Inference in Bayesian networks

Chapter 14.4

Outline

- Exact inference by enumeration
- Exact inference by variable elimination
- Approximate inference by stochastic simulation
- Approximate inference by Markov chain Monte Carlo

Inference tasks

- Simple queries: compute posterior marginal $P(X_i | E = e)$
  e.g., $P(\text{NoGas}|\text{Gauge = empty, Lights = on, Starts = false})$

- Conjunctive queries: $P(X_i, X_j | E = e) = P(X_i | E = e)P(X_j | X_i, E = e)$

- Optimal decisions: decision networks include utility information; probabilistic inference required for $P(\text{outcome}|\text{action, evidence})$

- Value of information: which evidence to seek next?

- Sensitivity analysis: which probability values are most critical?

- Explanation: why do I need a new starter motor?

Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

Simple query on the burglary network:

$P(B | j, m) = P(B, j, m) / P(j, m) = \alpha \sum_e P(B, e, j, a, j, m)$

Rewrite full joint entries using product of CPT entries:

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

Recursive depth-first enumeration: $O(n)$ space, $O(d^n)$ time

Enumeration algorithm

```python
function ENUMERATION-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 \{X\} \cup E \cup Y
Q(X) ← a distribution over X, initially empty
for each value x_i of X do
    extend e with value x_i for X
    \[Q(x)\] ← ENUMERATE-ALL(VARS|e), e
return Normalize(Q(X))

function ENUMERATE-ALL(vars, e) returns a real number
if Empty?(vars) then return 1.0
Y ← FIRST(vars)
if Y has value y in e then return $P(y | Pa(Y)) \times ENUMERATE-ALL(REST(vars), e)$
else return $\sum_y P(y | Pa(Y)) \times ENUMERATE-ALL(REST(vars), e_y)$
where e_y is e extended with Y = y
```

Evaluation tree

- Slightly intelligent way to sum out variables from the joint without actually constructing its explicit representation

- Simple query on the burglary network:
  $P(B | j, m) = P(B, j, m) / P(j, m) = \alpha \sum_e P(B, e, j, a, j, m)$

- Rewrite full joint entries using product of CPT entries:
  $P(B | j, m) = \alpha \sum_e P(B) P(e) P(a | B, e) P(j | a) P(m | a)$

- Recursive depth-first enumeration: $O(n)$ space, $O(d^n)$ time
Inference by variable elimination

Variable elimination: carry out summations right-to-left, storing intermediate results (factors) to avoid recomputation.

\[ P(B|j, m) = \alpha P(B) \sum_c P(c) \sum_{a|B,c} P(a|B,c) P(j|a) P(m|a) \]
\[ = \alpha P(B) \sum_c P(c) \sum_{a|B,c} f(j|a) f_M(a) \]
\[ = \alpha P(B) \sum_c P(c) \sum_{a|B,c} f(j|a) f_M(a) \]
\[ = \alpha P(B) \sum_c P(c) f_{EAM}(b, c) \text{ (sum out } A) \]
\[ = \alpha P(B) f_{EAM}(b) \text{ (sum out } E) \]
\[ = \alpha f_B(b) \times f_{EAM}(b) \]

Variable elimination: Basic operations

Summing out a variable from a product of factors:
move any constant factors outside the summation
add up submatrices in pointwise product of remaining factors

\[ \sum_c f_1 \cdots f_k = f_1 \times \cdots \times f_k \sum_c f_{i+1} \cdots \times f_k = f_1 \times \cdots \times f_k \]
assumed \( f_1, \ldots, f_i \) do not depend on \( X \)

Pointwise product of factors \( f_i \) and \( f_j \):
\[ f_i(x_1, \ldots, x_j, y_j, \ldots, y_k) \times f_j(y_1, \ldots, y_j, z_1, \ldots, z_l) \]
\[ = f(x_1, \ldots, x_j, y_j, \ldots, y_k, z_1, \ldots, z_l) \]

E.g., \( f_i(a, b) \times f_j(b, c) = f(a, b, c) \)

Variable elimination algorithm

```
function ELIMINATION-ASSIGN(X, e, m) returns a distribution over X
inputs: X, the query variable
        e, evidence specified as an event
        m, a belief network specifying joint distribution \( P(X_1, \ldots, X_n) \)
factors = []
vars = REVERSE(\( \text{VARs}[m] \))
for each var in vars do
    factors = [MAKE-FACTOR(var, e)|factors]
    if var is a hidden variable then factors += SUM-OUT(var, factors)
return NORMALIZE(POINTWISE-PRODUCT(factors))
```

Irrelevant variables

Consider the query \( P(\text{JohnCalls}|\text{Burglary} = \text{true}) \)
\[ P(j) = \alpha P(b) \sum_c P(c) \sum_{a|b,c} P(a|b,c) P(j|a) \frac{P(m|a)}{P(m)} \]
Sum over \( m \) is identically 1; \( M \) is irrelevant to the query.

Thm 1: \( Y \) is irrelevant unless \( Y \in \text{Ancestors}(\{X\} \cup E) \)

Here, \( XX = \text{JohnCalls}, EE = \{ \text{Burglary} \} \), and \( \text{Ancestors}(\{X\} \cup E) = \{ \text{Alarm, Earthquake} \} \)
so \( \text{MaryCalls} \) is irrelevant.

(Compare this to backward chaining from the query in Horn clause KBs)

Irrelevant variables contd.

Defn: moral graph of Bayes net: marry all parents and drop arrows

Defn: \( A \) is \( m \)-separated from \( B \) by \( C \) iff separated by \( C \) in the moral graph

Thm 2: \( Y \) is irrelevant if \( m \)-separated from \( X \) by \( E \)

For \( P(\text{JohnCalls}|\text{Alarm} = \text{true}) \), both \( \text{Burglary} \) and \( \text{Earthquake} \) are irrelevant

Complexity of exact inference

Singly connected networks (or polytrees):
- any two nodes are connected by at most one (undirected) path
- time and space cost of variable elimination are \( O(d^n) \)

Multiply connected networks:
- can reduce 3SAT to exact inference \( \Rightarrow \) NP-hard
- equivalent to counting 3SAT models \( \Rightarrow \) #P-complete

```
1. A v B v C
2. C v D v ~A
3. B v C v ~D
```
Inference by stochastic simulation

Basic idea:
1) Draw $N$ samples from a sampling distribution $S$
2) Compute an approximate posterior probability $\hat{P}$
3) Show this converges to the true probability $P$

Outline:
- Sampling from an empty network
- Rejection sampling: reject samples disagreeing with evidence
- Likelihood weighting: use evidence to weight samples
- Markov chain Monte Carlo (MCMC): sample from a stochastic process whose stationary distribution is the true posterior

Sampling from an empty network

function $\text{Prior-Sample}(b)$ returns an event sampled from $b$
inputs: $b$, a belief network specifying joint distribution $P(X_1, \ldots, X_n)$
$x$ — an event with $n$ elements
for $i = 1$ to $n$ do
  $x_i$ — a random sample from $P(X_i | \text{parents}(X_i))$
given the values of $\text{Parents}(X_i)$ in $x$
return $x$

Example

| $C$ | $P(S|C)$ |
|-----|----------|
| T   | .10      |
| F   | .50      |

| $C$ | $P(R|C)$ |
|-----|----------|
| T   | .80      |
| F   | .20      |

Example

<table>
<thead>
<tr>
<th>$P(C)$</th>
<th>.50</th>
</tr>
</thead>
</table>

Example

<table>
<thead>
<tr>
<th>$P(C)$</th>
<th>.50</th>
</tr>
</thead>
</table>

Example

<table>
<thead>
<tr>
<th>$P(C)$</th>
<th>.50</th>
</tr>
</thead>
</table>

Example

<table>
<thead>
<tr>
<th>$P(C)$</th>
<th>.50</th>
</tr>
</thead>
</table>

Example

<table>
<thead>
<tr>
<th>$P(C)$</th>
<th>.50</th>
</tr>
</thead>
</table>
### Example

|       | C | P(S|C) |
|-------|---|-------|
|       | T | .10   |
|       | F | .50   |

|       | C | P(R|C) |
|-------|---|-------|
|       | T | .80   |
|       | F | .20   |

|       | S | R   | P(W|S,R) |
|-------|---|-----|---------|
|       | T | T   | .99     |
|       | T | F   | .90     |
|       | F | T   | .90     |
|       | F | F   | .01     |

### Sampling from an empty network contd.

Probability that \( \text{PriorSample} \) generates a particular event

\[ S_{PS}(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | \text{parents}(X_i)) = P(x_1, \ldots, x_n) \]

i.e., the true prior probability

E.g., \( S_{PS}(t, f, t, t) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324 = P(t, f, t, t) \)

Let \( N_{PS}(x_1, \ldots, x_n) \) be the number of samples generated for event \( x_1, \ldots, x_n \)

Then we have

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

That is, estimates derived from \( \text{PriorSample} \) are consistent

Shorthand: \( \hat{P}(x_1, \ldots, x_n) \approx P(x_1, \ldots, x_n) \)

### Rejection sampling

\( P(X|e) \) estimated from samples agreeing with \( e \)

**Function** Rejection-Sampling \((X, e, ln, N) \) returns an estimate of \( P(X|e) \)

local variables: \( N \), a vector of counts over \( X \), initially zero

for \( j \) from 1 to \( N \) do

\( x \leftarrow \text{Prior-Sample}(e) \)

if \( x \) is consistent with \( e \) then

\( N[e] \leftarrow N[e]+1 \)

where \( e \) is the value of \( X \) in \( x \)

return Normalize \( [N[e]] \)

E.g., estimate \( P(\text{Rain}|\text{Sprinkler} = \text{true}) \) using 100 samples

27 samples have \( \text{Sprinkler} = \text{true} \)

Of these, 8 have \( \text{Rain} = \text{true} \) and 19 have \( \text{Rain} = \text{false} \).

\( P(\text{Rain}|\text{Sprinkler} = \text{true}) = \text{Normalize}(8, 19) = (0.296, 0.704) \)

Similar to a basic real-world empirical estimation procedure

### Analysis of rejection sampling

\( P(X|e) = a N_{PS}(X|e) \) (algorithm defn.)

\( = \frac{N_{PS}(X|e)}{N_{PS}(e)} \) (normalized by \( N_{PS}(e) \))

\( \approx \frac{P(X|e)}{P(e)} \) (property of \( \text{PriorSample} \))

\( = \text{P}(X|e) \) (defn. of conditional probability)

Hence rejection sampling returns consistent posterior estimates

Problem: hopelessly expensive if \( P(e) \) is small

\( P(e) \) drops off exponentially with number of evidence variables!
Likelihood weighting

Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

function **Likelihood-Weighting**(*X*, e, b, *N*) returns an estimate of \(P(X|e)\)

local variables: \(W\), a vector of weighted counts over \(X\), initially zero

for \(j = 1\) to \(N\) do

\(x, w \leftarrow \text{Weighted-Sample}(b)\)

\(W[x] \leftarrow W[x] + w\) where \(x\) is the value of \(X\) in \(x\)

return \(\text{Normalize}(W[X])\)

function **Weighted-Sample**(*b*, *e*) returns an event and a weight

\(x\) an event with \(n\) elements; \(w\) an integer

for \(i = 1\) to \(n\) do

if \(X_i\) has a value \(x_i\) in \(e\)

then \(w \leftarrow w \times P(X_i = x_i | \text{parents}(X_i))\)

else \(x_i \leftarrow \text{a random sample from } P(X_i | \text{parents}(X_i))\)

return \(x, w\)

Chapter 14.4

Likelihood weighting example

<table>
<thead>
<tr>
<th>PC,</th>
<th>cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PR,</th>
<th>sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.10</td>
</tr>
<tr>
<td>F</td>
<td>.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WC,</th>
<th>wet</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.80</td>
</tr>
<tr>
<td>F</td>
<td>.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S, R, W(S,R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T, T</td>
</tr>
<tr>
<td>T, F</td>
</tr>
<tr>
<td>F, T</td>
</tr>
<tr>
<td>F, F</td>
</tr>
</tbody>
</table>

\(w = 1.0\)

\(w = 1.0 \times 0.1\)

Chapter 14.4

Likelihood weighting example

<table>
<thead>
<tr>
<th>PC,</th>
<th>cloudy</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PR,</th>
<th>sprinkler</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.10</td>
</tr>
<tr>
<td>F</td>
<td>.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WC,</th>
<th>wet</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>.80</td>
</tr>
<tr>
<td>F</td>
<td>.20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S, R, W(S,R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T, T</td>
</tr>
<tr>
<td>T, F</td>
</tr>
<tr>
<td>F, T</td>
</tr>
<tr>
<td>F, F</td>
</tr>
</tbody>
</table>

\(w = 1.0\)

\(w = 1.0 \times 0.1\)
Likelihood weighting example

\[ w = 1.0 \times 0.1 \]

Approximate inference using MCMC

“State” of network = current assignment to all variables.

Generate next state by sampling one variable given Markov blanket

Sample each variable in turn, keeping evidence fixed

\[
\text{function MCMC-Ask}(X, e, l_m, N) \text{ returns an estimate of } P(X|e) \\
\text{local variables: } N[X], \text{ a vector of counts over } X, \text{ initially zero} \\
Z, \text{ the nonevidence variables in } l_m \\
x, \text{ the current state of the network, initially copied from } e \\
\text{initialize } x \text{ with random values for the variables in } Y \\
\text{for } j = 1 \text{ to } N \text{ do} \\
\text{for each } Z_i \text{ in } Z \text{ do} \\
sample the value of } Z_i \text{ in } x \text{ from } P(Z_i | mb(Z_i)) \\
given the values of } MB(Z_i) \text{ in } x \\
N[x] = N[x] + 1 \text{ where } x \text{ is the value of } X \text{ in } x \\
\text{return Normalize}[N[X]]
\]

Can also choose a variable to sample at random each time

The Markov chain

With \(Sprinkler = true, WetGrass = true\), there are four states:

Wander about for a while, average what you see

Likelihood weighting analysis

Sampling probability for \(\text{WeightedSample}\) is

\[
S_{WS}(x, e) = \prod_{i=1}^{n} P(z_i|\text{parents}(Z_i))
\]

Note: pays attention to evidence in ancestors only

\[ \rightarrow \text{ somewhere "in between" prior and posterior distribution} \]

Weight for a given sample \(x, e\) is

\[
w(x, e) = \prod_{i=1}^{m} P(e_i|\text{parents}(E_i))
\]

Weighted sampling probability is

\[
S_{WS}(x, e) w(x, e) = \prod_{i=1}^{n} P(z_i|\text{parents}(Z_i)) \prod_{i=1}^{m} P(e_i|\text{parents}(E_i)) = P(x, e) \text{ (by standard global semantics of network)}
\]

Hence likelihood weighting returns consistent estimates but performance still degrades with many evidence variables because a few samples have nearly all the total weight

MCMC example contd.

Estimate \(P(Rain|Sprinkler = true, WetGrass = true)\)

Sample \(Cloudy\) or \(Rain\) given its Markov blanket, repeat. Count number of times \(Rain\) is true and false in the samples.

E.g., visit 100 states

31 have \(Rain = true\), 69 have \(Rain = false\)

\[
P(Rain|Sprinkler = true, WetGrass = true) = \text{Normalize}(31, 69) = (0.31, 0.69)
\]

Theorem: chain approaches stationary distribution:

long-run fraction of time spent in each state is exactly proportional to its posterior probability
Markov blanket sampling

Markov blanket of Cloudy is Sprinkler and Rain
Markov blanket of Rain is Cloudy, Sprinkler, and WetGrass

Probability given the Markov blanket is calculated as follows:

\[ P(X_i | \text{mb}(X_i)) = \frac{P(X_i \mid \text{parents}(X_i)) \prod_{Z \in \text{Children}(X_i)} P(Z \mid \text{parents}(Z))}{Z} \]

Easily implemented in message-passing parallel systems, brains

Main computational problems:
1) Difficult to tell if convergence has been achieved
2) Can be wasteful if Markov blanket is large:
   \[ P(X_i | \text{mb}(X_i)) \] won’t change much (law of large numbers)

Summary

Exact inference by variable elimination:
- polytime on polytrees, NP-hard on general graphs
- space = time, very sensitive to topology

Approximate inference by LW, MCMC:
- LW does poorly when there is lots of (downstream) evidence
- LW, MCMC generally insensitive to topology
- Convergence can be very slow with probabilities close to 1 or 0
- Can handle arbitrary combinations of discrete and continuous variables