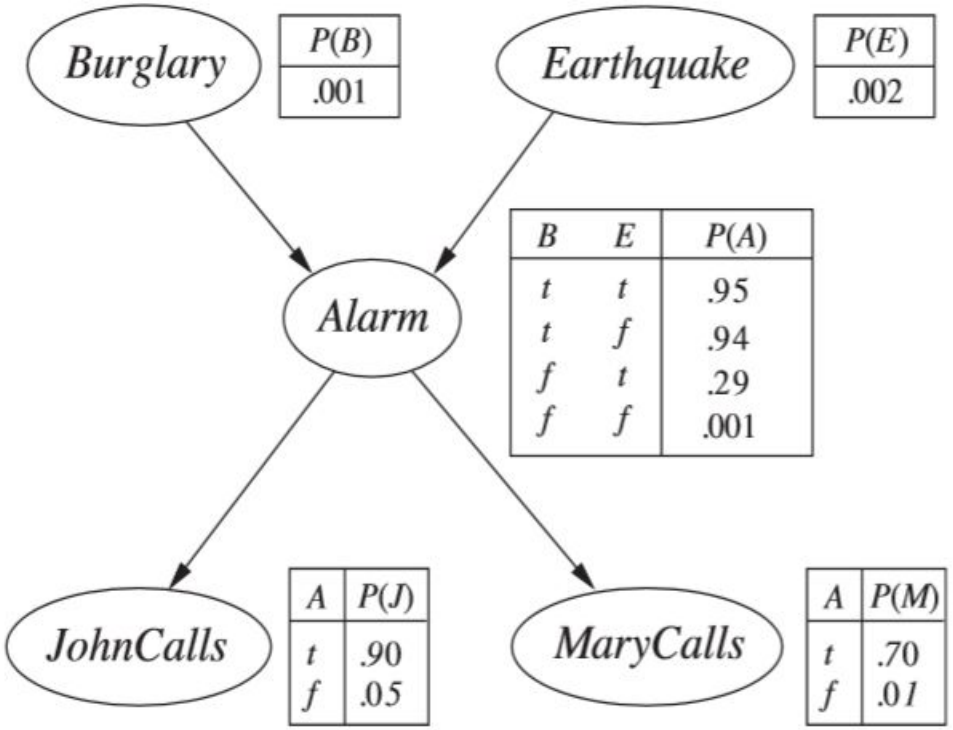


Figure 14.2 A typical Bayesian network, showing both the topology and the conditional probability tables (CPTs). In the CPTs, the letters B, E, A, J, and M stand for *Burglary*, *Earthquake*, *Alarm*, *JohnCalls*, and *MaryCalls*, respectively.

Contd..

- The conditional distributions in Figure 14.2 are shown as a conditional probability table, or CPT.
- Each row in a CPT contains the conditional probability of each node value for a conditioning case.
- A conditioning case is just a possible combination of values for the parent nodes

The Semantics of Bayesian Networks

- Representing the full joint distribution
- One way to define what the network means—its semantics—is to define the way in which it represents a specific joint distribution over all the variables.
- parameters assigned to each node correspond to conditional probabilities
- P(Xi |Parents(Xi))

The Semantics of Bayesian Networks

- A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as

$$P(X1 = x1 \wedge \ldots \wedge Xn = xn).$$

- We use the notation P(x1,...,xn) as an abbreviation for this. The value of this entry is given by the formula

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} \theta(x_i \,|\, parents(X_i)) \ ,$$

The Semantics of Bayesian Networks

- From this definition, it is easy to prove that the parameters θ(Xi |Parents(Xi)) are exactly the conditional probabilities P(Xi |Parents(Xi)) implied by the joint distribution.
- Hence, we can rewrite Equation (1) as

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \,|\, parents(X_i)) \ .$$

P(j, m, a, ¬b, ¬e)=P(j |a)P(m| a)P(a|¬b ∧¬e)P(¬b)P(¬e)

= 0 .90 × 0.70 × 0.001 × 0.999 × 0.998

= 0.000628 .

A method for reconstructing Bayesian Network

- The next step is to explain how to construct a Bayesian network in such a way that the resulting joint distribution is a good representation of a given domain.
- We will now show that Equation implies certain conditional independence relationships that can be used to guide the knowledge engineer in constructing the topology of the network.
- First, we rewrite the entries in the joint distribution in terms of conditional probability, using the product rule.

A method for reconstructing Bayesian Network

- P(x1,...,xn)=P(xn|xn−1,...,x1)P(xn−1,...,x1)
- Then we repeat the process, reducing each conjunctive probability to a conditional probability and a smaller conjunction.
- We end up with one big product:

$$P(x_1,\ldots,x_n) = P(x_n\,|\,x_{n-1},\ldots,x_1)P(x_{n-1}\,|\,x_{n-2},\ldots,x_1)\cdots P(x_2\,|\,x_1)P(x_1)$$
$$= \prod_{i=1}^{n} P(x_i\,|\,x_{i-1},\ldots,x_1)\,.$$

A method for reconstructing Bayesian Network

- This identity is called the chain rule. It holds for any set of random variables.
- The specification of the joint distribution is equivalent to the general assertion that, for every variable Xi in the network,

$$P(X_i \mid X_{i-1},\ldots,X_1) = P(X_i \mid \text{Parents}(X_i)) \quad \ldots(2)$$

provided that $\text{Parents}(X_i) \subseteq \{X_{i-1},\ldots,X_1\}$.

A method for reconstructing Bayesian Network

- Equation (2) says is that the Bayesian network is a correct representation of the domain only if each node is conditionally independent of its other predecessors in the node ordering, given its parents.
- We can satisfy this condition with this methodology:

1. **Nodes:** First determine the set of variables that are required to model the domain. Now order them, {X1,...,Xn}.

- Any order will work, but the resulting network will be more compact if the variables are ordered such that causes precede effects.

A method for reconstructing Bayesian Network

2. Links: For i = 1 to n do:

- Choose, from X1,...,Xi−1, a minimal set of parents for Xi, such that Equation (2) is satisfied.
- For each parent insert a link from the parent to Xi.
- CPTs: Write down the conditional probability table, P(Xi|Parents(Xi)).

Compactness and node ordering

- In a locally structured domain, we will get a compact Bayesian network only if we choose the node ordering well.
- What happens if we happen to choose the wrong order?
- Consider the burglary example again.
- Suppose we decide to add the nodes in the order MaryCalls, JohnCalls, Alarm, Burglary, Earthquake.
- We then get the somewhat more complicated network shown in Figure 14.3(a). The process goes as follows:

Compactness and node ordering

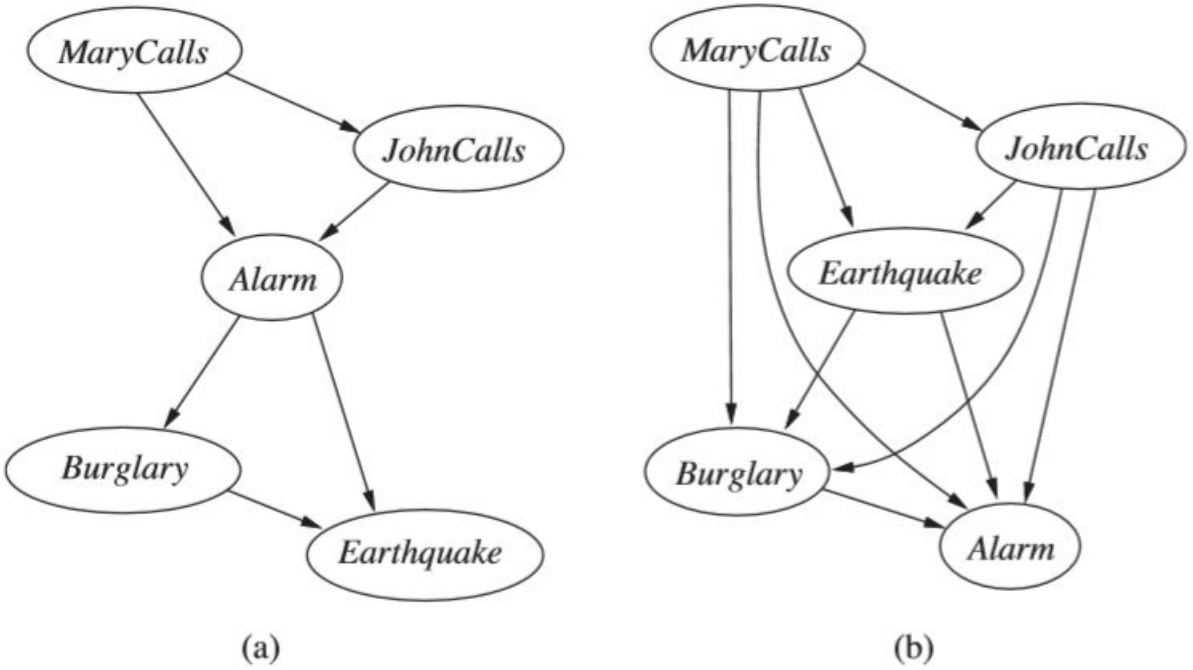


Figure 14.3 Network structure depends on order of introduction. In each network, we have introduced nodes in top-to-bottom order.

Compactness and node ordering

- Adding $MaryCalls$: No parents.
- Adding $JohnCalls$: If Mary calls, that probably means the alarm has gone off, which of course would make it more likely that John calls. Therefore, $JohnCalls$ needs $MaryCalls$ as a parent.
- Adding $Alarm$: Clearly, if both call, it is more likely that the alarm has gone off than if just one or neither calls, so we need both $MaryCalls$ and $JohnCalls$ as parents.
- Adding $Burglary$: If we know the alarm state, then the call from John or Mary might give us information about our phone ringing or Mary's music, but not about burglary:

$$\mathbf{P}(Burglary \mid Alarm, JohnCalls, MaryCalls) = \mathbf{P}(Burglary \mid Alarm) \ .$$

  Hence we need just $Alarm$ as parent.
- Adding $Earthquake$: If the alarm is on, it is more likely that there has been an earthquake. (The alarm is an earthquake detector of sorts.) But if we know that there has been a burglary, then that explains the alarm, and the probability of an earthquake would be only slightly above normal. Hence, we need both $Alarm$ and $Burglary$ as parents.

Conditional Independence relations in Bayesian Networks

- The topological semantics specifies that each variable is conditionally independent of its non-descendants, given its parents.
- For example, in Figure, JohnCalls is independent of Burglary, Earthquake, and MaryCalls given the value of Alarm.
- Another important independence property is implied by the topological semantics: a node is conditionally independent of all other nodes in the network, given its parents, children, and children's parents—that is, given its Markov blanket.
- For example, Burglary is independent of JohnCalls and MaryCalls, givenAlarm and Earthquake.

Conditional Independence relations in Bayesian Networks

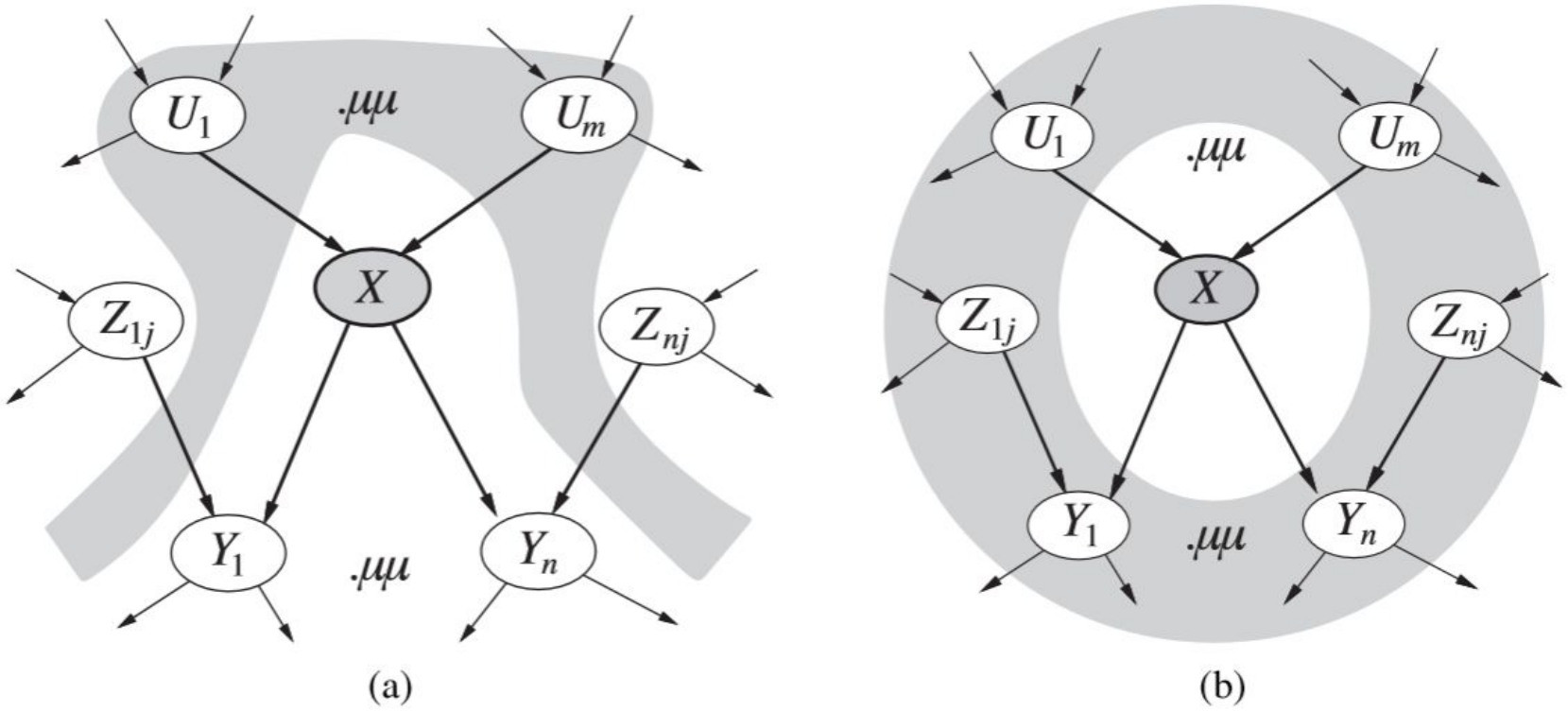


Figure 14.4 (a) A node X is conditionally independent of its non-descendants (e.g., the Z_{ij}s) given its parents (the U_is shown in the gray area). (b) A node X is conditionally independent of all other nodes in the network given its Markov blanket (the gray area).

Efficient Representation of Conditional Distributions

- The simplest example is provided by deterministic nodes.
- A deterministic node has its value specified exactly by the values of its parents, with no uncertainty.
- The relationship can also be numerical: for example, if the parent nodes are the prices of a particular model of car at several dealers and the child node is the price that a bargain hunter ends up paying, then the child node is the minimum of the parent values;

Efficient Representation of Conditional Distributions

- Uncertain relationships can often be characterized by so-called noisy logical relationships.
- The standard example is the noisy-OR relation, which is a generalization of the logical OR.
- In propositional logic, we might say that Fever is true if and only if Cold, Flu, or Malaria is true.
- The noisy-OR model allows for uncertainty about the ability of each parent to cause the child to be true—the causal relationship between parent and child may be inhibited, and so a patient could have a cold, but not exhibit a fever

Efficient Representation of Conditional Distributions

- First, it assumes that all the possible causes are listed.
- Second, it assumes that inhibition of each parent is independent of inhibition of any other parents: for example, whatever inhibits Malaria from causing a fever is independent of whatever inhibits Flu from causing a fever

Efficient Representation of Conditional Distributions

- Let us suppose these individual inhibition probabilities are as follows:

  q_{cold} = P(¬fever |cold,¬flu,¬malaria)=0.6 ,

  q_{flu} = P(¬fever |¬cold,flu,¬malaria)=0.2 ,

  $q_{malaria}$ = P(¬fever |¬cold,¬flu,malaria)=0.1 .

  Then, from this information and the noisy-OR assumptions, the entire CPT can be built.

The general rule is that

$$P(x_i \,|\, parents(X_i)) = 1 - \prod_{\{j:X_j = true\}} q_j \ ,$$

Where the product is taken over the parents that are set to true for that row of the CPT. The following table illustrates this calculation:

Cold	*Flu*	*Malaria*	$P(Fever)$	$P(\neg Fever)$
F	F	F	0.0	1.0
F	F	T	0.9	**0.1**
F	T	F	0.8	**0.2**
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	**0.6**
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

Bayesian nets with continuous variables

- Many real-world problems involve continuous quantities, such as height, mass, temperature, and money; infact, much of statistics deals with random variables whose domains are continuous.
- By definition, continuous variables have an infinite number of possible values, so it is impossible to specify conditional probabilities explicitly for each value.

Bayesian nets with continuous variables

- One possible way to handle continuous variables isto avoid them by using discretization—that is, dividing up the possible values into a fixed set of intervals.
- For example, temperatures could be divided into ($<0_oC$), (0_oC-100_oC), and ($>100_oC$).
- Discretization is sometimes an adequate solution, but often results in a considerable loss of accuracy and very large CPTs.
- The most common solution is to define standard families of probability density functions that are specified by a finite number of parameters.

Bayesian nets with continuous variables

- For example, a Gaussian (or normal) distribution $N(\mu,\sigma_2)(x)$ has the mean μ and the variance σ_2 as parameters.
- Yet another solution—sometimes called a nonparametric representation—is to define the conditional distribution implicitly with a collection of instances, each containing specific values of the parent and child variables.

Bayesian nets with continuous variables

- A network with both discrete and continuous variables is called a hybrid Bayesian network.
- To specify a hybrid network, we have to specify two new kinds of distributions:

A) The conditional distribution for a continuous variable given discrete or continuous parents;

B) And the conditional distribution for a discrete variable given continuous parents

Hybrid Bayesian network

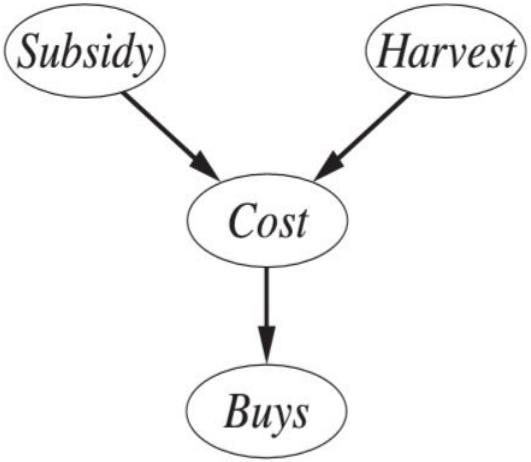


Figure 14.5 A simple network with discrete variables (*Subsidy* and *Buys*) and continuous variables (*Harvest* and *Cost*).

Consider the simple example in Figure 14.5, in which a customer buys some fruit depending on its cost, which depends in turn on the size of the harvest and whether the government's subsidy scheme is operating. The variable Cost is continuous and has continuous and discrete parents; the variable Buys is discrete and has a continuous parent.

Bayesian nets with continuous variables

- For the Cost variable, we need to specify P(Cost |Harvest, Subsidy).
- The discrete parent is handled by enumeration—that is, by specifying both P(Cost|Harvest, subsidy) and P(Cost |Harvest, ¬subsidy).
- To handle Harvest, we specify how the distribution over the cost c depends on the continuous value h of Harvest.

$$P(c \mid h, subsidy) = N(a_t h + b_t, \sigma_t^2)(c) = \frac{1}{\sigma_t \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_t h+b_t)}{\sigma_t}\right)^2}$$

$$P(c \mid h, \neg subsidy) = N(a_f h + b_f, \sigma_f^2)(c) = \frac{1}{\sigma_f \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{c-(a_f h+b_f)}{\sigma_f}\right)^2} .$$

Bayesian nets with continuous variables

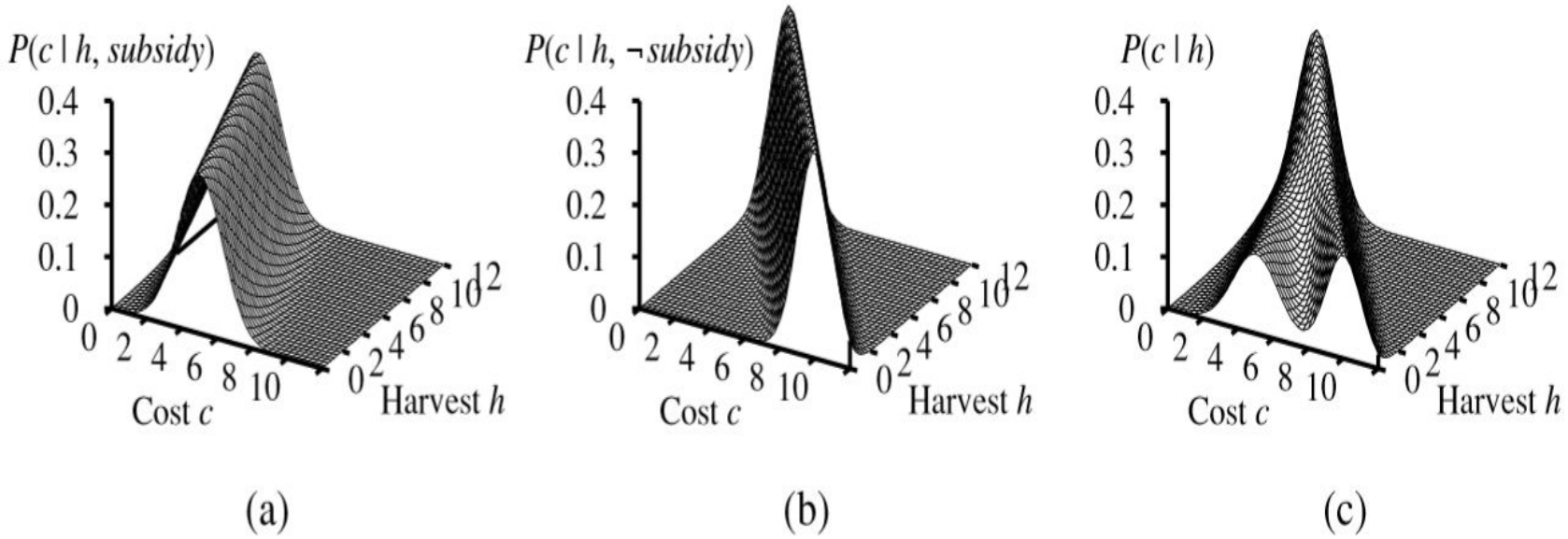


Figure 14.6 The graphs in (a) and (b) show the probability distribution over *Cost* as a function of *Harvest* size, with *Subsidy* true and false, respectively. Graph (c) shows the distribution $P(Cost \mid Harvest)$, obtained by summing over the two subsidy cases.

Bayesian nets with continuous variables

- Now we turn to the distributions for discrete variables with continuous parents. Consider, for example, the Buys node in Figure 14.5.
- It seems reasonable to assume that the customer will buy if the cost is low and will not buy if it is high and that the probability of buying varies smoothly in some intermediate region.

Bayesian nets with continuous variables

- It uses the logistic function 1/(1 + e−x) to produce a soft threshold.

$$P(buys \mid Cost = c) = \frac{1}{1 + exp(-2\frac{-c+\mu}{\sigma})} .$$

Exact Inference in Bayesian Networks.

- The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of query variables, given some observed event—that is, some assignment of values to a set of evidence variables.
- X denotes the query variable; E denotes the set of evidence variables E1,...,Em, and e is a particular observed event; Y will denotes the non-evidence, non query variables $Y_1,...,Y_l$ (called the hidden variables). Thus, the complete set of variables is $X=\{X\} \cup E \cup Y$.

Posterior Probability distribution P(X |e).

Exact Inference in Bayesian Networks

- In the burglary network, we might observe the event in which JohnCalls =true and MaryCalls =true.
- The probability that a burglary has occurred

P(Burglary |JohnCalls = true, MaryCalls =true)

=<0.284,0.716> .

Inference by enumeration

- Any conditional probability can be computed by summing terms from the full joint distribution.
- More specifically, a query P(X |e) can be answered, which we repeat here for convenience:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) .$$

Inference by enumeration

- A query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.
- Consider the query
- P(Burglary |JohnCalls =true,MaryCalls =true). The hidden variables for this query are Earthquake and Alarm.

Inference by enumeration

- we have

$$\mathbf{P}(B \mid j, m) = \alpha \mathbf{P}(B, j, m) = \alpha \sum_e \sum_a \mathbf{P}(B, j, m, e, a,) \ .$$

- The semantics of Bayesian networks then gives us an expression in terms of CPT entries. For simplicity, we do this just for Burglary =true:

$$P(b \mid j, m) = \alpha \sum_e \sum_a P(b) P(e) P(a \mid b, e) P(j \mid a) P(m \mid a) \ .$$

Inference by enumeration

- To compute this expression, we have to add four terms, each computed by multiplying five numbers

$$P(b|j,m)=$$

$$P(b,j,m,a,e)+P(b,j,m,\neg a,e)+P(b,j,m,a,\neg e)+P(b,j,m,\neg a,\neg e)$$

Inference by enumeration

- An improvement can be obtained from the following simple observations: the P(b) term is a constant and can be moved outside the summations over a and e, and the P(e) term can be moved outside the summation over a.
- Hence, we have

$$P(b \mid j,m) = \alpha P(b) \sum_{e} P(e) \sum_{a} P(a \mid b,e) P(j \mid a) P(m \mid a) . \quad \text{.....(1)}$$

Inference by enumeration

- This expression can be evaluated by looping through the variables in order, multiplying CPT entries as we go.
- For each summation, we also need to loop over the variable's possible values.
- The structure of this computation is shown in Figure

Inference by enumeration

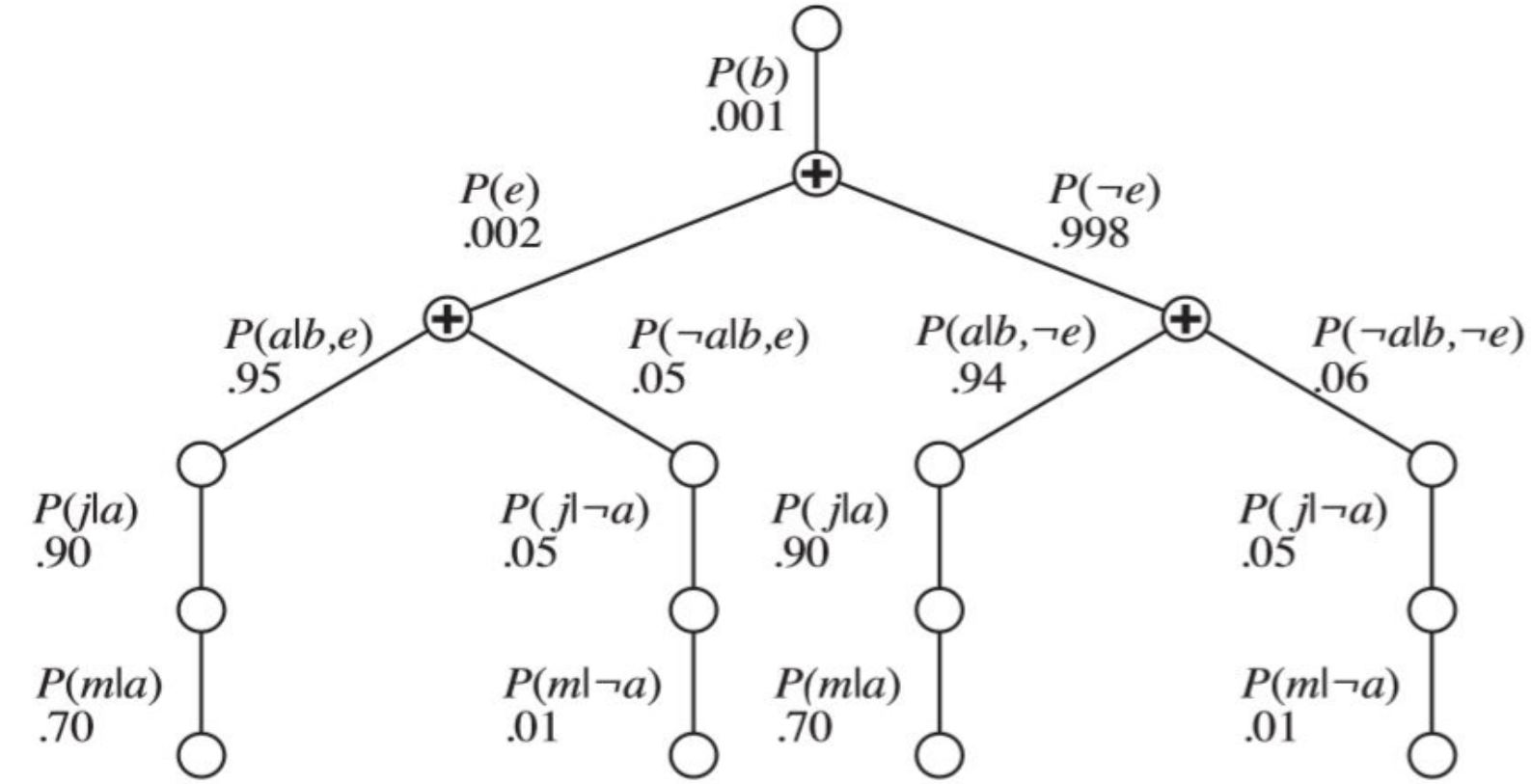


Figure 14.8 The structure of the expression shown in Equation (14.4). The evaluation proceeds top down, multiplying values along each path and summing at the "+" nodes. Notice the repetition of the paths for j and m.

Inference by enumeration

- The evaluation process for the expression in Equation (1) is shown as an expression tree in Figure 14.8.
- The ENUMERATION-ASK algorithm in Figure evaluates such trees using depth-first recursion.
- The algorithm is very similar in structure to the backtracking algorithm for solving CSPs.

Enumeration Algorithm

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

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

---

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

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

The variable elimination algorithm

- The enumeration algorithm can be improved substantially by eliminating repeated calculations of the kind illustrated in Figure 14.8.
- The idea is simple: do the calculation once and save the results for later use.
- This is a form of dynamic programming.

The variable elimination algorithm

- There are several versions of this approach; we present the variable elimination algorithm, which is the simplest.
- Variable elimination works by evaluating expressions such as Equation (1) in right-to-left order(that is, bottom up in Figure 14.8).
- Intermediate results are stored, and summations over each variable are done only for those portions of the expression that depend on the variable.

The variable elimination algorithm

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

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

- Notice that we have annotated each part of the expression with the name of the corresponding factor;
- Each factor is a matrix indexed by the values of its argument variables

The variable elimination algorithm

- For example, the factors f4(A) and f5(A) corresponding to P(j |a) and P(m|a) depend just on A because J and M are fixed by the query.
- They are therefore two-element vectors:

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

The variable elimination algorithm

- The query expression can be written as

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

- where the "×" operator is not ordinary matrix multiplication but instead the point-wise product operation, to be described shortly.
- The process of evaluation is a process of summing out variables (right to left) from point-wise products of factors to produce new factors, eventually yielding a factor that is the solution, i.e., the posterior distribution over the query variable.

The variable elimination algorithm

- The steps are as follows:
- First, we sum out A from the product of f3, f4, andf5. This gives us a new 2 × 2 factor f6(B,E) whose indices range over just B and E:

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

Now we are left with the expression

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

The variable elimination algorithm

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

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

This leaves the expression

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

Operations on factors

- The point-wise product of two factors f1 and f2 yields a new factor f whose variables are the union of the variables in f1 and f2 and whose elements are given by the product of the corresponding elements in the two factors. Suppose the two factors have variables Y1,...,Yk in common.

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

Point-wise Multiplication

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

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

The variable elimination algorithm

- Summing out a variable from a product of factors is done by adding up the sub matrices formed by fixing the variable to each of its values in turn.
- For example, to sum out A from f3(A,B,C), we write

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

The variable elimination algorithm

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

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

The variable elimination algorithm

- Now the point-wise product inside the summation is computed, and the variable is summed out of the resulting matrix.
- Notice that matrices are not multiplied until we need to sum out a variable from the accumulated product.
- At that point, we multiply just those matrices that include the variable to be summed out.

Variable Elimination Algorithm

function ELIMINATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $factors \leftarrow [\,]$
  for each var **in** ORDER(bn.VARS) **do**
    $factors \leftarrow [\text{MAKE-FACTOR}(var, \mathbf{e}) | factors]$
    if var is a hidden variable **then** $factors \leftarrow \text{SUM-OUT}(var, factors)$
  return NORMALIZE(POINTWISE-PRODUCT($factors$))

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

Variable Elimination Algorithm

- Continuing this process, we eventually find that every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.
- A variable elimination algorithm can therefore remove all these variables before evaluating the query.

$P(b \mid j,m)$

$$P(b \mid j,m) = \alpha \sum_a \sum_e P(j|a)P(m|a)P(a|b,e)P(b)P(e)$$

$$= \alpha P(b) \sum_a P(j|a)P(m|a) \sum_e P(a|b,e)P(e)$$

$$= \alpha P(b) \sum_a P(j|a)P(m|a)(P(a|b,e)P(e) + P(a|b,\neg e)P(\neg e))$$

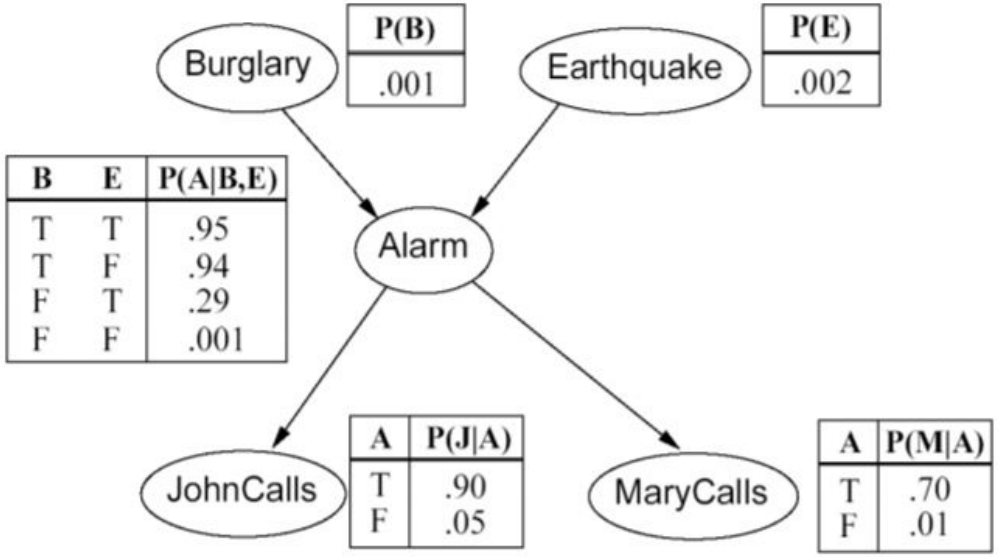
$$= \alpha P(b) \, (P(j|a)P(m|a) \, (P(a|b,e)P(e) + P(a|b,\neg e)P(\neg e))$$

$$+ P(j|\neg a)P(m|\neg a)(P(\neg a|b,e)P(e) + P(\neg a|b,\neg e)P(\neg e)))$$

$$= \alpha * .001*(.9*.7*(.95*.002 + .94*.998)$$

$$+.05*.01*(.05*.002 + .71*.998))$$

$$= \alpha * .00059$$



Clustering Algorithm

- The variable elimination algorithm is simple and efficient for answering individual queries.
- If we want to compute posterior probabilities for all the variables in a network, however, it can be less efficient.
- For example, in a polytree network, one would need to issue O(n) queries costing O(n) each, for a total of O(n2) time.

Clustering Algorithm

- Using clustering algorithms (also known as join tree algorithms), the time can be reduced to O(n).
- For this reason, these algorithms are widely used in commercial Bayesian network tools.
- The basic idea of clustering is to join individual nodes of the network to form cluster nodes in such a way that the resulting network is a polytree.

Clustering Algorithm

- For example, the multiply connected network shown in Figure 14.12(a) can be converted into a polytree by combining the Sprinkler and Rain node into a cluster node called Sprinkler+Rain, as shown in Figure 14.12(b).
- The two Boolean nodes are replaced by a "meganode" that takes on four possible values: tt, tf, ft, and ff.
- The meganode has only one parent, the Boolean variable Cloudy, so there are two conditioning cases.

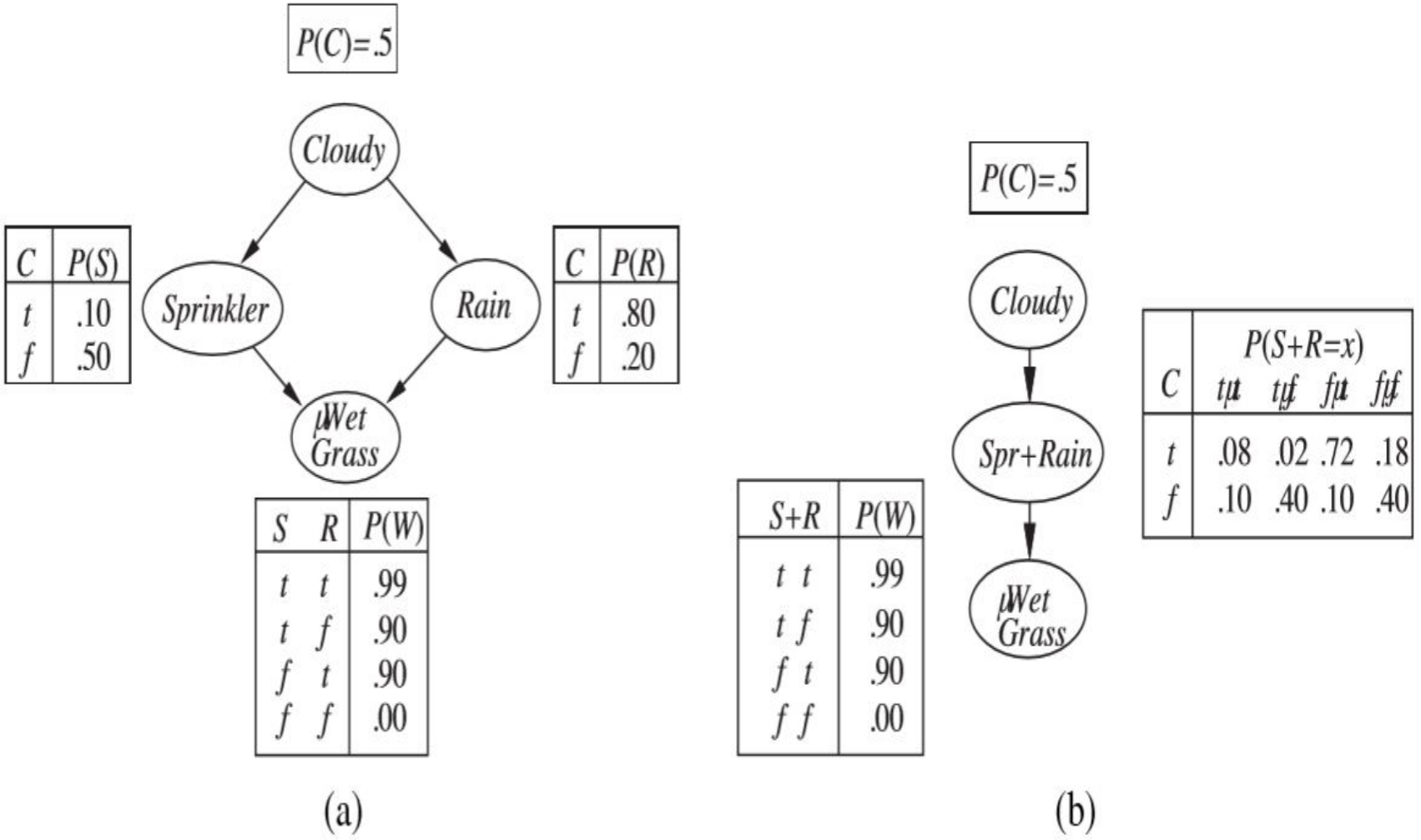


Figure 14.12 (a) A multiply connected network with conditional probability tables. (b) A clustered equivalent of the multiply connected network.

Approximate Inference in Bayesian Networks.

- Given the intractability of exact inference in large, multiply connected networks, it is essential to consider approximate inference methods.
- This section describes randomized sampling algorithms, also called Monte Carlo algorithms, that provide approximate answers whose accuracy depends on the number of samples generated.

Direct Sampling Method

- The primitive element in any sampling algorithm is the generation of samples from a known probability distribution.
- For example, an unbiased coin can be thought of as a random variable Coin with values <heads, tails> and a prior distribution P(Coin)=<0.5,0.5>.
- Sampling from this distribution is exactly like flipping the coin: with probability 0.5 it will return heads, and with probability 0.5 it will return tails.

Direct Sampling Method

- The simplest kind of random sampling process for Bayesian networks generates events from a network that has no evidence associated with it.
- The idea is to sample each variable in turn, in topological order.
- The probability distribution from which the value is sampled is conditioned on the values already assigned to the variable's parents.
- This algorithm is shown in Figure 14.13.

Sampling Algorithm

function PRIOR-SAMPLE(bn) **returns** an event sampled from the prior specified by bn
  inputs: bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with n elements
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return x

Figure 14.13 A sampling algorithm that generates events from a Bayesian network. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents.

Direct Sampling Method

- We can illustrate its operation on the network in Figure 14.12(a), assuming an ordering [Cloudy, Sprinkler, Rain, WetGrass]:

1. Sample from P(Cloudy)=<0.5,0.5>, value is true.
2. Sample from P(Sprinkler |Cloudy =true)=<0.1,0.9>, value is false.
3. Sample from P(Rain |Cloudy =true)=<0.8,0.2>, value is true.
4. Sample from P(WetGrass |Sprinkler =false,Rain =true)=<0.9,0.1>,value is true.

- In this case, PRIOR-SAMPLE returns the event [true,false,true,true].

Direct Sampling Method

- It is easy to see that PRIOR-SAMPLE generates samples from the prior joint distribution specified by the network.
- First, let SPS(x1,...,xn) be the probability that a specific event is generated by the PRIOR-SAMPLE algorithm. Just looking at the sampling process, we have

$$S_{PS}(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

Direct Sampling Method

- In any sampling algorithm, the answers are computed by counting the actual samples generated.
- Suppose there are N total samples, and let NPS(x1,...,xn) be the number of times the specific event x1,...,xn occurs in the set of samples.
- We expect this number, as a fraction of the total, to converge in the limit to its expected value according to the sampling probability:

Direct Sampling Method

$$\lim_{N\to\infty} \frac{N_{PS}(x_1,\ldots,x_n)}{N} = S_{PS}(x_1,\ldots,x_n) = P(x_1,\ldots,x_n)\,. \tag{14.5}$$

For example, consider the event produced earlier: $[true, false, true, true]$. The sampling probability for this event is

$$S_{PS}(true, false, true, true) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324\,.$$

Hence, in the limit of large N, we expect 32.4% of the samples to be of this event.

- Whenever we use an approximate equality ("≈") in what follows, we mean it in exactly this sense—that the estimated probability becomes exact in the large-sample limit.
- Such an estimate is called consistent.

Direct Sampling Method

- For example, one can produce a consistent estimate of the probability of any partially specified event x1,...,xm, where m ≤ n, as follows: P(x1,...,xm) ≈ NPS(x1,...,xm)/N .
- That is, the probability of the event can be estimated as the fraction of all complete events generated by the sampling process that match the partially specified event.
- we generate 1000 samples from the sprinkler network, and 511 of them have Rain =true, then the estimated probability of rain, written as P(Rain =true), is 0.511.

Rejection sampling in Bayesian networks

- Rejection sampling is a general method for producing samples from a hard-to-sample distribution given an easy-to-sample distribution. In its simplest form, it can be used to compute conditional probabilities—that is, to determine P(X |e).
- The algorithm is shown in Figure.
- First, it generates samples from the prior distribution specified by the network.
- Then, it rejects all those that do not match the evidence. Finally, the estimate ˆ P(X =x|e) is obtained by counting how often X =x occurs in the remaining samples.

Rejection sampling in Bayesian networks

- Let ˆ P(X |e) be the estimated distribution that the algorithm returns.
- From the definition of the algorithm, we have

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e}) = \frac{\mathbf{N}_{PS}(X, \mathbf{e})}{N_{PS}(\mathbf{e})} .$$

From Equation (14.6), this becomes

$$\hat{\mathbf{P}}(X \mid \mathbf{e}) \approx \frac{\mathbf{P}(X, \mathbf{e})}{P(\mathbf{e})} = \mathbf{P}(X \mid \mathbf{e}) .$$

Rejection sampling in Bayesian networks

- Continuing with our example from Figure 14.12(a), let us assume that we wish to estimate P(Rain |Sprinkler =true), using 100 samples.
- Of the 100 that we generate, suppose that 73 have Sprinkler =false and are rejected, while 27 have Sprinkler =true; of the 27, 8 have Rain =true and 19 have Rain =false.
- P(Rain|Sprinkler=true)≈ NORMALIZE(<8,19>)
- =<0.296,0.704>

Rejection sampling in Bayesian networks

- As more samples are collected, the estimate will converge to the true answer.
- The biggest problem with rejection sampling is that it rejects so many samples! The fraction of samples consistent with the evidence e drops exponentially as the number of evidence variables grows, so the procedure is simply unusable for complex problems.

Rejection Sampling Algorithm

function REJECTION-SAMPLING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network
    N, the total number of samples to be generated
  local variables: **N**, a vector of counts for each value of X, initially zero

  for $j = 1$ to N **do**
    $\mathbf{x} \leftarrow$ PRIOR-SAMPLE(bn)
    if **x** is consistent with **e** **then**
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x]+1$ where x is the value of X in **x**
  return NORMALIZE(**N**)

Figure 14.14 The rejection-sampling algorithm for answering queries given evidence in a Bayesian network.

LIKELIHOOD WEIGHTING

- Likelihood weighting avoids the in-efficiency of rejection sampling by generating only events that are consistent with the evidence e.
- LIKELIHOOD-WEIGHTING fixes the values for the evidence variables E and samples only the non-evidence variables.
- This guarantees that each event generated is consistent with the evidence. Not all events are equal, however.

LIKELIHOOD WEIGHTING

- Before tallying the counts in the distribution for the query variable, each event is weighted by the likelihood that the event accords to the evidence, as measured by the product of the conditional probabilities for each evidence variable, given its parents.
- Let us apply the algorithm to the network shown in Figure 14.12(a), with the query P(Rain|Cloudy =true,WetGrass =true) and the ordering Cloudy, Sprinkler, Rain, WetGrass. (Any topological ordering will do.) The process goes as follows:
- First, the weight w is set to 1.0. Then an event is generated:

LIKELIHOOD WEIGHTING

1. *Cloudy* is an evidence variable with value *true*. Therefore, we set

$$w \leftarrow w \times P(Cloudy = true) = 0.5 .$$

2. *Sprinkler* is not an evidence variable, so sample from $\mathbf{P}(Sprinkler \mid Cloudy = true) = \langle 0.1, 0.9 \rangle$; suppose this returns *false*.
3. Similarly, sample from $\mathbf{P}(Rain \mid Cloudy = true) = \langle 0.8, 0.2 \rangle$; suppose this returns *true*.
4. *WetGrass* is an evidence variable with value *true*. Therefore, we set

$$w \leftarrow w \times P(WetGrass = true \mid Sprinkler = false, Rain = true) = 0.45 .$$

LIKELIHOOD WEIGHTING

- Here WEIGHTED-SAMPLE returns the event [true,false,true,true] with weight 0.45, and this is tallied under Rain =true.
- To understand why likelihood weighting works, we start by examining the sampling probability SWS for WEIGHTED-SAMPLE.
- Remember that the evidence variables E are fixed with values e.
- We call the nonevidence variables Z (including the query variable X).
- The algorithm samples each variable in Z given its parent values:

$$S_{WS}(\mathbf{z}, \mathbf{e}) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \ .$$

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$

inputs: X, the query variable
e, observed values for variables **E**
bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
N, the total number of samples to be generated

local variables: **W**, a vector of weighted counts for each value of X, initially zero

for $j = 1$ to N **do**
x, $w \leftarrow$ WEIGHTED-SAMPLE(bn, **e**)
$\mathbf{W}[x] \leftarrow \mathbf{W}[x] + w$ where x is the value of X in **x**
return NORMALIZE(**W**)

---

function WEIGHTED-SAMPLE(bn, **e**) **returns** an event and a weight

$w \leftarrow 1$; **x** $\leftarrow$ an event with n elements initialized from **e**
foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
if X_i is an evidence variable with value x_i in **e**
then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
else **x**$[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
return **x**, w

Figure 14.15 The likelihood-weighting algorithm for inference in Bayesian networks. In WEIGHTED-SAMPLE, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

Inference by Markov chain simulation

- Markov chain Monte Carlo (MCMC) algorithms work quite differently from rejection sampling and likelihood weighting.
- Instead of generating each sample from scratch, MCMC algorithms generate each sample by making a random change to the preceding sample.
- It is therefore helpful to think of an MCMC algorithm as being in a particular current state specifying a value for every variable and generating a next state by making random changes to the current state.
- Here we describe a particular form of MCMC called Gibbs sampling, which is especially well suited for Bayesian networks.

Gibbs sampling in Bayesian networks

- The Gibbs sampling algorithm for Bayesian networks starts with an arbitrary state (with the evidence variables fixed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables Xi.
- The sampling for Xi is done conditioned on the current values of the variables in the Markov blanket of Xi.

Gibbs sampling in Bayesian networks

- The algorithm therefore wanders randomly around the state space—the space of possible complete assignments—flipping one variable at a time, but keeping the evidence variables fixed.
- Consider the query P(Rain |Sprinkler =true,WetGrass =true) applied to the network in Figure 14.12(a).

Gibbs sampling in Bayesian networks

- The evidence variables Sprinkler and WetGrass are fixed to their observed values and the nonevidence variables Cloudy and Rain are initialized randomly— let us say to true and false respectively.
- Thus, the initial state is [true,true,false,true]. Now the nonevidence variables are sampled repeatedly in an arbitrary order.

Gibbs sampling in Bayesian networks

- For example:

1. Cloudy is sampled, given the current values of its Markov blanket variables: in this case, we sample from P(Cloudy |Sprinkler = true, Rain =false). (Shortly, we will show how to calculate this distribution.) Suppose the result is Cloudy =false.

Then the new current state is [false, true, false, true].

2. Rain is sampled, given the current values of its Markov blanket variables: in this case, we sample from P(Rain |Cloudy = false, Sprinkler = true, WetGrass =true). Suppose this yields Rain =true.

The new current state is [false, true,true,true].

Gibbs sampling in Bayesian networks

- Each state visited during this process is a sample that contributes to the estimate for the query variable Rain.
- If the process visits 20 states where Rain is true and 60 states where Rain is false, then the answer to the query is NORMALIZE(<20,60>)=<0.25,0.75>.

Gibbs Algorithm

function GIBBS-ASK(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: **N**, a vector of counts for each value of X, initially zero
    Z, the nonevidence variables in bn
    x, the current state of the network, initially copied from **e**

  initialize **x** with random values for the variables in **Z**
  for $j = 1$ to N **do**
    for each Z_i in **Z do**
      set the value of Z_i in **x** by sampling from $\mathbf{P}(Z_i|mb(Z_i))$
      $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in **x**
  return NORMALIZE(**N**)

Figure 14.16 The Gibbs sampling algorithm for approximate inference in Bayesian networks; this version cycles through the variables, but choosing variables at random also works.