Inference in belief networks

CHAPTER 15.3–4 + NEW

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 $\mathbf{P}(X_i|\mathbf{E}=\mathbf{e})$
e.g., $P(NoGas|Gauge=empty, Lights=on, Starts=false)$

Conjunctive queries: $\mathbf{P}(X_i, X_j|\mathbf{E}=\mathbf{e}) = \mathbf{P}(X_i|\mathbf{E}=\mathbf{e})\mathbf{P}(X_j|X_i, \mathbf{E}=\mathbf{e})$

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

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?

Inference by enumeration

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

Simple query on the burglary network:
$\mathbf{P}(B|J=true, M=true)$
$= \mathbf{P}(B, J=true, M=true)/P(J=true, M=true)$
$= \alpha\mathbf{P}(B, J=true, M=true)$
$= \alpha\Sigma_e\Sigma_a\mathbf{P}(B, e, a, J=true, M=true)$

Rewrite full joint entries using product of CPT entries:
$P(B=true|J=true, M=true)$
$= \alpha\Sigma_e\Sigma_a P(B=true)P(e)P(a|B=true, e)P(J=true|a)P(M=true|a)$
$= \alpha P(B=true)\Sigma_e P(e)\Sigma_a P(a|B=true, e)P(J=true|a)P(M=true|a)$

Enumeration algorithm

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

```
ENUMERATIONASK(X,e,bn) returns a distribution over X
inputs: X, the query variable
        e, evidence specified as an event
        bn, a belief network specifying joint distribution P(X1,...,Xn)

  Q(x) ← a distribution over X
  for each value xi of X do
      extend e with value xi for X
      Q(xi) ← ENUMERATEALL(VARS[bn],e)
  return NORMALIZE(Q(X))
---------------------------------------------------------------
ENUMERATEALL(vars,e) returns a real number
  if EMPTY?(vars) then return 1.0
  else do
      Y ← FIRST(vars)
      if Y has value y in e
          then return P(y | Pa(Y)) × ENUMERATEALL(REST(vars),e)
          else return Σy P(y | Pa(Y)) × ENUMERATEALL(REST(vars),ey)
              where ey is e extended with Y = y
```


Inference by variable elimination

Enumeration is inefficient: repeated computation
e.g., computes $P(J=true|a)P(M=true|a)$ for each value of e

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

$\mathbf{P}(B|J=true, M=true)$
$= \alpha\, \underbrace{\mathbf{P}(B)}_{B}\, \Sigma_e\, \underbrace{P(e)}_{E}\, \Sigma_a\, \underbrace{\mathbf{P}(a|B, e)}_{A}\, \underbrace{P(J=true|a)}_{J}\, \underbrace{P(M=true|a)}_{M}$
$= \alpha\mathbf{P}(B)\Sigma_e P(e)\Sigma_a \mathbf{P}(a|B, e)P(J=true|a)f_M(a)$
$= \alpha\mathbf{P}(B)\Sigma_e P(e)\Sigma_a \mathbf{P}(a|B, e)f_J(a)f_M(a)$
$= \alpha\mathbf{P}(B)\Sigma_e P(e)\Sigma_a f_A(a, b, e)f_J(a)f_M(a)$
$= \alpha\mathbf{P}(B)\Sigma_e P(e)f_{\bar{A}JM}(b, e)$ (sum out A)
$= \alpha\mathbf{P}(B)f_{\bar{E}\bar{A}JM}(b)$ (sum out E)
$= \alpha f_B(b) \times f_{\bar{E}\bar{A}JM}(b)$

Variable elimination: Basic operations

Pointwise product of factors f_1 and f_2:

$$f_1(x_1, \ldots, x_j, y_1, \ldots, y_k) \times f_2(y_1, \ldots, y_k, z_1, \ldots, z_l) \\ = f(x_1, \ldots, x_j, y_1, \ldots, y_k, z_1, \ldots, z_l)$$

E.g., $f_1(a,b) \times f_2(b,c) = f(a,b,c)$

Summing out a variable from a product of factors: move any constant factors outside the summation:

$$\Sigma_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_i \, \Sigma_x \, f_{i+1} \times \cdots \times f_k = f_1 \times \cdots \times f_i \times f_{\bar{X}}$$

assuming $f_1, \ldots, f_i$ do not depend on X

Variable elimination algorithm

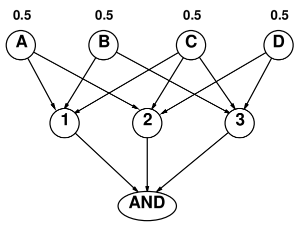
```
function EliminationAsk(X,e,bn) returns a distribution over X
   inputs: X, the query variable
           e, evidence specified as an event
           bn, a belief network specifying joint distribution P(X1,...,Xn)
   if X ∈ e then return observed point distribution for X
   factors ← []; vars ← Reverse(Vars[bn])
   for each var in vars do
      factors ← [MakeFactor(var, e)|factors]
      if var is a hidden variable then factors ← SumOut(var,factors)
   return Normalize(PointwiseProduct(factors))
```


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 $O(d^k n)$

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

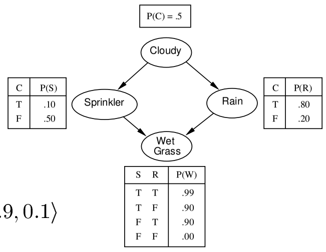
Basic idea:
1) Draw N samples from a sampling distribution S
2) Compute an approximate posterior probability $\hat{P}$
3) Show this converges to the true probability P

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

Sampling from an empty network

```
function PriorSample(bn) returns an event sampled from P(X1,...,Xn) specified by bn
   x ← an event with n elements
   for i = 1 to n do
      xi ← a random sample from P(Xi | Parents(Xi))
   return x
```

$\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$
sample $\rightarrow true$
$\mathbf{P}(Sprinkler|Cloudy) = \langle 0.1, 0.9 \rangle$
sample $\rightarrow false$
$\mathbf{P}(Rain|Cloudy) = \langle 0.8, 0.2 \rangle$
sample $\rightarrow true$
$\mathbf{P}(WetGrass|\neg Sprinkler, Rain) = \langle 0.9, 0.1 \rangle$
sample $\rightarrow true$



Sampling from an empty network contd.

Probability that PriorSample generates a particular event

$$S_{PS}(x_1 \ldots x_n) = \Pi_{i=1}^{n} P(x_i|Parents(X_i)) = P(x_1 \ldots x_n)$$

i.e., the true prior probability

Let $N_{PS}(\mathbf{Y}=\mathbf{y})$ be the number of samples generated for which $\mathbf{Y}=\mathbf{y}$, for any set of variables $\mathbf{Y}$.

Then $\hat{P}(\mathbf{Y}=\mathbf{y}) = N_{PS}(\mathbf{Y}=\mathbf{y})/N$ and

$$\lim_{N\to\infty} \hat{P}(\mathbf{Y}=\mathbf{y}) = \Sigma_{\mathbf{h}} S_{PS}(\mathbf{Y}=\mathbf{y}, \mathbf{H}=\mathbf{h}) \\ = \Sigma_{\mathbf{h}} P(\mathbf{Y}=\mathbf{y}, \mathbf{H}=\mathbf{h}) \\ = P(\mathbf{Y}=\mathbf{y})$$

That is, estimates derived from PriorSample are consistent

Rejection sampling

$\hat{\mathbf{P}}(X|\mathbf{e})$ estimated from samples agreeing with $\mathbf{e}$

```
function REJECTIONSAMPLING(X,e,bn,N) returns an approximation to P(X|e)
   N[X] ← a vector of counts over X, initially zero
   for j = 1 to N do
       x ← PRIORSAMPLE(bn)
       if x is consistent with e then
           N[x] ← N[x]+1 where x is the value of X in x
   return NORMALIZE(N[X])
```

E.g., estimate $\mathbf{P}(Rain|Sprinkler = true)$ using 100 samples
27 samples have $Sprinkler = true$
Of these, 8 have $Rain = true$ and 19 have $Rain = false$.

$\hat{\mathbf{P}}(Rain|Sprinkler = true) = \text{NORMALIZE}(\langle 8, 19\rangle) = \langle 0.296, 0.704\rangle$

Similar to a basic real-world empirical estimation procedure

Analysis of rejection sampling

$\hat{\mathbf{P}}(X|\mathbf{e}) = \alpha \mathbf{N}_{PS}(X, \mathbf{e})$ (algorithm defn.)
$= \mathbf{N}_{PS}(X, \mathbf{e})/N_{PS}(\mathbf{e})$ (normalized by $N_{PS}(\mathbf{e})$)
$\approx \mathbf{P}(X, \mathbf{e})/P(\mathbf{e})$ (property of PRIORSAMPLE)
$= \mathbf{P}(X|\mathbf{e})$ (defn. of conditional probability)

Hence rejection sampling returns consistent posterior estimates

Problem: hopelessly expensive if $P(\mathbf{e})$ is small

Likelihood weighting

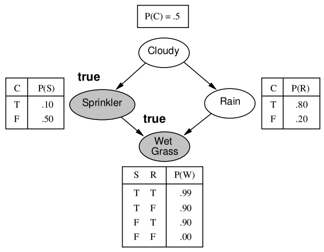
Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

```
function WEIGHTEDSAMPLE(bn,e) returns an event and a weight
   x ← an event with n elements; w ← 1
   for i = 1 to n do
       if X_i has a value x_i in e
           then w ← w × P(X_i = x_i | Parents(X_i))
           else x_i ← a random sample from P(X_i | Parents(X_i))
   return x, w

function LIKELIHOODWEIGHTING(X,e,bn,N) returns an approximation to P(X|e)
   W[X] ← a vector of weighted counts over X, initially zero
   for j = 1 to N do
       x, w ← WEIGHTEDSAMPLE(bn)
       W[x] ← W[x] + w where x is the value of X in x
   return NORMALIZE(W[X])
```


Likelihood weighting example

Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$



LW example contd.

Sample generation process:
1. $w \leftarrow 1.0$
2. Sample $\mathbf{P}(Cloudy) = \langle 0.5, 0.5\rangle$; say $true$
3. $Sprinkler$ has value $true$, so
   $w \leftarrow w \times P(Sprinkler = true|Cloudy = true) = 0.1$
4. Sample $\mathbf{P}(Rain|Cloudy = true) = \langle 0.8, 0.2\rangle$; say $true$
5. $WetGrass$ has value $true$, so
   $w \leftarrow w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099$

Likelihood weighting analysis

Sampling probability for WEIGHTEDSAMPLE is
$S_{WS}(\mathbf{y}, \mathbf{e}) = \Pi_{i=1}^{l} P(y_i|Parents(Y_i))$
Note: pays attention to evidence in *ancestors* only
$\Rightarrow$ somewhere "in between" prior and posterior distribution

Weight for a given sample $\mathbf{y}, \mathbf{e}$ is
$w(\mathbf{y}, \mathbf{e}) = \Pi_{i=1}^{m} P(e_i|Parents(E_i))$

Weighted sampling probability is
$S_{WS}(\mathbf{y}, \mathbf{e}) w(\mathbf{y}, \mathbf{e})$
$= \Pi_{i=1}^{l} P(y_i|Parents(Y_i)) \;\; \Pi_{i=1}^{m} P(e_i|Parents(E_i))$
$= P(\mathbf{y}, \mathbf{e})$ (by standard global semantics of network)

Hence likelihood weighting returns consistent estimates
but performance still degrades with many evidence variables

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

```
function MCMC-Ask(X, e, bn, N) returns an approximation to P(X|e)
   local variables: N[X], a vector of counts over X, initially zero
                    Y, 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 Y
   for j = 1 to N do
       N[x] ← N[x] + 1 where x is the value of X in x
       for each Y_i in Y do
           sample the value of Y_i in x from P(Y_i|MB(Y_i)) given the values of MB(Y_i) in x
   return Normalize(N[X])
```

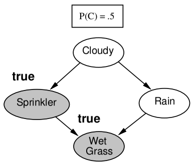
Approaches stationary distribution: long-run fraction of time spent in each state is exactly proportional to its posterior probability

MCMC Example

Estimate $\mathbf{P}(Rain|Sprinkler=true, WetGrass=true)$

Sample $Cloudy$ then $Rain$, repeat.
Count number of times $Rain$ is true and false in the samples.

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



C	P(S)
T	.10
F	.50

C	P(R)
T	.80
F	.20

S	R	P(W)
T	T	.99
T	F	.90
F	T	.90
F	F	.00

MCMC example contd.

Random initial state: $Cloudy=true$ and $Rain=false$

1. $\mathbf{P}(Cloudy|MB(Cloudy)) = \mathbf{P}(Cloudy|Sprinkler, \neg Rain)$
   sample $\rightarrow false$
2. $\mathbf{P}(Rain|MB(Rain)) = \mathbf{P}(Rain|\neg Cloudy, Sprinkler, WetGrass)$
   sample $\rightarrow true$

Visit 100 states
31 have $Rain=true$, 69 have $Rain=false$

$\hat{\mathbf{P}}(Rain|Sprinkler=true, WetGrass=true)$
$= \textsc{Normalize}(\langle 31, 69\rangle) = \langle 0.31, 0.69\rangle$

MCMC analysis: Outline

Transition probability $q(\mathbf{y} \rightarrow \mathbf{y}')$

Occupancy probability $\pi_t(\mathbf{y})$ at time t

Equilibrium condition on π_t defines stationary distribution $\pi(\mathbf{y})$
Note: stationary distribution depends on choice of $q(\mathbf{y} \rightarrow \mathbf{y}')$

Pairwise detailed balance on states guarantees equilibrium

Gibbs sampling transition probability:
sample each variable given current values of all others
$\Rightarrow$ detailed balance with the true posterior

For Bayesian networks, Gibbs sampling reduces to
sampling conditioned on each variable's Markov blanket

Stationary distribution

$\pi_t(\mathbf{y})$ = probability in state $\mathbf{y}$ at time t
$\pi_{t+1}(\mathbf{y}')$ = probability in state $\mathbf{y}'$ at time $t+1$

π_{t+1} in terms of π_t and $q(\mathbf{y} \rightarrow \mathbf{y}')$

$$\pi_{t+1}(\mathbf{y}') = \Sigma_{\mathbf{y}} \pi_t(\mathbf{y}) q(\mathbf{y} \rightarrow \mathbf{y}')$$

Stationary distribution: $\pi_t = \pi_{t+1} = \pi$

$$\pi(\mathbf{y}') = \Sigma_{\mathbf{y}} \pi(\mathbf{y}) q(\mathbf{y} \rightarrow \mathbf{y}') \qquad \text{for all } \mathbf{y}'$$

If π exists, it is unique (specific to $q(\mathbf{y} \rightarrow \mathbf{y}')$)

In equilibrium, expected "outflow" = expected "inflow"

Detailed balance

"Outflow" = "inflow" for each pair of states:

$$\pi(\mathbf{y}) q(\mathbf{y} \rightarrow \mathbf{y}') = \pi(\mathbf{y}') q(\mathbf{y}' \rightarrow \mathbf{y}) \qquad \text{for all } \mathbf{y},\ \mathbf{y}'$$

Detailed balance $\Rightarrow$ stationarity:

$$\begin{aligned}\Sigma_{\mathbf{y}} \pi(\mathbf{y}) q(\mathbf{y} \rightarrow \mathbf{y}') &= \Sigma_{\mathbf{y}} \pi(\mathbf{y}') q(\mathbf{y}' \rightarrow \mathbf{y}) \\ &= \pi(\mathbf{y}') \Sigma_{\mathbf{y}} q(\mathbf{y}' \rightarrow \mathbf{y}) \\ &= \pi(\mathbf{y}')\end{aligned}$$

MCMC algorithms typically constructed by designing a transition probability q that is in detailed balance with desired π

Gibbs sampling

Sample each variable in turn, given *all other variables*

Sampling Y_i, let $\bar{\mathbf{Y}}_i$ be all other nonevidence variables
Current values are y_i and $\bar{\mathbf{y}}_i$; $\mathbf{e}$ is fixed
Transition probability is given by

$$q(\mathbf{y} \to \mathbf{y}') = q(y_i, \bar{\mathbf{y}}_i \to y_i', \bar{\mathbf{y}}_i) = P(y_i'|\bar{\mathbf{y}}_i, \mathbf{e})$$

This gives detailed balance with true posterior $P(\mathbf{y}|\mathbf{e})$:

$$\begin{aligned} \pi(\mathbf{y})q(\mathbf{y} \to \mathbf{y}') &= P(\mathbf{y}|\mathbf{e})P(y_i'|\bar{\mathbf{y}}_i, \mathbf{e}) = P(y_i, \bar{\mathbf{y}}_i|\mathbf{e})P(y_i'|\bar{\mathbf{y}}_i, \mathbf{e}) \\ &= P(y_i|\bar{\mathbf{y}}_i, \mathbf{e})P(\bar{\mathbf{y}}_i|\mathbf{e})P(y_i'|\bar{\mathbf{y}}_i, \mathbf{e}) \quad \text{(chain rule)} \\ &= P(y_i|\bar{\mathbf{y}}_i, \mathbf{e})P(y_i', \bar{\mathbf{y}}_i|\mathbf{e}) \quad \text{(chain rule backwards)} \\ &= q(\mathbf{y}' \to \mathbf{y})\pi(\mathbf{y}') = \pi(\mathbf{y}')q(\mathbf{y}' \to \mathbf{y}) \end{aligned}$$

Markov blanket sampling

A variable is independent of all others given its Markov blanket:
$P(y_i'|\bar{\mathbf{y}}_i, \mathbf{e}) = P(y_i'|MB(Y_i))$

Probability given the Markov blanket is calculated as follows:
$P(y_i'|MB(Y_i)) = P(y_i'|Parents(Y_i))\prod_{Z_j \in Children(Y_i)} P(z_j|Parents(Z_j))$

Hence computing the sampling distribution over Y_i for each flip requires just cd multiplications if Y_i has c children and d values; can cache it if c not too large.

Main computational problems:
1) Difficult to tell if convergence has been achieved
2) Can be wasteful if Markov blanket is large:
$P(Y_i|MB(Y_i))$ won't change much (law of large numbers)

Performance of approximation algorithms

Absolute approximation: $|P(X|\mathbf{e}) - \hat{P}(X|\mathbf{e})| \leq \epsilon$

Relative approximation: $\frac{|P(X|\mathbf{e}) - \hat{P}(X|\mathbf{e})|}{P(X|\mathbf{e})} \leq \epsilon$

Relative $\Rightarrow$ absolute since $0 \leq P \leq 1$ (may be $O(2^{-n})$)

Randomized algorithms may fail with probability at most δ

Polytime approximation: poly$(n, \epsilon^{-1}, \log \delta^{-1})$

Theorem (Dagum and Luby, 1993): both absolute and relative approximation for either deterministic or randomized algorithms are NP-hard for any $\epsilon, \delta < 0.5$

(Absolute approximation polytime with no evidence—Chernoff bounds)