Artificial Intelligence

Probabilistic Inference

Christopher Simpkins

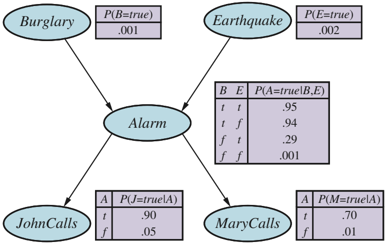
Kennesaw State University

Exact Inference in Bayesian Networks (AIMA 13.3)

Most common task in probabilistic inference: compute the *posterior probability* of a set of **query variables** given some **event** represented as a set of **evidence variables**.

Notation:

- Query variable: X
- Set of evidence variables: $\boldsymbol{E} = \{E_1, \ldots, E_m\}$
- Particular observed event: $\boldsymbol{e}$
- Hidden (nonevidence, nonquery) variables: $\boldsymbol{Y} = \{Y_1, \ldots, Y_l\}$
- Typical query: $Pr(X \mid e)$



Example:

- X is the boolean random variable $Burglary$
- $\boldsymbol{E} = \{JohnCalls, MaryCalls\}$
- $\boldsymbol{e} = \{JohnCalls = true, MaryCalls = true\}$
- $\boldsymbol{Y} = \{EarthQuake, Alarm\}$

$$Pr(Buglary \mid JohnCalls = true, MaryCalls = true) =< 0.284, 0.716 > .$$

Inference by Enumeration

Recall that we can use the full joint distribution to answer any query:

$$Pr(X|\boldsymbol{e}) = \alpha Pr(X, \boldsymbol{e}) = \alpha \sum_{y} Pr(X, \boldsymbol{e}, \boldsymbol{y}) \tag{12.9}$$

And that a Bayes net completely represents the full joint distribution, so we can reduce the computation of a joint to:

$$Pr(x_1, \ldots, x_n) = \prod_{i=1}^{n} Pr(x_i|parents(X_i)) \tag{13.2}$$

Using these two equations we can enumerate the appropriate probabilities to calculate the answer to any probabilistic query.

- In particular, we can get the answer by computing sums of products of conditional probabilities from a Bayes net.

Example: $Pr(Burglary \mid JohnCalls = true, MaryCalls = true)$.

Using abbreviations and substituting into Eq 12.9 above (e and a are hidden):

$$Pr(B \mid j, m) = \alpha Pr(B, j, m) = \alpha \sum_e \sum_a Pr(B, j, m, e, a)$$

Then we substitute Eq 13.2 for $Pr(B, j, m, e, a)$ to get (onyly showing Burglary=true):

$$\begin{aligned} Pr(b \mid j, m) &= \alpha \sum_e \sum_a Pr(b) Pr(e) Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a) && (1) \\ &= \alpha Pr(b) \sum_e \sum_a Pr(e) Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a) && (2) \\ &= \alpha Pr(b) \sum_e Pr(e) \sum_a Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a) && (3) \end{aligned}$$

1. Substitute Eq 13.2 for $Pr(B, j, m, e, a)$
2. Pull out $Pr(b)$ from summations because it doesn't depend on the other variable and is thus a constant in all the summation terms.
3. Pull out $Pr(e)$ from the summation over the a values because each value of e doesn't depend on the other variables in the summation over the a values and is thus a constant in the summation terms over the values of a.

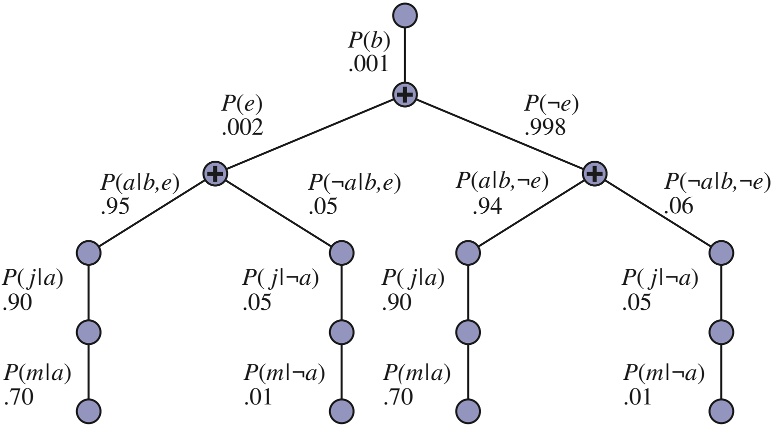
Steps 2 and 3 above reduce the complexity of the computation from $O(n2^n)$ to $O(2^n)$.

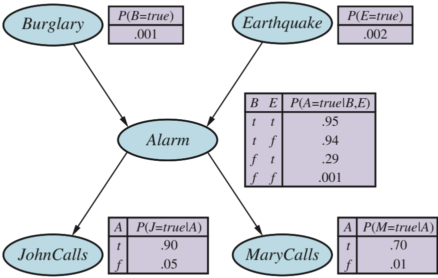
Caclulation of $Pr(b \mid j, m)$

Substiting the values from the CPTs in the Bayes net into

$$\alpha Pr(b) \sum_{e} Pr(e) \sum_{a} Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a)$$

we get the expression tree:





Enumeration Algorithm

The ENUMERATION-ASK algorithm evaluates these expression trees using depth-first, left-to-right recursion.

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

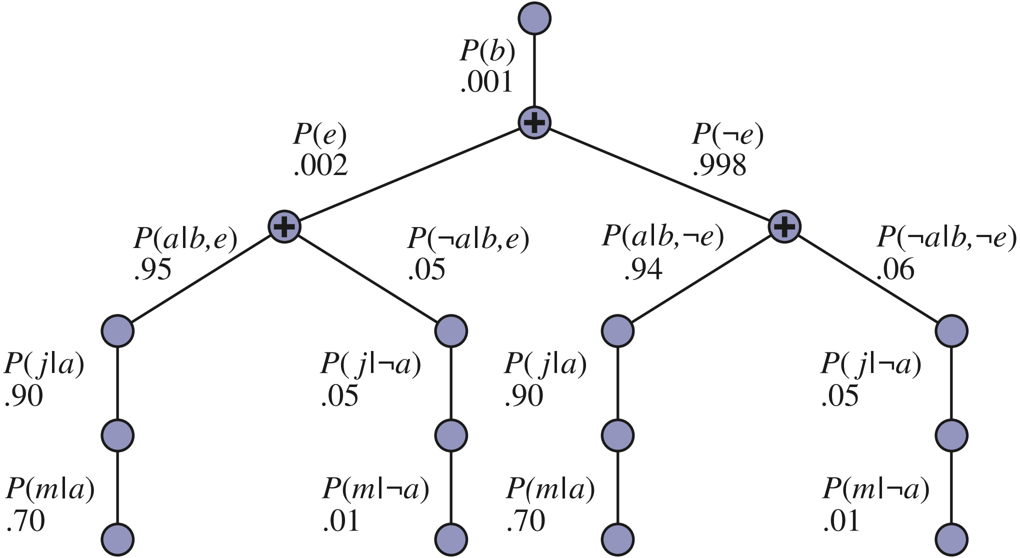
  $\mathbf{Q}(X) \leftarrow$ a distribution over X, initially empty
  for each value x_i of X **do**
    $\mathbf{Q}(x_i) \leftarrow$ ENUMERATE-ALL($vars$, $\mathbf{e}_{x_i}$)
      where $\mathbf{e}_{x_i}$ is $\mathbf{e}$ extended with $X = x_i$
  return NORMALIZE($\mathbf{Q}(X)$)

function ENUMERATE-ALL($vars$, $\mathbf{e}$) **returns** a real number
  if EMPTY?($vars$) **then return** 1.0
  $V \leftarrow$ FIRST($vars$)
  if V is an evidence variable with value v in $\mathbf{e}$
    then return $P(v \mid parents(V)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}$)
    else return $\sum_v P(v \mid parents(V)) \times$ ENUMERATE-ALL(REST($vars$), $\mathbf{e}_v$)
      where $\mathbf{e}_v$ is $\mathbf{e}$ extended with $V = v$

Unfortunately, its time complexity is $O(s^n)$. But we can improve it ...

Repeated Calculations

Notice that the subexpressions for the products $Pr(j \mid a)Pr(m \mid a)$ and $Pr(j \mid \neg a)Pr(m \mid \neg a)$ are computed twice, once for each value of E.



Variable Elimination

The enumeration algorithm can be improved substantially by eliminating repeated calculations.

- Idea: do the calculation once and save the results for later use.
- This is a form of dynamic programming.
- Several versions of this approach; variable elimination algorithm is simplest.

Variable elimination works by evaluating expressions such as

$$Pr(b \mid j, m) = \alpha Pr(b) \sum_{e} Pr(e) \sum_{a} Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a) \tag{13.5}$$

in right-to-left order (that is, bottom up in the expression tree), storing intermediate results, and only doing summations for portions of the expression that depend on the variable.

Example: Variable Elimination in Burglary Network

First, annotate the **factors** in the expression for the network:

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

- Each factor is a matrix indexed by the values of its argument variables.
- Notice that the factors for $Pr(j \mid a)$ and $Pr(m|a)$ do not include j and m. This is because the values of j and m ($JohnCalls = true$ and $MaryCalls - true$) are fixed by the query.

So the factors are:

$$\boldsymbol{f}_1(B) = \begin{bmatrix} Pr(b) \\ Pr(\neg b) \end{bmatrix} = \begin{bmatrix} 0.001 \\ 0.999 \end{bmatrix} \qquad \boldsymbol{f}_2(E) = \begin{bmatrix} Pr(e) \\ Pr(\neg e) \end{bmatrix} = \begin{bmatrix} 0.002 \\ 0.998 \end{bmatrix}$$

$$\boldsymbol{f}_4(A) = \begin{bmatrix} Pr(j \mid a) \\ Pr(j \mid \neg a) \end{bmatrix} = \begin{bmatrix} 0.090 \\ 0.05 \end{bmatrix} \qquad \boldsymbol{f}_5(A) = \begin{bmatrix} Pr(m \mid a) \\ Pr(m \mid \neg a) \end{bmatrix} = \begin{bmatrix} 0.070 \\ 0.01 \end{bmatrix}$$

$\boldsymbol{f}_3(A, B, E)$ is a little more complicated ...

$\boldsymbol{f}_3(A, B, E)$

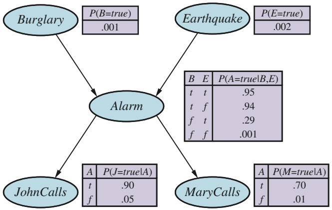
$$Pr(B \mid j, m) = \alpha\, \underbrace{Pr(B)}_{\boldsymbol{f}_1(B)} \sum_e \underbrace{Pr(e)}_{\boldsymbol{f}_2(E)} \sum_a \underbrace{Pr(a \mid B, e)}_{\boldsymbol{f}_3(A,B,E)} \underbrace{Pr(j \mid a)}_{\boldsymbol{f}_4(A)} \underbrace{Pr(m \mid a)}_{\boldsymbol{f}_5(A)}$$

$\boldsymbol{f}_3(A, B, E)$ is a $2 \times 2 \times 2$ matrix (or a rank-3 tensor). Here's one way to think about it:

- First index with A, yielding two 2×2 submatrices (one for each of the two values of A).
- Rows of each submatrix is indexed by B and columns by E.
- The entries in the submatrices are the values of $Pr(A \mid B, E)$

$$\boldsymbol{f}_3^{(a)}(B, E) = \begin{bmatrix} Pr(a \mid b, e) & Pr(a \mid b, \neg e) \\ Pr(a \mid \neg b, e) & Pr(a \mid \neg b, \neg e) \end{bmatrix} = \begin{bmatrix} 0.95 & 0.94 \\ 0.29 & 0.001 \end{bmatrix}$$

$$\boldsymbol{f}_3^{(\neg a)}(B, E) = \begin{bmatrix} Pr(\neg a \mid b, e) & Pr(\neg a \mid b, \neg e) \\ Pr(\neg a \mid \neg b, e) & Pr(\neg a \mid \neg b, \neg e) \end{bmatrix} = \begin{bmatrix} 0.05 & 0.06 \\ 0.71 & 0.999 \end{bmatrix}$$



Factorized Query

From our original query:

$$Pr(b \mid j, m) = \alpha Pr(b) \sum_e Pr(e) \sum_a Pr(a \mid b, e) Pr(j \mid a) Pr(m \mid a) \tag{13.5}$$

We annotated the factors:

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

And now we substitute the factor expressions for the original expresions so we can manipulate the factors using the **pointwise product** operation, denoted with $\times$ here:

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

Now we are ready to evaluate the expression ...

Expression Evaluation

First, sum out A from the pointwise product of $\boldsymbol{f}_3(A, B, E)$, $\boldsymbol{f}_4(A)$, and $\boldsymbol{f}_5(A)$ yielding a new 2×2 factor, $\boldsymbol{f}_6(B, E)$:

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

Now the query expression is $Pr(B \mid j, m) = \alpha \boldsymbol{f}_1(B) \times \sum_e \boldsymbol{f}_2(E) \times \boldsymbol{f}_6(B, E)$

Next, sum out E from the product of $\boldsymbol{f}_2(E)$ and $\boldsymbol{f}_6(B, E)$, yielding a new factor $\boldsymbol{f}_7(B)$:

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

Which leaves our final form of the query: $Pr(B \mid j, m) = \alpha \boldsymbol{f}_1(B) \times \boldsymbol{f}_7(B)$

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

Operations on Factors

Two basic operations in variable elimination:

1. the pointwise product operation, and
2. summing out hidden variables from products of factors.

Pointwise Product Example

The pointwise product of two factors $\boldsymbol{f}$ and $\boldsymbol{g}$ yields a new factor $\boldsymbol{h}$ whose variables are the union of the variables in $\boldsymbol{f}$ and $\boldsymbol{g}$ and whose elements are given by the product of the corresponding elements in the two factors.

Given X, Y, Z boolean variables, result of pointwise product $\boldsymbol{f}(X,Y) \times \boldsymbol{g}(Y,Z) = \boldsymbol{h}(X,Y,Z)$ is:

X	Y	$\mathbf{f}(X,Y)$	Y	Z	$\mathbf{g}(Y,Z)$	X	Y	Z	$\mathbf{h}(X,Y,Z)$
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	$.1 \times .4 = .04$

Summing out Variables

Summing out a variable from a product of factors is done by adding up the submatrices formed by fixing the variable to each of its values in turn. For example, to sum out X from $h(X, Y, Z)$, we write

$$
\begin{aligned}
\boldsymbol{h}_2(Y, Z) &= \sum_x \boldsymbol{h}(X, Y, Z) \\
&= \boldsymbol{h}(x, Y, Z) + \boldsymbol{h}(\neg x, Y, Z) \\
&= \begin{bmatrix} .06 & .24 \\ .42 & .28 \end{bmatrix} + \begin{bmatrix} .18 & .72 \\ .06 & .04 \end{bmatrix} \\
&= \begin{bmatrix} .24 & .96 \\ .48 & .32 \end{bmatrix}
\end{aligned}
$$

Variable Elimination Algorithm

With these two basic operations, we can implement the 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 with variables $vars$

  $factors \leftarrow [\,]$
  for each V **in** ORDER($vars$) **do**
    $factors \leftarrow$ [MAKE-FACTOR(V, **e**)] $+$ $factors$
    if V is a hidden variable **then** $factors \leftarrow$ SUM-OUT(V, $factors$)
  return NORMALIZE(POINTWISE-PRODUCT($factors$))

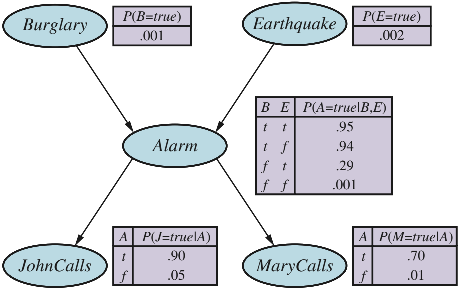
Notes about the `order` function:

- Any ordering works, some orderings lead to more efficient algorithms.
- No tractable algorithm for determining optimal ordering.
- One heuristic: eliminate whichever variable minimizes the size of the next factor to be contructed.
- General rule: every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.

Complexity of Exact Inference in Polytrees

Notice that the Alarm Bayes net is **singly connected**, a.k.a., a **polytree**:

- there is at mose one undirected path between any two nodes in the network.

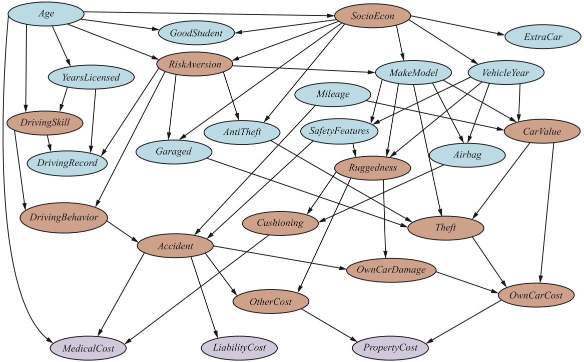


The time and space complexity of polytrees is linear in the size of the network.

- Size of network is defined as number of CPT entries.
- If $|parents(X_i)| \leq c, \forall i \in n$ for some constant c and number of nodes n, then complexity is also linear in number of nodes.

Complexity of Exact Inference in Multiply-connected Networks

Now consider **multiply-connected** networks such as the car insurance network:



- Variable elimination can have exponential worst-case time and space complexity in multiply-connected networks.
- *Since inference in Bayes nets includes inference in propositional logic as a special case, Bayes net inference is* ***NP-hard***.

Approximate Inference for Bayesian Networks

Exact inference in large Bayesian networks is intractable, so we need approximate inference methods. The mothods we'll learn are randomized sampling agorithms, a.k.a., **Monte Carlo** algorithms. They work by

- generating random events based on the probabilities in the Bayes net and
- counting up the different answers found in those random events.

The more the samples, the closer to the true probability distribution.

We'll learn two families of Monte carlo algorithms:

- direct sampling, and
- Markov chain sampling.

Direct Sampling Methods

The primitive element in any sampling algorithm is the generation of samples from a known probability distribution.

Feed a sample from U(1, 1) to the inverse CDF of a distribution to sample the distribution.

Prior Sampling

To sample from a Bayes net with no associated evidence, sample each node in topological order.

- Sample unconditioned nodes according to their prior distributions.
  - Samples become fixed values for sampling CPTs of child nodes.
- Continue sampling child nodes, in order, until all variables are sampled.

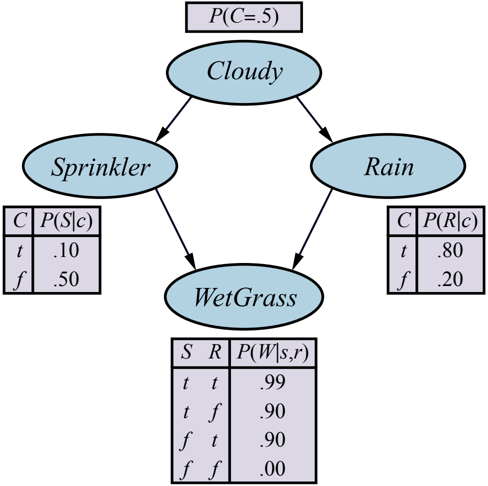
function PRIOR-SAMPLE(*bn*) **returns** an event sampled from the prior specified by *bn*
  inputs: *bn*, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with n elements
  for each 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

Example: Generating a Sample from Sprinkler Bayes Net

1. Sample from $Pr(Cloudy) = \langle 0.5, 0.5 \rangle$
   - Get value of true.
2. Sample from $Pr(Sprinkler \mid c) = \langle 0.1, 0.9 \rangle$
   - Get value of false.
3. Sample from $Pr(Rain \mid c) = \langle 0.8, 0.2 \rangle$
   - Get value of true.
4. Sample from $Pr(WetGrass \mid \neg s, r) = \langle 0.9, 0.1 \rangle$
   - Get value of true.

Full sampled event: $[true, false, true, true]$



From Sampling to Probability Estimates

A Bayes net represents a full joint distribution, so a sample generated from a Bayes net is a sample from the prior joint distribution over all the variables. So, if S_{PS} is a sample generated by the PRIOR-SAMPLE algorithm, then:

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

To estimate these probabilities using samples, we simply generate N samples, count the number of events of interest among the N samples, and divide that number by N. This should converge to the true probability:

$$\lim_{N \to \infty} \frac{N_{PS}(x_1, \ldots, x_n)}{N} = S_{PS}(x_1, \ldots, x_n) = Pr(x_1, \ldots, x_n)$$

From the CPTs in the Bayes net, the probability of the example event we sampled, $[true, false, true, true]$ is:

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

So if we generated $N = 1000$ samples, we would expect to see $[true, false, true, true]$ 324 times.

Consistent Probability Estimates

An estimate that becomes exact in the large-sample limit is called **consistent**. A consistent estimate of the probability of any partially speciified event $x_1, \ldots, x_m$ for $x \leq n$ is:

$$Pr(x_1, \ldots, x_m) \approx \frac{N_{PS}(x_1, \ldots, x_m)}{N} = \hat{P}r(x_1, \ldots, x_m)$$

That is, $\hat{P}r(x_1, \ldots, x_m)$ is an approximation of $Pr(x_1, \ldots, x_m)$ that converges to the true probability as N approaches ∞

Rejection Sampling

```
function REJECTION-SAMPLING(X, e, bn, N) returns an estimate of P(X | 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: C, a vector of counts for each value of X, initially zero

  for j = 1 to N do
     x ← PRIOR-SAMPLE(bn)
     if x is consistent with e then
        C[j] ← C[j]+1 where x_j is the value of X in x
  return NORMALIZE(C)
```

Importance Sampling

The general statistical technique of **importance sampling** aims to emulate the effect of sampling from a distribution P[1] using samples from another distribution Q whose samples are easier to obtain.

We ensure that the answers are correct in the limit by applying a correction factor $\frac{P(\boldsymbol{x})}{Q(\boldsymbol{x})}$, also known as a **weight**, to each sample $\boldsymbol{x}$ when counting up the samples.

Our query variable will always be one of the nonevidence variables, Z. The idea of importance sampling is to sample from $P(\boldsymbol{z} \mid \boldsymbol{e})$:

$$\hat{P}(\boldsymbol{z} \mid \boldsymbol{e}) = \frac{N_P(\boldsymbol{z})}{N} \approx P(\boldsymbol{z} \mid \boldsymbol{e})$$

but using an arbitrary distribution $Q(\boldsymbol{z})$:

$$\hat{P}(\boldsymbol{z} \mid \boldsymbol{e}) = \frac{N_Q(\boldsymbol{z})}{N} \frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q(\boldsymbol{z})} \approx Q(\boldsymbol{z}) \frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q(\boldsymbol{z})} = P(\boldsymbol{z} \mid \boldsymbol{e})$$

The question is: which Q to use? We want Q as close as possible to $P(\boldsymbol{z} \mid \boldsymbol{e})$ with $Q(\boldsymbol{z}) \neq 0$ whenever $P(\boldsymbol{z} \mid \boldsymbol{e}) \neq 0$. The most common approach is **likelihood weighting** ...

---

[1] Using P instead of Pr to align with the book so we can distinguish probabilities under different distributions, e.g., P and Q.

Likelihood Weighting

Fix the values for the evidence variables $\boldsymbol{E}$ and sample all the nonevidence variables in topological order, each conditioned on its parents – guarantees each sample consistent with evidence.

Let the sampling distribution be Q_{WS} and nonevidence variables be $\boldsymbol{Z} = \{Z_1, \ldots, Z_l\}$. Then:

$$Q_{WS}(\boldsymbol{z}) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i))$$

Now we need to know the weight for each sample. By general scheme for importance sampling:

$$w(\boldsymbol{z}) = \frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})} = \alpha \frac{P(\boldsymbol{z}, \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})}$$

where the normalizing factor $\alpha = \frac{1}{P(e)}$ is the same for all samples. Since $\boldsymbol{z}$ and $\boldsymbol{e}$ cover all the variables in the Bayes net, $P(\boldsymbol{z}, \boldsymbol{e})$ is the product of the conditional probabilities for the nonevidence variables times the product of the conditional probabilities for the evidence variables:

$$\begin{aligned} w(\boldsymbol{z}) &= \alpha \frac{P(\boldsymbol{z}, \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})} = \alpha \frac{\prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \prod_{i=1}^{m} P(e_i \mid parents(E_i))}{\prod_{i=1}^{l} P(z_i \mid parents(Z_i))} \\ &= \alpha \prod_{i=1}^{m} P(e_i \mid parents(E_i)) \end{aligned} \quad (14.9)$$

The name of this method comes from the fact that probabilities of evidence are generally called likelihoods.

Likelihood Weighting Algorithm

function LIKELIHOOD-WEIGHTING($X, \mathbf{e}, bn, N$) **returns** an estimate of $\mathbf{P}(X \mid \mathbf{e})$
  inputs: X, the query variable
      $\mathbf{e}$, observed values for variables $\mathbf{E}$
      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: $\mathbf{W}$, a vector of weighted counts for each value of X, initially zero

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

function WEIGHTED-SAMPLE($bn, \mathbf{e}$) **returns** an event and a weight

  $w \leftarrow 1$; $\mathbf{x} \leftarrow$ an event with n elements, with values fixed from $\mathbf{e}$
  for i = 1 **to** n **do**
    if X_i is an evidence variable with value x_{ij} in $\mathbf{e}$
      then $w \leftarrow w \times P(X_i = x_{ij} \mid parents(X_i))$
      else $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$, w

Likelihood Weighting Example

Given the query

- $Pr(Rain \mid Cloudy = true, WetGrass = true)$

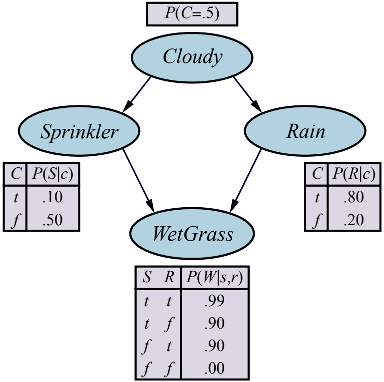
and the ordering

- $Cloudy, Sprinkler, Rain, WetGrass,$

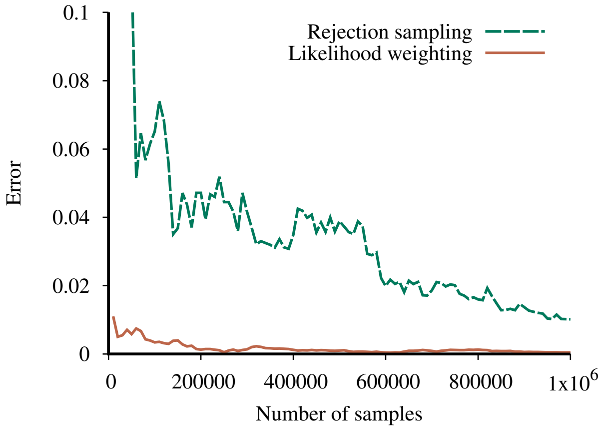
we initialize $w = 1.0$ and proceed as follows:

1. $Cloudy$ is an evidence variable with value true. Therefore, we set
$$w \leftarrow w \cdot Pr(c) = 0.5.$$
2. $Sprinkler$ is not an evidence variable, so sample from $Pr(Sprinkler \mid Cloudy = true) = \langle 0.1, 0.9 \rangle$; suppose this returns false.
3. $Rain$ is not an evidence variable, so sample from $Pr(Rain \mid Cloudy = true) = \langle 0.8, 0.2 \rangle$; suppose this returns true.
4. $WetGrass$ is an evidence variable with value true. Therefore, we set
$$w \leftarrow w \times Pr(WetGrass = true \mid \neg s, r) = 0.5 \cdot 0.9 = 0.45.$$

Here WEIGHTED-SAMPLE returns the event $[true, false, true, true]$ with weight 0.45, which we tally under $Rain = true$.



Rejection vs. Importance Sampling



Markov Chain Monte Carlo (MCMC) Algorithms

Instead of generating each sample from scratch, MCMC algorithms generate a sample by making a random change to the preceding sample.

Think of an MCMC algorithm as being in a particular current state that specifies a value for every variable and generating a next state by making random changes to the current state.

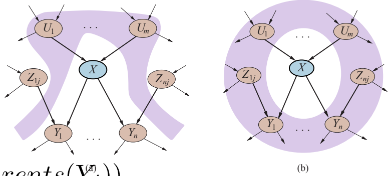
We'll learn two MCMC algorithms:

- Gibbs sampling, and
- Metropolis-Hastings

Gibbs Sampling and Markov Blankets

Gibbs Sampling

- Fix the evidence variables.
- Start in a arbitrary state sampled from the nonevidence variables.
  - Each X_i is sampled from the values in its Markov blanket, $mb(X_i)$



$$P(x_i \mid mb(X_i)) = \alpha P(x_i \mid parents(X_i)) \prod_{Y_j \in Children(X_i)} P(y_i \mid parents(Y_j)) \tag{13.10}$$

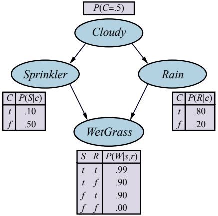
- Wander the state space randomly, keeping the evidence variables fixed and flipping one nonevidence variable by sampling from a distributoin calculated Equation 14.10.

Gibbs sampling for X_i means sampling conditioned on the current values of the variables in its Markov blanket.

Gibbs Sampling Example: Step 1

Consider the query $P(Rain \mid Sprinkler = true, WetGrass = true)$.

- Evidence variables $Sprinkler$ and $WetGrass$ fixed to observed values (true), nonevidence variables $Cloudy$ and $Rain$ initialized randomly to, say, true and false respectively.
  - Initial state is $[true, \mathbf{true}, false, \mathbf{true}]$. Fixed evidence variables marked in bold.



Now the nonevidence variables Z_i are sampled repeatedly in some random order according to a probability distribution $\rho(i)$ for choosing variables. For example:

1. $Cloudy$ is chosen and then sampled, given the current values of its Markov blanket: in this case, we sample from $P(Cloudy|Sprinkler = true, Rain = false)$ whose distribution is calculated using Equation 13.10:

$$P(x_i \mid mb(X_i)) = \alpha P(x_i \mid parents(X_i)) \prod_{Y_j \in Children(X_i)} P(y_i \mid parents(Y_j)) \qquad (13.10)$$

$$P(c \mid s, \neg r) = \alpha P(c) P(s \mid c) P(\neg r \mid c) = \alpha 0.5 \cdot 0.1 \cdot 0.2$$

$$P(\neg c \mid s, \neg r) = \alpha P(\neg c) P(s \mid \neg c) P(\neg r \mid \neg c) = \alpha 0.5 \cdot 0.5 \cdot 0.8$$

yielding $\alpha\langle 0.001, 0.020\rangle \approx \langle 0.048, 0.952\rangle$

- Suppose the result is $Cloudy = false$. Then the new current state is $[false, \mathbf{true}, false, \mathbf{true}]$.

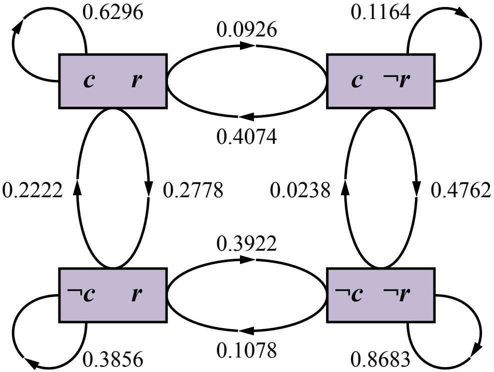
Gibbs Sampling Example: Step 2

2. Sampling $\rho(i)$ then chooses $Rain$ as next variable. We sample from $P(Rain \mid Cloudy = false, Sprinkler = true, WetGrass = true)$ using Equation 13.10 (just like Step 1, not shown here).

- Suppose this yields $Rain = true$. The new current state is $[false, \textbf{true}, true, \textbf{true}]$.

Figure on the right shows the complete Markov chain for the case where variables are chosen uniformly, i.e., $\rho(Cloudy) = \rho(Rain) = 0.5$.

- The algorithm wanders around in this graph, following links with stated probabilities.
- 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 $NORMALIZE(\langle 20, 60 \rangle) = \langle 0.25, 0.75 \rangle$.



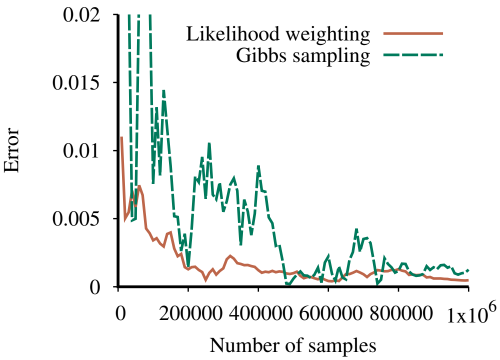
Gibbs Sampling Algorithm

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

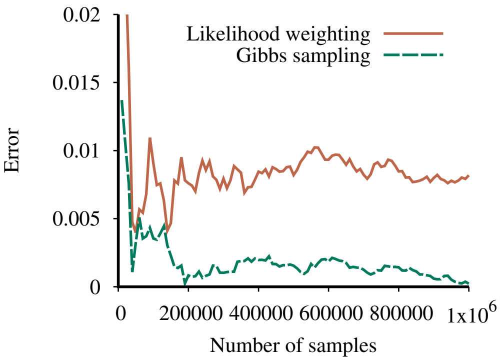
  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Z}$
  for $k = 1$ **to** N **do**
    choose any variable Z_i from $\mathbf{Z}$ according to any distribution $\rho(i)$
    set the value of Z_i in $\mathbf{x}$ by sampling from $\mathbf{P}(Z_i \mid mb(Z_i))$
    $\mathbf{C}[j] \leftarrow \mathbf{C}[j] + 1$ where x_j is the value of X in $\mathbf{x}$
  return NORMALIZE($\mathbf{C}$)

Gibbs Sampling vs. Importance Sampling

Performance of Gibbs sampling compared to likelihood weighting on the car insurance network.



(a)

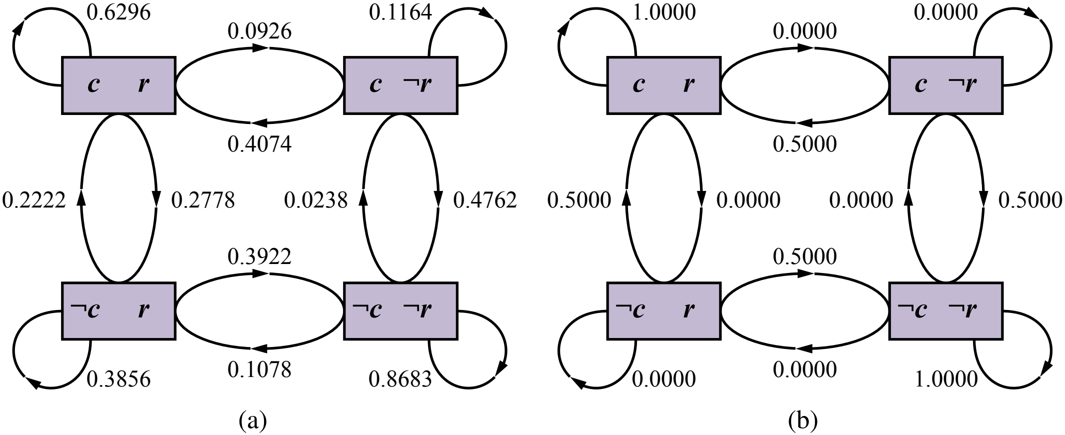


(b)

- (a) for the standard query on $PropertyCost$, and
- (b) for the case where the output variables are observed and Age is the query variable.

Gibbs sampling typically outperforms likelihood weighting when evidence is mostly downstream.

Markov Chains



- (a) The states and transition probabilities of the Markov chain for the query $P(Rain|Sprinkler = true, WetGrass = true)$. Note the self-loops: the state stays the same when either variable is chosen and then resamples the same value it already has.
- (b) The transition probabilities when the CPT for $Rain$ constrains it to have the same value as $Cloudy$.

Metropolis


METROPOLIS
MANUSKRIPT:
THEA VON HARBOU
MUSIK:
GOTTFRIED HUPPERTZ
SCHULZ
EIN FILM VON FRITZ LANG
JN DEN HAUPTROLLEN:
BRIGITTE HELM · GUSTAV FRÖHLICH,
ALFRED ABEL, RUDOLF KLEIN-ROGGE, THEODOR LOOS, FRITZ RASP, HEINRICH GEORGE
AN DER KAMERA: KARL FREUND, GÜNTHER RITTAU
UFA FILM IM VERLEIH DER PARUFAMET

Metropolis-Hastings Sampling

Like simulated annealing, MH has two stages in each iteration of the sampling process:

1. Sample a new state x' from a **proposal distribution** $q(x'|x)$, given the current state x.
2. Probabilistically accept or reject x' according to the **acceptance probability**

$$a(x'|x) = \min\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right)$$

If the proposal is rejected, the state remains at x.

$q(x'|x)$ could be defined as follows:

- With probability 0.95, perform a Gibbs sampling step to generate x'.
- Otherwise, generate x' by running WEIGHTED-SAMPLE using likelihood weighting.

This has the effect of getting MH "unstuck."