14.4 Exact Inference in Bayesian Networks

- Our task is to compute the posterior probability distribution for the query variable $X$, given
  - some assignment of values $e$ to the set of evidence variables $E = E_1, \ldots, E_m$, and
  - the hidden variables are $Y = Y_1, \ldots, Y_l$

- From the full joint probability distribution we can answer the query $P(X | e)$ by computing
  \[
  P(X | e) = \alpha P(X, e) = \alpha \sum_y P(X, e, y)
  \]

- A Bayesian network gives a complete representation of the full joint distribution
- The terms $P(X, e, y)$ can be written as products of conditional probabilities from the network
- Therefore, a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network
14.4.1 Inference by enumeration

- Consider query \( P(\text{Burglary} | \text{JohnCalls} = T, \text{MaryCalls} = T) \)
- The hidden variables are Earthquake and Alarm
  \[ P(\text{Burglary} | \text{Johncalls}, \text{marycalls}) = \alpha \sum_e \sum_a P(\text{Burglary}, e, a, \text{johncalls}, \text{marycalls}) \]
- The semantics of Bayesian networks gives us an expression in terms of CPT entries, e.g.
  \[ P(\text{burglary} | \text{johncalls}, \text{marycalls}) = \alpha \sum_e \sum_a P(\text{burglary}) P(e) P(a | \text{burglary}, e) P(\text{johncalls} | a) P(\text{marycalls} | a) \]
- Rearranging the terms gives a more efficient expression
  \[ P(\text{johncalls} | a) P(\text{marycalls} | a) \]
  \[ \alpha P(\text{burglary}) \sum_e P(e) \sum_a P(a | \text{burglary}, e) \]
Looping over possible values:

\[ P(\text{burglary} \mid \text{johncalls, maycalls}) \] and

\[ P(\neg\text{burglary} \mid \text{johncalls, maycalls}) \]

using the numbers used before yields

\[ P(\text{burglary} \mid \text{johncalls, maycalls}) \approx (0.284, 0.716) \]

- Evaluation of the DAG of a Bayesian net corresponds to the depth-first recursion of a tree, and thus the space complexity is only linear in the number of variables.
- Its time complexity for a network with \( n \) variables is always \( \mathcal{O}(2^n) \).
- E.g., the product \( P(\text{johncalls} \mid a) \, P(\text{maycalls} \mid a) \) needs to be recomputed for each value \( e \).
- By avoiding repeated subexpression re-evaluations helps to avoid wasted computations.

14.4.2 Variable elimination

- Repeated calculation of subexpressions can be avoided by calculating them just once and save the results for later use.
- Let us illustrate the variable elimination algorithm in evaluating the expression \( P(\text{burglary} \mid \text{johncalls, maycalls}) \):

\[
\alpha \frac{P(B)}{f_1(B)} \sum_e \frac{P(e)}{f_2(e)} \sum_a \frac{P(a \mid B, e)}{f_3(A, B, E)} \frac{P(j \mid a)}{f_4(A)} \frac{P(m \mid a)}{f_5(A)}
\]

- The factors of the expression have been associated with names.
- E.g., the factor \( f_5(A) = P(m \mid a) \), does not require summing over \text{MaryCalls}, because the value \text{maycalls} is already fixed.
- Therefore, it only depends on \( A \).
We store the probability, given each value of $a$, in a two-element vector
\[ f_5(A) = \left( \frac{P(m | a)}{P(m | \neg a)} \right) = \left( \frac{0.70}{0.01} \right) \]

Similarly, we store the factor for $f_4(A)$ as the two-element vector $P(j | a)$.

The factor $f_3(A, B, E)$ for $P(a | B, e)$ will be a $2 \times 2 \times 2$ matrix with the “first” element given by $P(a | b, e) = 0.95$ and the “last” by $P(\neg a | \neg b, \neg e) = 0.999$.

In terms of factors, the query is
\[ P(B | j, m) = a f_1(B) \times \sum_a f_2(E) \times \sum_a f_3(A, B, E) \times f_4(A) \times f_5(A) \]

The $\times$ operator is not ordinary matrix multiplication but instead the pointwise product operation.
Computing the pointwise product of a pair of factors

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$f_1(A, B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>.7</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>9</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>$f_2(B, C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

- We must sum out $Alarm$ from the product of factors $f_3, f_4,$ and $f_5$, which will give us a $2 \times 2$ factor $f_6$ whose indices range over just $Burglary$ and $Earthquake$

$$f_6(B, E) = \sum_{a} f_3(A, B, E) \times f_4(a) \times f_5(A) = f_3(a, B, E) \times f_4(a) \times f_5(a) + f_3(-a, B, E) \times f_4(-a) \times f_5(-a)$$
Now we are left with the expression

\[ P(B \mid j, m) = \alpha f_1(B) \times \sum_e f_2(E) \times f_6(B, E) \]

We sum out \textit{Earthquake} in the same way from the product \( f_2(E) \times f_6(B, E) \) which gives the matrix

\[ f_7(\text{Burglary}) = \sum_e f_2(E) \times f_6(B, E) = f_2(e) \times f_6(B, e) + f_2(-e) \times f_6(B, -e) \]

Now we compute the answer to the query \( P(\text{Burglary} \mid \text{john calls, mary calls}) \) by multiplying

\[ \alpha f_1(B) \times f_4(B), \text{ where } f_4(B) = P(\text{Burglary}) \]

In summing out variables from a product of a factors, any factor that does not depend on the variable can be moved outside the summation process.

- Variables that are irrelevant to the query can be removed.
- Query \( P(\text{John Calls} \mid \text{burglary}) \) yields an expression, whose last factor is \( \sum_m \sum_a P(\text{Mary Calls} = m \mid \text{Alarm} = a) \), which is equal to 1 by definition.
- In general, every variable that is not an ancestor of a query variable or evidence variable is irrelevant to the query.
• 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.

• E.g., to sum out $A$ from $f_3(A, B, C)$ we write

$$f(B, C) = \sum_a f_3(A, B, C) = f_3(a, B, C) + f_3(\neg a, B, C)$$

$$= \begin{pmatrix} .06 & .24 \\ .42 & .28 \end{pmatrix} + \begin{pmatrix} .18 & .72 \\ .06 & .04 \end{pmatrix} = \begin{pmatrix} .24 & .96 \\ .48 & .32 \end{pmatrix}$$

14.4.3 The complexity of exact inference

• A Bayesian network is a polytree if there is at most one undirected path between any two nodes in the network.

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

• Inference in Bayesian networks includes inference in propositional logic as a special case.

• Therefore, in general inference in Bayesian networks is NP-hard.

• In fact, it can be shown that the problem is as hard as that of computing the number of satisfying assignments for a propositional logic formula.

• This means that it is strictly harder than NP-complete problems, it is $\#P$-hard.
14.5 Approximate Inference

- Given the intractability of exact inference, it is essential to consider approximate inference methods.
- Approximation is based on random sampling from a known probability distribution (Monte Carlo algorithms).
- E.g., 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.
- Given a source of random numbers in the range $[0, 1]$, it is a simple matter to sample any distribution on a single variable.
14.5.1 Direct Sampling Methods

- From a Bayesian network that has no evidence associated with it, we can sample each variable in turn, in topological order.
- When the values of parent nodes have been drawn, we know from which distribution we have to sample the child.
- Let us fix a topological order for the nodes of our network:
  \( \langle \text{Cloudy}, \text{Sprinkler}, \text{Rain}, \text{WetGrass} \rangle \)
  1. Sample from \( P(\text{Cloudy}) = (0.5, 0.5) \); suppose this returns \( \text{True} \)
  2. Sample from \( P(\text{Sprinkler} | \text{cloudy}) = (0.1, 0.9) \); suppose this returns \( \text{False} \)
  3. Sample from \( P(\text{Rain} | \text{cloudy}) = (0.8, 0.2) \); suppose this returns \( \text{True} \)
  4. Sample from \( P(\text{WetGrass} | \neg \text{sprinkler}, \text{rain}) = (0.9, 0.1) \); suppose this returns \( \text{True} \)

- From the prior joint distribution specified by the network we have drawn the event \( \langle \text{True, False, True, True} \rangle \).
- Let \( S_{PS}(x_1, \ldots, x_n) \) be the probability that a specific event is generated by this prior sampling algorithm.
- Just looking at the sampling process, we have
  \[ S_{PS}(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(X_i)) \]
- On the other hand, this is also the probability of the event according to the Bayesian net’s representation of the joint distribution; i.e.:
  \[ S_{PS}(x_1, \ldots, x_n) = P(x_1, \ldots, x_n) \]
- Let \( N_{PS}(x_1, \ldots, x_n) \) be the frequency of the specific event \( x_1, \ldots, x_n \) and that there are \( N \) total samples.
We expect this frequency to converge in the limit to its expected value according to the sampling probability:

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

E.g., $S_{PS}(\text{True, False, True, True}) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324$, hence in the limit of large $N$, we expect 32.4% of the samples to be of this event.

The estimate of prior sampling is consistent in the sense that the estimated probability becomes exact in the large-sample limit.

One can also produce a consistent estimate of the probability of any partially specified event $x_1, \ldots, x_m$, where $m \leq n$:

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

We denote by $P(\cdot)$ the probability estimated from sample.

---

**Rejection Sampling**

To determine a conditional probability $P(X \mid e)$ we could apply the following simple sampling approach:

1. Generate samples from the prior distribution specified by the network.
2. Reject all those that do not match the evidence $e$.
3. The estimate $\tilde{P}(X = x \mid e)$ is obtained by counting how often $X = x$ occurs in the remaining samples.

The estimated distribution $\tilde{P}(X \mid e)$ that the algorithm returns is, by the definition of the algorithm

$$\alpha N_{ps}(X, e) = N_{ps}(X, e) / N_{ps}(e)$$

As an estimate of the probability of a partially specified event it is consistent

$$\tilde{P}(X \mid e) \approx \frac{P(X, e)}{P(e)} = P(X \mid e)$$
Let us generate 100 samples in order to estimate the distribution $P(Rain | Sprinkler = True)$.
- Suppose that 73 of those that we generate have $Sprinkler = False$ and are rejected.
- The remaining 27 have $Sprinkler = True$.
- Out of them 8 have $Rain = True$ and 19 have $Rain = False$.

Hence, we now have $\hat{P}(Rain \mid sprinkler) \approx (0.296, 0.704)$, while the true distribution is $(0.3, 0.7)$.

As more samples are collected, the estimate will converge to the true answer.

The standard deviation of the error in each probability will be proportional to $1/\sqrt{n}$, where $n$ is the number of samples used in the estimate.

The large number of rejected samples is a big problem:
- The fraction of samples consistent with the evidence drops exponentially as the number of evidence variables grows.

**Likelihood weighting**

- Rejection sampling is inefficient because it ends up rejecting so many of the generated samples.
- To avoid generating needles samples that anyhow get rejected, let us fix the values for the evidence variables $E$ and sample only the remaining variables $X$ and $Y$.
- Not all events are equal, however.
- Each event is weighted by the *likelihood* that the event accords to the evidence.
- The likelihood is measured by the product of the conditional probabilities for each evidence variable, given its parents.
- Intuitively, events in which the actual evidence appears unlikely should be given less weight.
To answer the query $P(Rain \mid sprinkler, wetgrass)$, the weight $w$ is first set to 1.0. Sample from $P(Cloudy) = (0.5, 0.5)$; suppose this returns True. Sprinkler is an evidence variable with value True, therefore we update the weight $w \leftarrow w \times P(sprinkler \mid cloudy) = 0.1$. Sample from $P(Rain \mid cloudy) = (0.8, 0.2)$; suppose this returns True. WetGrass is an evidence variable with value True: $w \leftarrow w \times P(wetgrass \mid sprinkler, rain) = 0.099$. Hence, the algorithm returns the event $[True, True, True, True]$ with weight 0.099 and this is tallied under $Rain = True$.

Let us denote $Z = \{X\} \cup Y$. The weighted sample algorithm samples each variable in $Z$ given its parent values

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

can include both hidden variables and evidence variables.

The sampling distribution $S_{WS}$ pays some attention to the evidence, unlike the prior distribution $P(z)$.

In $S_{WS}$ the sampled values for each $Z_i$ will be influenced by evidence among $Z_i$’s ancestors.

On the other hand, the true posterior distribution $P(z \mid e)$ also takes non-ancestor evidence into account.
The likelihood weight $w$ makes up for the difference between the actual and desired sampling distributions.

Let a given sample $x$ be composed from $z$ and $e$, then
\[ w(z, e) = \prod_{i=1}^{m} P(e_i | \text{parents}(E_i)) \]

The weighted probability of a sample, $S_{WS}(z, e) \cdot w(z, e)$, is
\[
\prod_{i=1}^{l} P(z_i | \text{parents}(Z_i)) \cdot \prod_{i=1}^{m} P(e_i | \text{parents}(E_i)) = P(z, e),
\]

because the two products cover all the variables in the network.

Now it is easy to show that likelihood weighting estimates are consistent:
\[
\hat{P}(x | e) = \alpha \sum_{y} N_{WS}(x, y, e) \cdot w(x, y, e) \quad \text{algorithm}
\]
\[
\approx \alpha' \sum_{y} S_{WS}(x, y, e) \cdot w(x, y, e) \quad \text{for large } N
\]
\[
= \alpha' \sum_{y} P(x, y, e)
\]
\[
= \alpha' P(x, e)
\]
\[
= P(x | e)
\]

Because likelihood weighting uses all the samples generated, it can be much more efficient than rejection sampling.

It will, however, suffer a degradation in performance as the number of evidence variables increases.

Because most samples will have very low weights, the weighted estimate will be dominated by a tiny fraction of samples.
14.5.2 Inference by Markov Chain Simulation

- Markov chain Monte Carlo (MCMC)
- A Monte Carlo algorithm is a randomized algorithm, which can give the false answer with a small probability (cf. Las Vegas)
- MCMC generates each event by making a random change to the preceding event
- The next state is generated by randomly sampling a value for one of the nonevidence variables $X_i$, conditioned on the current values in its Markov blanket
- The Markov blanket of a variable consists of its parents, children, and children’s parents
- MCMC therefore wanders randomly around the state space flipping one variable at a time, but keeping the evidence variables fixed

Gibbs sampling

- Consider the query $P(Rain | sprinkler, wetgrass)$
- Nonevidence variables $Cloudy$ and $Rain$ are initialized randomly to $True$ and $False$, say
- Thus, the initial state is $[True, True, False, True]$
- The nonevidence variables are sampled repeatedly in random order
  - $Cloudy$ is sampled, given the current values of its Markov blanket variables, i.e., $P(Cloudy | sprinkler, \neg Rain)$
  - Suppose the result is $Cloudy = False$. The new state is $[False, True, False, True]$
  - $Rain$ is sampled, given the current values in its Markov blanket, i.e., $P(Rain | \neg cloudy, sprinkler, wetgrass)$
  - Suppose this yields $Rain = True$. The new current state: $[False, True, True, True]$
If the process visits 20 states where *Rain* is *True* and 60 states in which it is *False*, then the answer to the query is 
\[
\text{Normalize}(20,60) = (0.25, 0.75)
\]

Gibbs sampling returns consistent estimates for posterior probabilities.

Through Markov chain analysis one can show that the sampling process settles into a "dynamic equilibrium" in which the long run fraction of time spent in each state is exactly proportional to its posterior probability.

This property follows from the specific transition probability with which the process moves from one state to another, as defined by the conditional distribution given the Markov blanket of the variable being sampled.

16 MAKING SIMPLE DECISIONS

Let us associate each state \( S \) with a numeric *utility* \( U(S) \), which expresses the desirability of the state.

A nondeterministic action \( a \) will have possible outcome states \( \text{Result}(a) = s' \).

Prior to the execution of \( a \) the agent assigns probability
\[
P(\text{Result}(a) = s' \mid a, e)
\]
to each outcome, where \( e \) summarizes the agent’s available evidence of the world.

The *expected utility* of action \( a \) can now be calculated:
\[
EU(a \mid e) = \sum_{s'} P(\text{Result}(a) = s' \mid a, e) \cdot U(s')
\]
The principle of \textit{maximum expected utility} (MEU) says that a rational agent should choose an action that maximizes the agent's expected utility \( \arg \max_a EU(a | e) \)

If we wanted to choose the best sequence of actions using this equation, we would have to enumerate all action sequences, which is clearly infeasible for long sequences.

If the utility function correctly reflects the performance measure by which the behavior is being judged \( \Rightarrow \)

– using MEU the agent will achieve the highest possible performance score averaged over the environments in which it could be placed.

Let us model a nondeterministic action with a \textit{lottery} \( L \), where possible outcomes \( S_1, \ldots, S_n \) can occur with probabilities \( p_1, \ldots, p_n \)

\[
L = [p_1, S_1; p_2, S_2; \ldots; p_n, S_n]
\]

16.2 The Basis of Utility Theory

\( A > B \) Agent prefers lottery \( A \) over \( B \)
\( A \sim B \) The agent is indifferent between \( A \) and \( B \)
\( A \succeq B \) The agent prefers \( A \) to \( B \) or is indifferent between them.

• Deterministic lottery \([1, A] \equiv A\)

• Reasonable constraints on the preference relation (in the name of rationality)
  – Orderability: given any two states, a rational agent must either prefer one to the other or else rate the two as equally preferable
    \( (A > B) \lor (B > A) \lor (A \sim B) \)
  – Transitivity:
    \( (A > B) \land (B > C) \Rightarrow (A > C) \)
Preferences lead to utility

1. Existence of Utility Function:
   If an agent’s preferences follow the axioms of utility, then there exists a real-valued function \( U \) s.t.
   \[
   U(A) > U(B) \iff A \succ B
   \]
   \[
   U(A) = U(B) \iff A \sim B
   \]

2. Expected Utility of a Lottery:
   The utility of a lottery is
   \[
   U([p_1, S_1; \ldots; p_n, S_n]) = \sum_{i=1}^{n} p_i U(S_i)
   \]
   • Because the outcome of a nondeterministic action is a lottery, this gives us the MEU decision rule from slide 110
The axioms do not specify a unique utility function. For example, we can transform a utility function \( U(S) \) into

\[ U'(S) = a \cdot U(S) + b \]

where \( b \) is a constant and \( a \) is any positive constant.

E.g., \( temp^{\circ}F = 1.8 \cdot temp^{\circ}C + 32 \)

Clearly, this affine transformation leaves the agent’s behavior unchanged.

In deterministic contexts, where there are states but no lotteries, behavior is unchanged by any monotonic transformation.

E.g., the cube root of the utility \( \sqrt[3]{U(S)} \)

Utility function is ordinal — it really provides just rankings of states rather than meaningful numerical values.

16.3 Utility Functions

Money (or an agent’s total net assets) would appear to be a straightforward utility measure.

The agent exhibits a monotonic preference for definite amounts of money.

We need to determine a model for lotteries involving money:

- We have won a million euros in a TV game show.
- The host offers to flip a coin, if the coin comes up heads, we end up with nothing, but if it comes up tails, we win three million euros.
- Is the only rational choice to accept the offer which has the expected monetary value of 1.5 million euros?

The true question is maximizing total wealth (not winnings).
Normalized utilities

- The scale of utilities reaches from the best possible prize $u_1$ to the worst possible catastrophe $u_\perp$
- *Normalized utilities* use a scale with $u_\perp = 0$ and $u_1 = 1$
- Utilities of intermediate outcomes are assessed by asking the agent to indicate a preference between the given outcome state $S$ and a standard lottery $[p, u_1; 1-p, u_\perp]$

- The probability $p$ is adjusted until the agent is indifferent between $S$ and the standard lottery
- Assuming normalized utilities, the utility of $S$ is given by $p$