Bayesian inference

- ♢ 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?

Enumeration algorithm

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

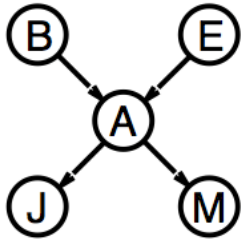
```
ENUMERATIONASK(X,e,bn) returns a distribution over X
inputs: X, the query variable
        e, evidence specified as an event
        bn, a belief network specifying joint distribution P(X_1, ..., X_n)

  Q(x) ← a distribution over X
  for each value x_i of X do
      extend e with value x_i for X
      Q(x_i) ← 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 Σ_y P(y | Pa(Y)) × ENUMERATEALL(REST(vars),e_y)
               where e_y is e extended with Y = y
```

Enumeration algorithm example

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

Simple query on the burglary network:

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

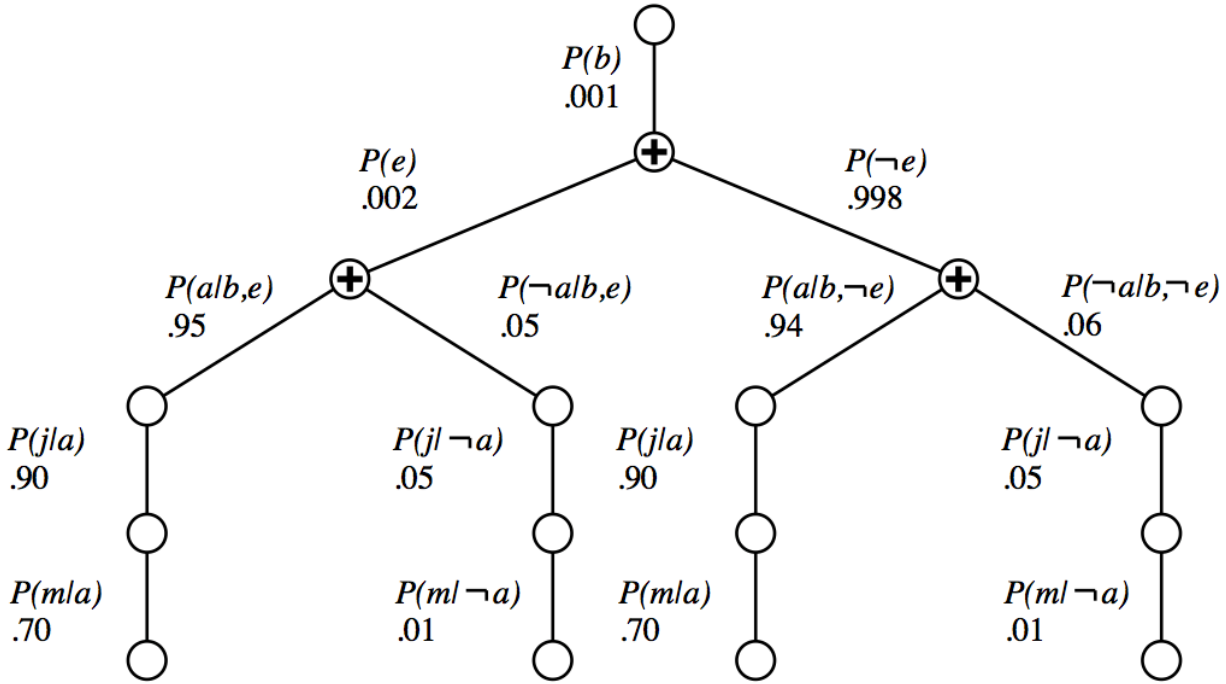


Rewrite full joint entries using product of CPT entries:

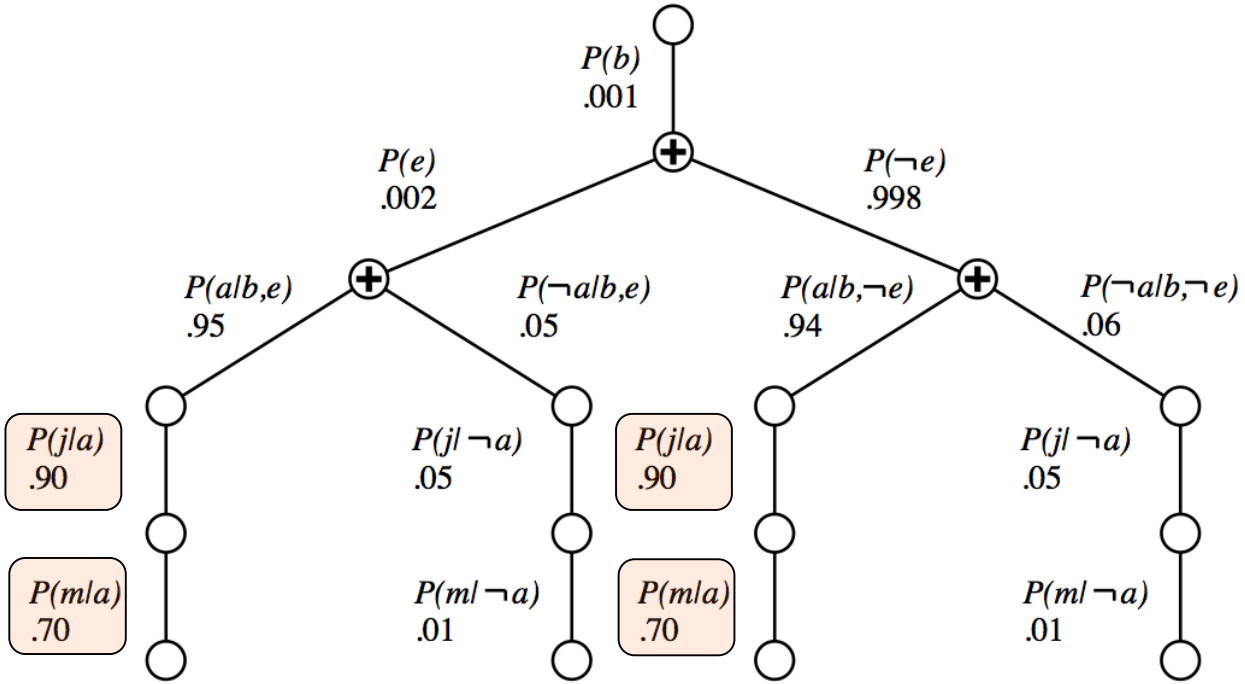
$$\mathbf{P}(B|j,m)$$
$$= \alpha \sum_e \sum_a \mathbf{P}(B)P(e)\mathbf{P}(a|B,e)P(j|a)P(m|a)$$
$$= \alpha\mathbf{P}(B) \sum_e P(e) \sum_a \mathbf{P}(a|B,e)P(j|a)P(m|a)$$

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

Enumeration algorithm example



Enumeration algorithm example



Enumeration is inefficient: repeated computation

e.g., computes $P(j|a)P(m|a)$ for each value of e

Variable elimination algorithm

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

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

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:

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

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

Pointwise Product

- Pointwise multiplication of factors when variable is summed out or at last step
- 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. $f_1(a, b) \times f_2(b, c) = f(a, b, c)$:

a	b	$f_1(a, b)$
T	T	.3
T	F	.7
F	T	.9
F	F	.1

b	c	$f_2(b, c)$
T	T	.2
T	F	.8
F	T	.6
F	F	.4

a	b	c	$f(a, b, c)$
T	T	T	.3 * .2
T	T	F	.3 * .8
T	F	T	.7 * .6
T	F	F	.7 * .4
F	T	T	.9 * .2
F	T	F	.9 * .8
F	F	T	.1 * .6
F	F	F	.1 * .4

Summing Out

- Summing out a variable from a product of factors
  - Move any constant factors outside the summation
  - Add up sub-matrices in pointwise product of remaining factors

$$\sum_x f_1 \times \cdots \times f_k = f_1 \times \cdots \times f_l \times \left(\sum_x f_{l+1} \times \cdots \times f_k \right) = f_1 \times \cdots \times f_l \times f_{\overline{X}}$$

- E.g. $\sum_a f(a, b, c) = f_{\overline{a}}(b, c)$:

a	b	c	$f(a, b, c)$	b	c	$f_{\overline{a}}(b, c)$
T	T	T	.3 * .2	T	T	.3 * .2 + .9 * .2
T	T	F	.3 * .8	T	F	.3 * .8 + .9 * .8
T	F	T	.7 * .6	F	T	.7 * .6 + .1 * .6
T	F	F	.7 * .4	F	F	.7 * .4 + .1 * .4
F	T	T	.9 * .2			
F	T	F	.9 * .8			
F	F	T	.1 * .6			
F	F	F	.1 * .4			

Variable elimination algorithm

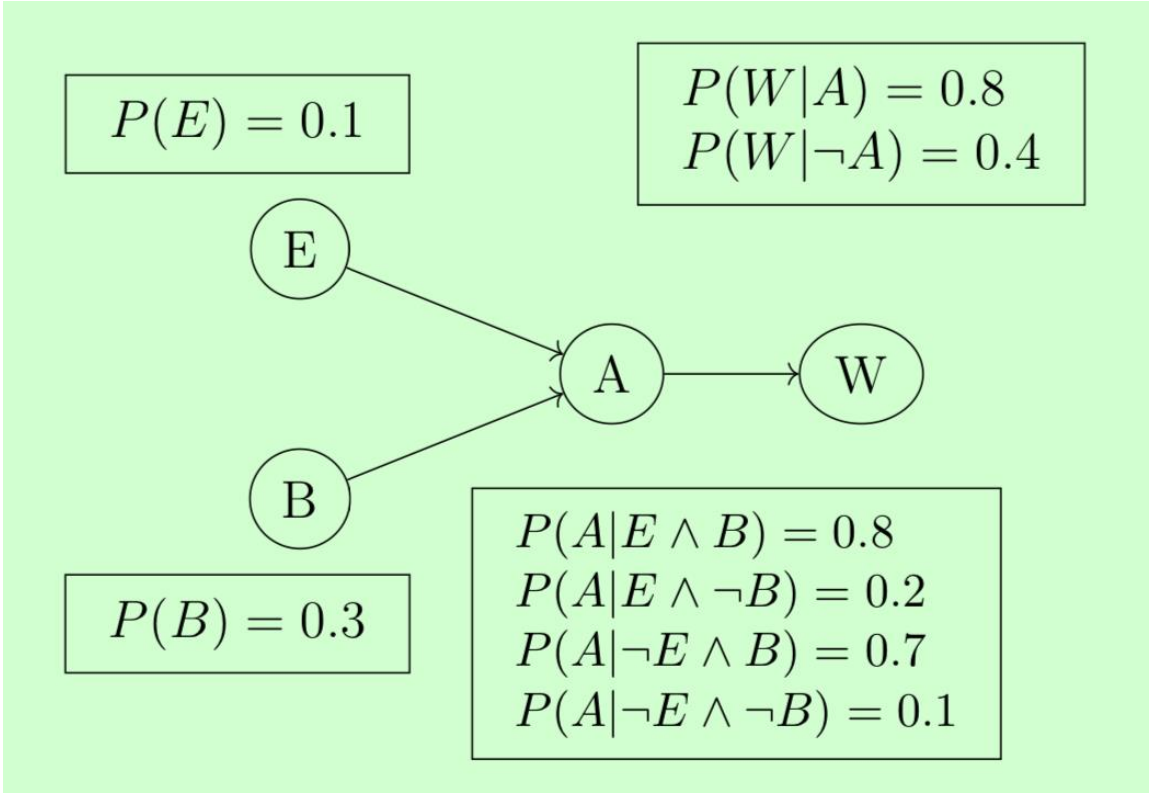
function ELIMINATIONASK(X,**e**,bn) **returns** a distribution over X
  inputs: X, the query variable
    e, evidence specified as an event
    bn, a belief network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  if $X \in$ **e** **then return** observed point distribution for X
  $factors \leftarrow [\,]$; $vars \leftarrow$ REVERSE(VARS[bn])
  for each var **in** $vars$ **do**
    $factors \leftarrow [\text{MAKEFACTOR}(var, \mathbf{e}) | factors]$
    if var is a hidden variable **then** $factors \leftarrow$ SUMOUT(var,$factors$)
  return NORMALIZE(POINTWISEPRODUCT($factors$))

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

Calculate $P(B|\neg A)$

Query variable: B
Evidence variable: A
Hidden variables: E, W



Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

$$P(B \mid \neg a) = \frac{P(B \wedge \neg a)}{P(b \wedge \neg a) + P(\neg b \wedge \neg a)}$$

Using the Bayesian network and summing out hidden variables E and W:

$$P(B \wedge \neg a)$$
$$= \sum_e \sum_w P(B)P(e)P(\neg a|B \wedge e)P(w|\neg a)$$

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

Move summations to the right and restrict them to the summed-over terms:

$$P(B \wedge \neg a)$$

$$= P(B) \left(\sum_e P(e) P(\neg a | B \wedge e) \right) \left(\sum_w P(w | \neg a) \right)$$

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

(1) Define factors.

We will define one factor for each node in the Bayesian network.

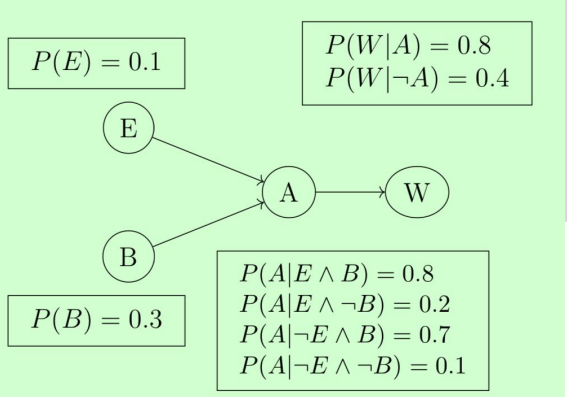
$P(B)$	$P(E)$	$P(A\|B \wedge E)$	$P(W\|A)$
$f_1(B)$	$f_2(E)$	$f_3(A,B,E)$	$f_4(W,A)$

$f_1(B)$:

B	val
t	0.3
f	0.7

$f_2(E)$:

E	val
t	0.1
f	0.9



$f_3(A,B,E)$:

A	B	E	val
t	t	t	0.8
t	t	f	0.7
t	f	t	0.2
t	f	f	0.1
f	t	t	0.2
f	t	f	0.3
f	f	t	0.8
f	f	f	0.9

$f_4(W,A)$:

W	A	val
t	t	0.8
t	f	0.4
f	t	0.2
f	f	0.6

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

(2) Restrict factors.

This step assigns the observed values to the evidence variables.

For each evidence variable, find all the factors that contain the variable. For each such factor, restrict the factor such that the evidence variable takes its observed value.

- Restrict $f_3(A, B, E)$ to $\neg a$ to produce $f_5(B, E)$.

  (Intuitively, this step is converting $P(A|B \wedge E)$ to $P(\neg a|B \wedge E)$.)

$f_5(B, E)$:

B	E	val
t	t	0.2
t	f	0.3
f	t	0.8
f	f	0.9

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

- Restrict $f_4(W, A)$ to $\neg a$ to produce $f_6(W)$.

  (Intuitively, this step is converting $P(W|A)$ to $P(W|\neg a)$.)

$f_6(W)$:

W	val
t	0.4
f	0.6

  Whenever we perform any operation, discard all the factors that we used and create a new factor to store the result. We will use each factor exactly once. After the restrict operations, we have a new list of factors containing f1, f2, f5 and f6.

New factor list: $f_1(B)$, $f_2(E)$, $f_5(B, E)$, $f_6(W)$.

Example

(3) Eliminate the hidden variables since they do not appear in the probability that we want to calculate.

First, we need to choose an order of the hidden variables. For this example, let's eliminate W first and then E.

For each hidden variable, we will do two things. First, find all the factors containing the variable and multiply them together. Second, sum out the hidden variable from the product.

i. In this step, we will eliminate the hidden variable W.

(Intuitively, this step is calculating $\sum_w P(w|\neg a)$.)

Multiply all the factors containing W: $f_6(W)$. Only one factor contains W, so we do not have to multiply any factors.

Sum out W from $f_6(W)$ to get $f_7()$. Note that, W was the only variable in f6, so f7 has no variable and contains a number.

$f_7()$:

val
1.0

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

ii. In this step, we will eliminate the hidden variable E.

(Intuitively, this step is calculating $\sum_e P(e)P(\neg a|B \wedge e)$.)

Multiply all the factors containing E: $f_2(E) \times f_5(B, E) = f_8(B, E)$.

$f_8(B, E)$:

B	E	val
t	t	0.02
t	f	0.27
f	t	0.08
f	f	0.81

Sum out E from $f_8(B, E)$ to get $f_9(B)$.

$f_9(B)$:

B	val
t	0.29
f	0.89

New factor list: $f_1(B)$, $f_7()$, $f_9(B)$.

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

At this point, each remaining factor should contain at most the query variables. Whether we need to perform step 4 depends on the number of factors remaining.

If there are multiple factors left, we need to multiply all the factors together. For our example, we multiply f1, f7, and f9 together to produce a new factor f10. If there is only one factor left, we can skip step 5 and continue to the normalization step.

(4) Multiply all remaining factors.

$$f_1(B) \times f_7() \times f_9(B) = f_{10}(B).$$

$f_{10}(B)$:

B	val
t	0.087
f	0.623

New factor list: $f_{10}(B)$.

Example (https://cs.uwaterloo.ca/~a23gao/cs486686_f21/lecture_notes/Lecture_13_on_Variable_Elimination_Algorithm.pdf)

(5) Normalize $f_{10}(B)$ to get $f_{11}(B)$.

We will normalize the final factor so that it represents a probability distribution. Normalizing the factor produces a new factor f11. The final factor is a distribution containing two probabilities $P(b|\neg a)$ and $P(\neg b|\neg a)$.

$f_{11}(B)$:

B	val
t	0.123
f	0.877

The final factor is $f_{11}(B)$. Thus, $P(b|\neg a) = 0.123$ and $P(\neg b|\neg a) = 0.877$.

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 $\mathbf{P}(X_1, \ldots, X_n)$

  if $X \in$ **e** **then return** observed point distribution for X
  $factors \leftarrow [\,]$; $vars \leftarrow$ REVERSE(VARS[bn])
  for each var **in** $vars$ **do**
    $factors \leftarrow [\text{MAKEFACTOR}(var, \mathbf{e}) | factors]$
    if var is a hidden variable **then** $factors \leftarrow$ SUMOUT(var,$factors$)
  return NORMALIZE(POINTWISEPRODUCT($factors$))

Complexity of exact inference

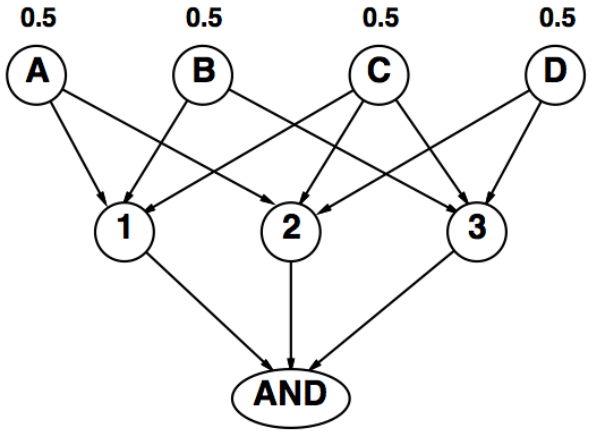
Singly connected networks (or polytrees):

- 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

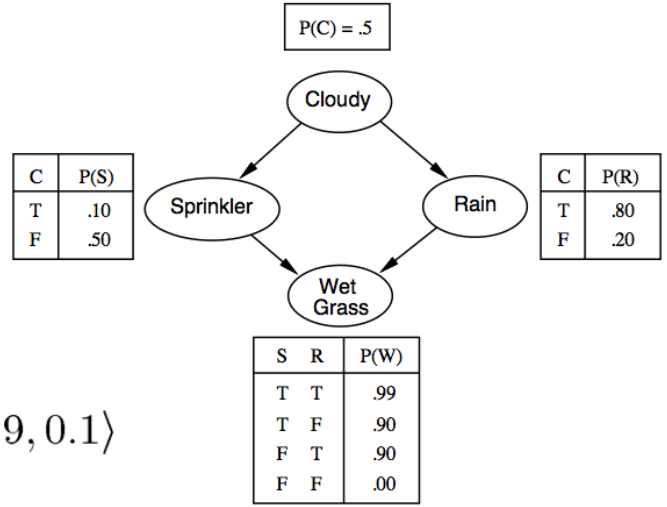
Outline:

- 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 $\mathbf{P}(X_1, \ldots, X_n)$ specified by bn
  $\mathbf{x} \leftarrow$ an event with n elements
  for $i = 1$ **to** n **do**
    $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid Parents(X_i))$
  return $\mathbf{x}$

$\mathbf{P}(Cloudy) = \langle 0.5, 0.5 \rangle$
  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

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 $\mathbf{Y}$.

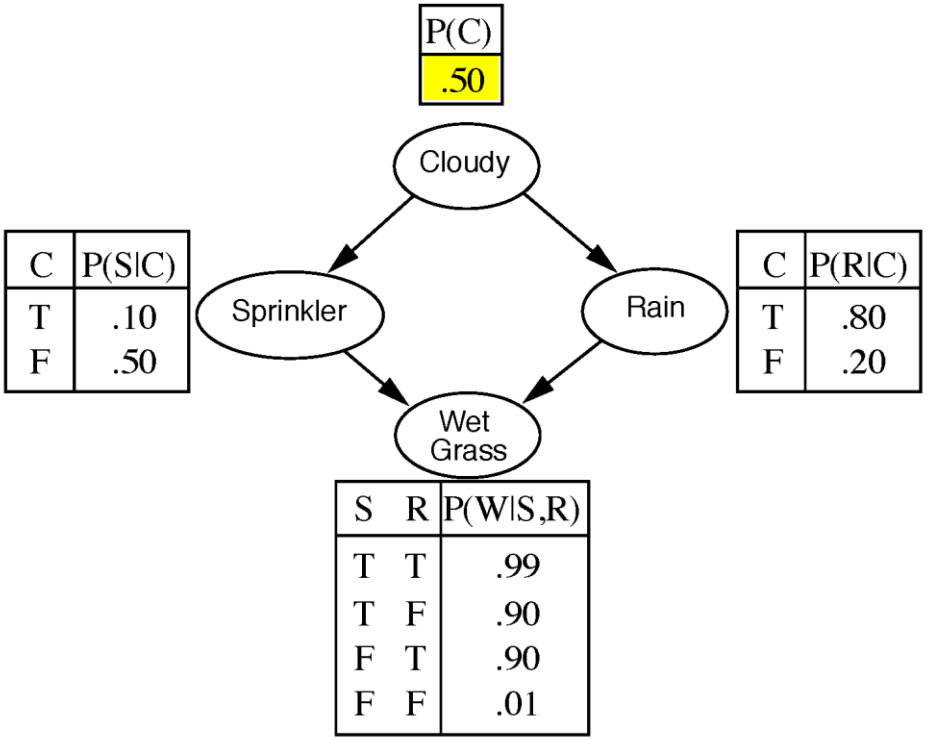
Then $\hat{P}(\mathbf{Y} = \mathbf{y}) = N_{PS}(\mathbf{Y} = \mathbf{y})/N$ and

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

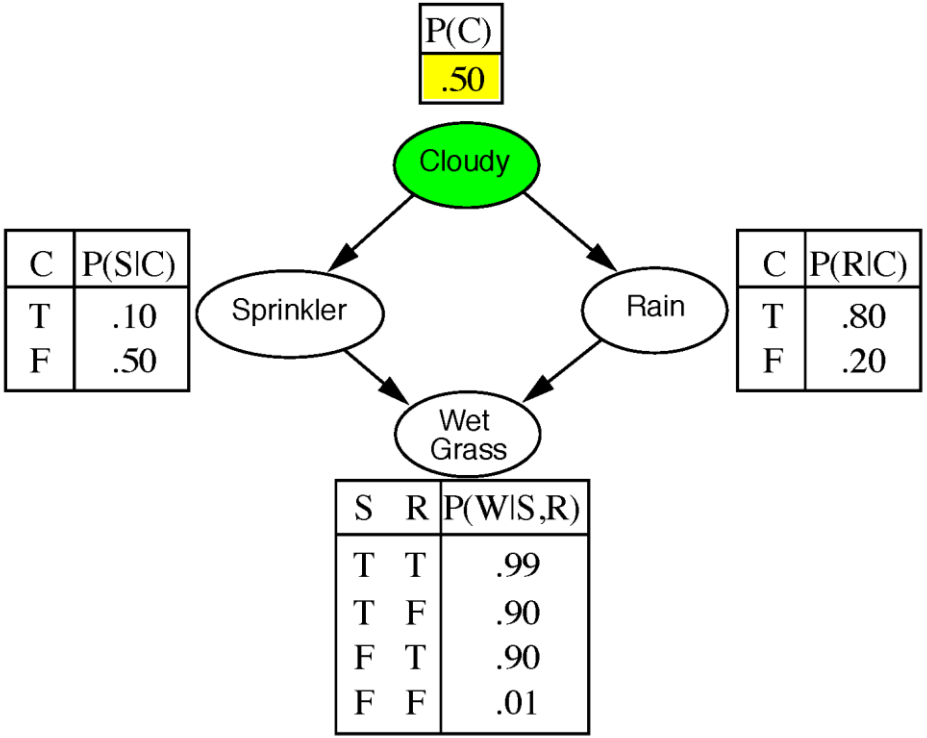
That is, estimates derived from PRIORSAMPLE are consistent

http://courses.csail.mit.edu/6.034s/handouts/spring12/chapter14_mod_b.pdf

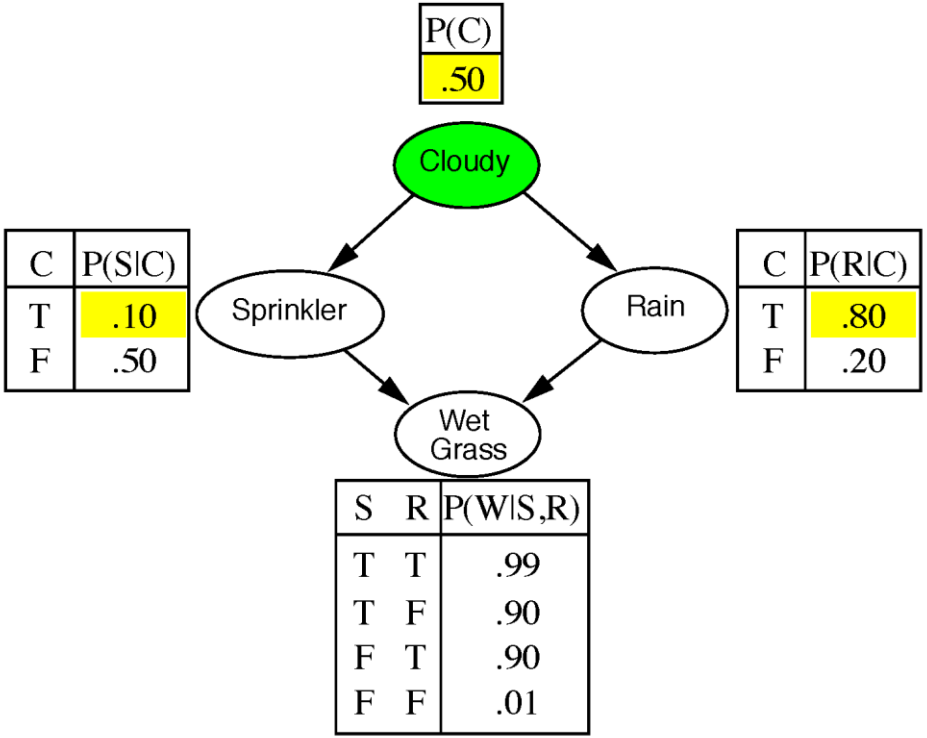
Example



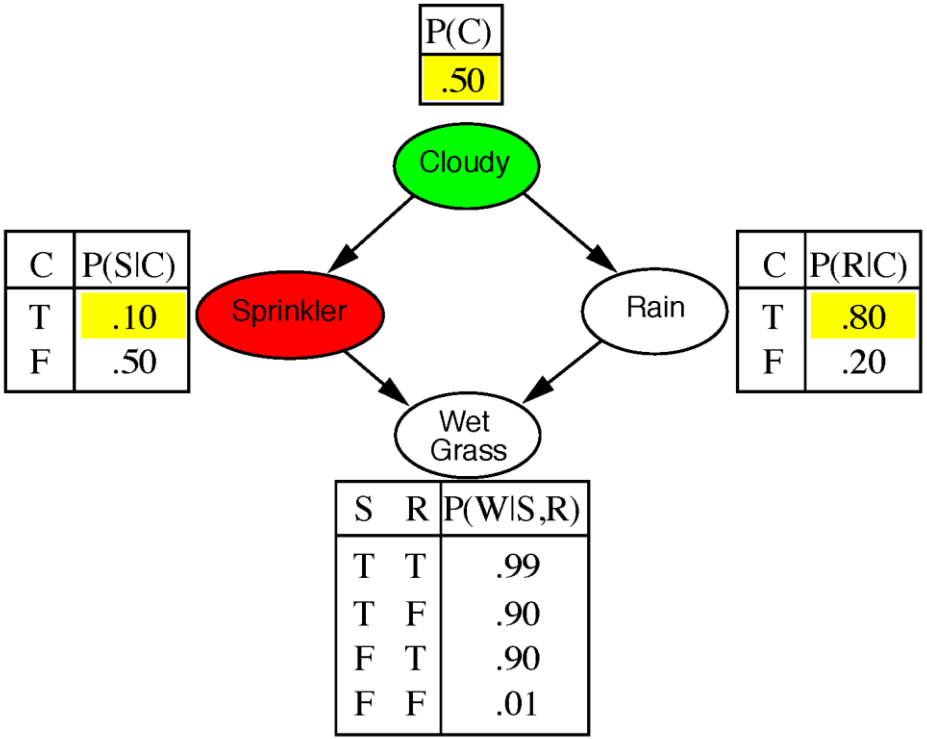
Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

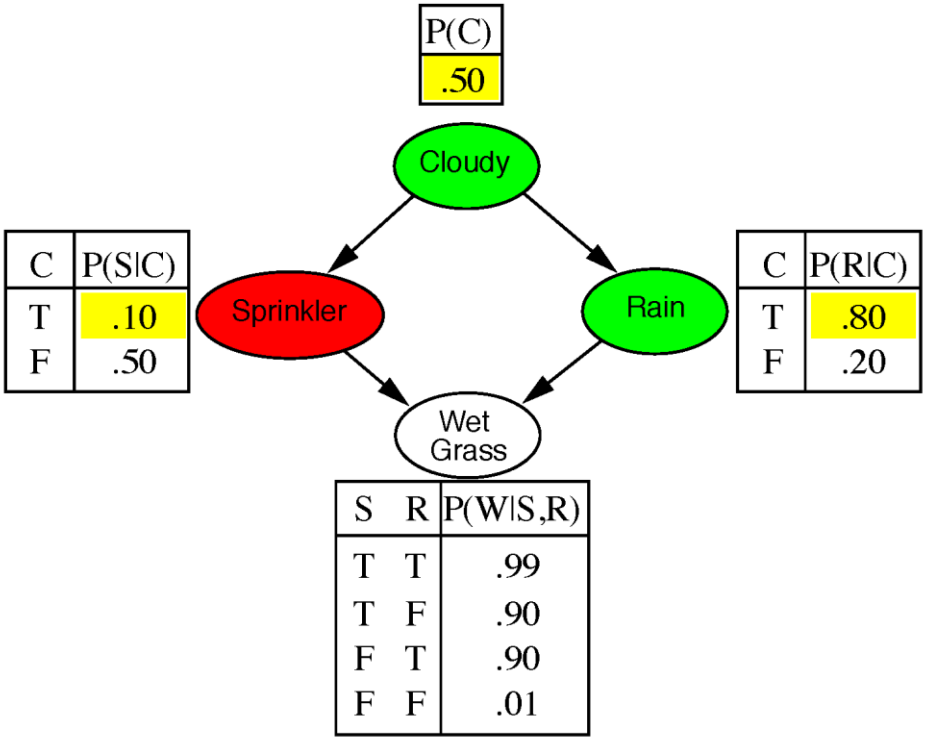
Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

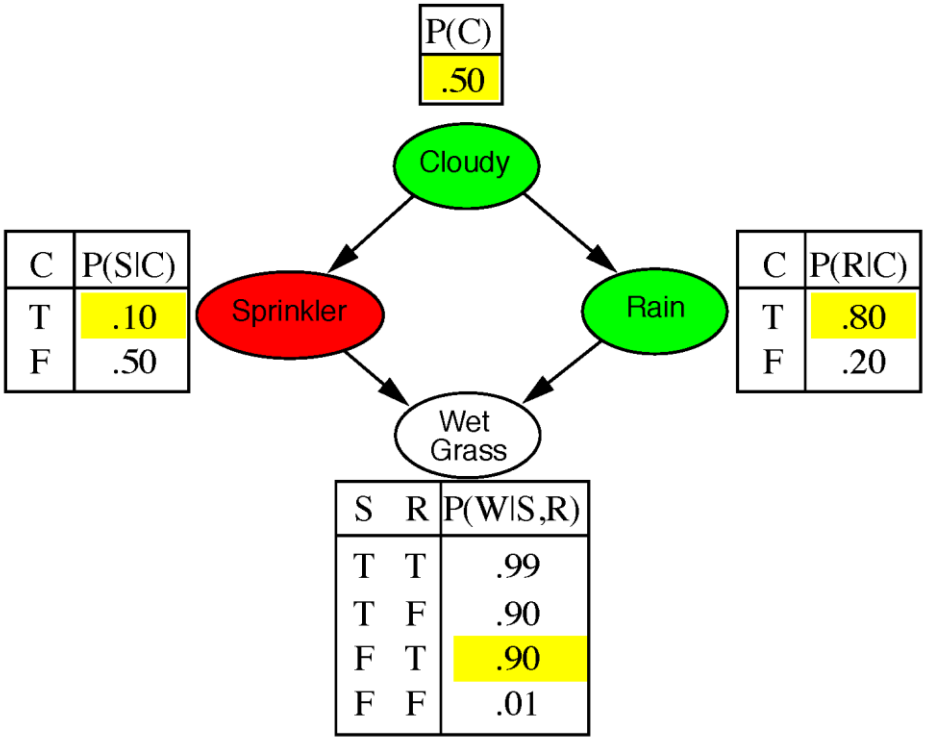
Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

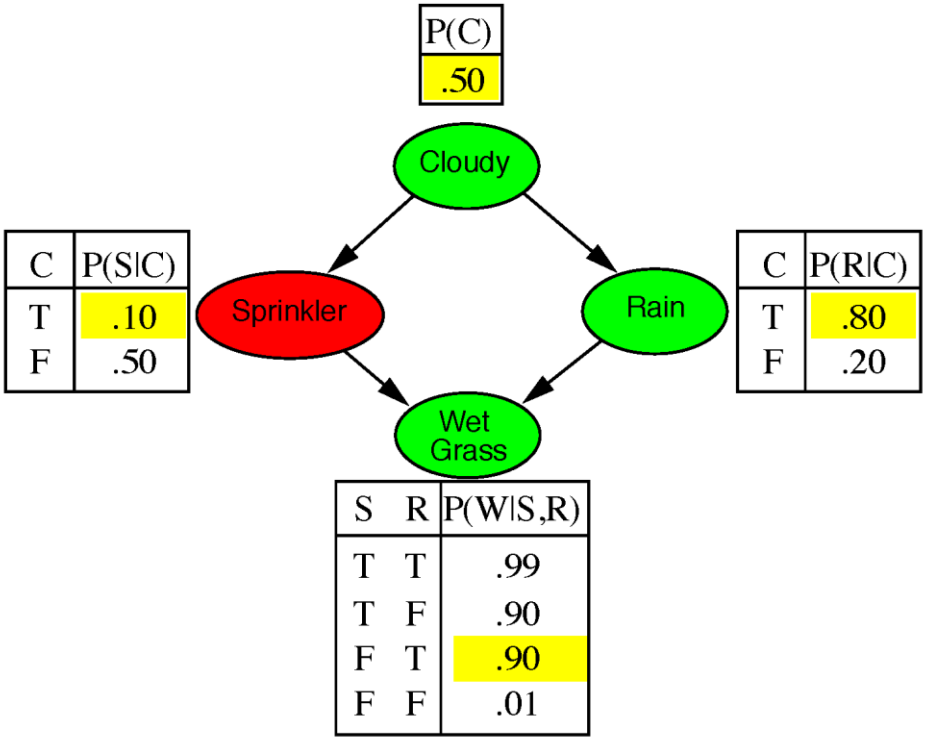
Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

Example


P(C)
.50
Cloudy
C P(S|C)
T .10
F .50
Sprinkler
Rain
C P(R|C)
T .80
F .20
Wet Grass
S R P(W|S,R)
T T .99
T F .90
F T .90
F F .01

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

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

Hence rejection sampling returns consistent posterior estimates

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

Likelihood weighting

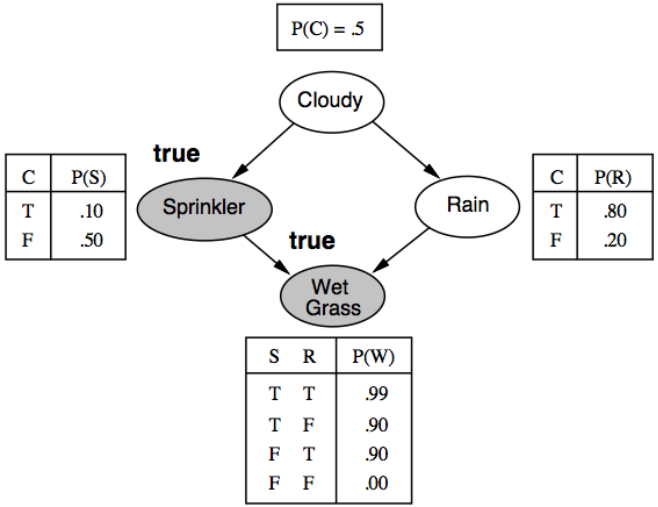
Idea: fix evidence variables, sample only nonevidence variables, and weight each sample by the likelihood it accords the evidence

function WEIGHTEDSAMPLE(bn,**e**) **returns** an event and a weight
  x $\leftarrow$ an event with n elements; $w \leftarrow 1$
  for $i = 1$ **to** n **do**
    if X_i has a value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid Parents(X_i))$
      else $x_i \leftarrow$ a random sample from $\mathbf{P}(X_i \mid Parents(X_i))$
  return **x**, w

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

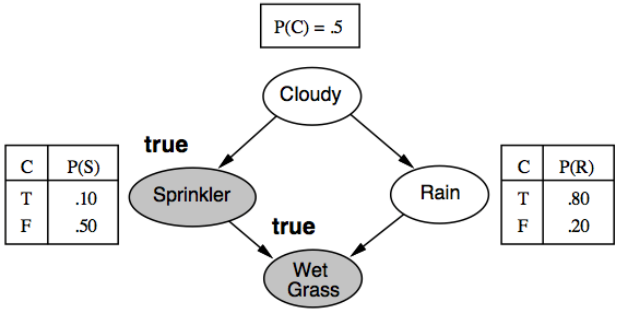
Likelihood weighting example

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



Likelihood weighting example

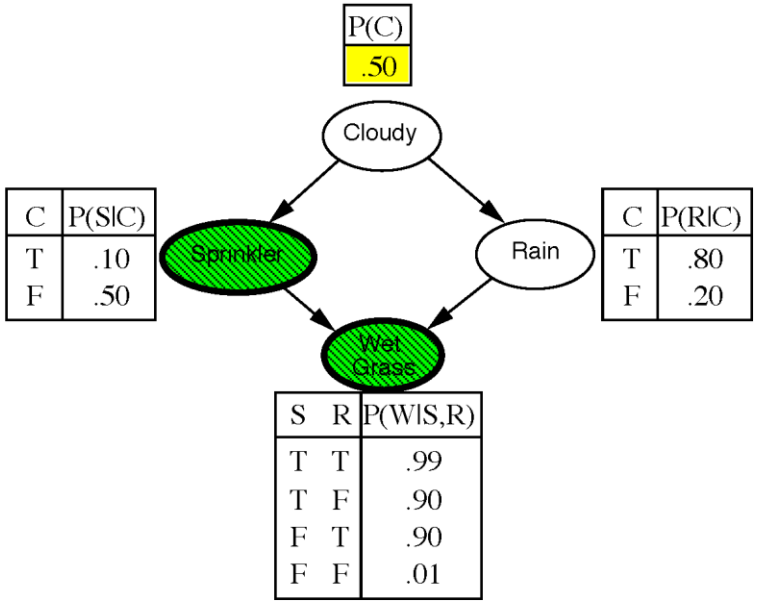
Estimate $\mathbf{P}(Rain|Sprinkler = true, WetGrass = true)$



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
   $w \leftarrow w \times P(WetGrass = true|Sprinkler = true, Rain = true) = 0.099$

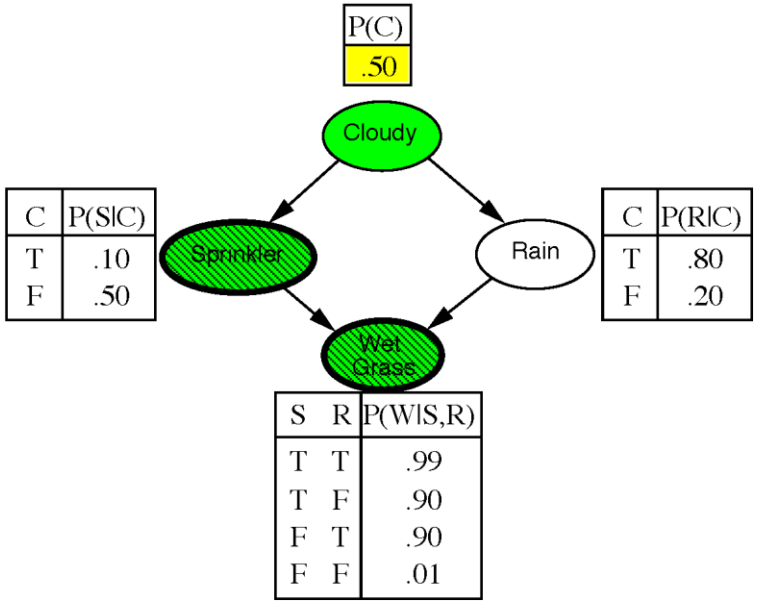
Likelihood weighting example



$w = 1.0$

Likelihood weighting example

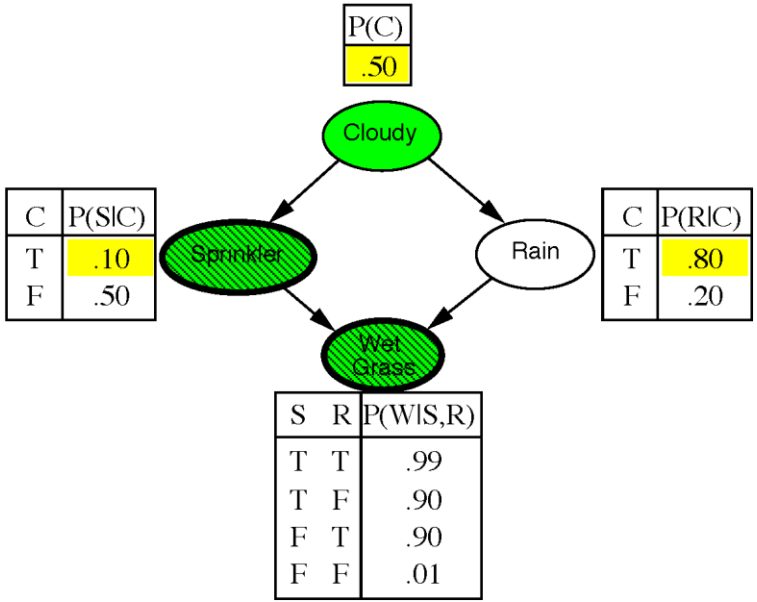
Sample Cloudy: true



$w = 1.0$

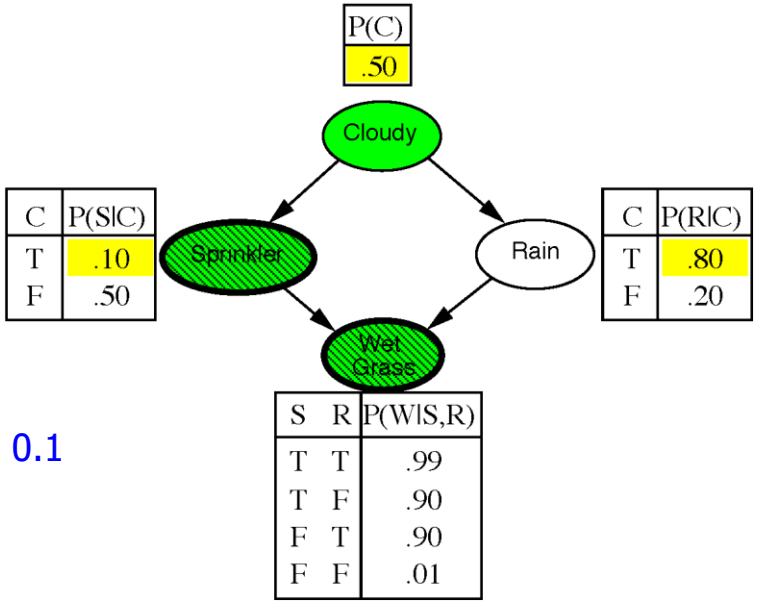
Likelihood weighting example

Cloudy=true now given for the remainder of this sample



$w = 1.0$

Likelihood weighting example



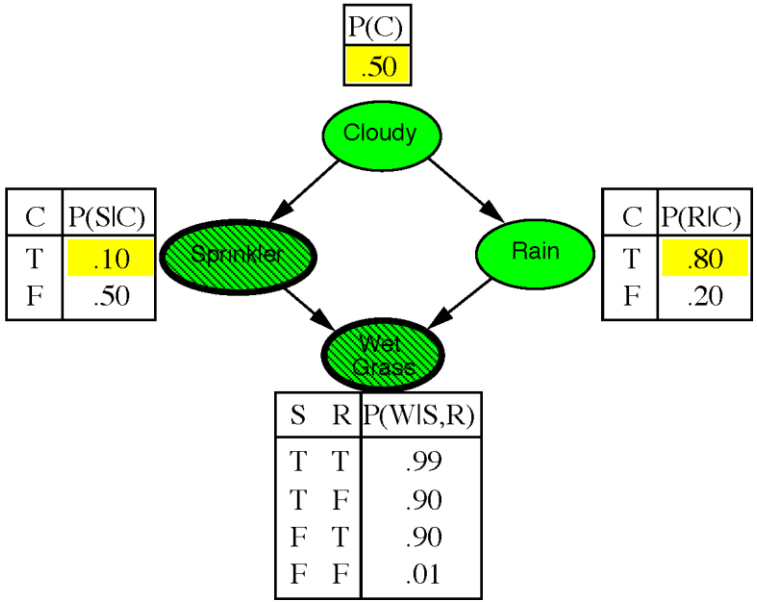
Sprinkler=true was given.

So, update w with
P(Sprinkler=true|parents) =
P(Sprinkler=true|Cloudy=true) = 0.1

$w = 1.0 \times 0.1$

Likelihood weighting example

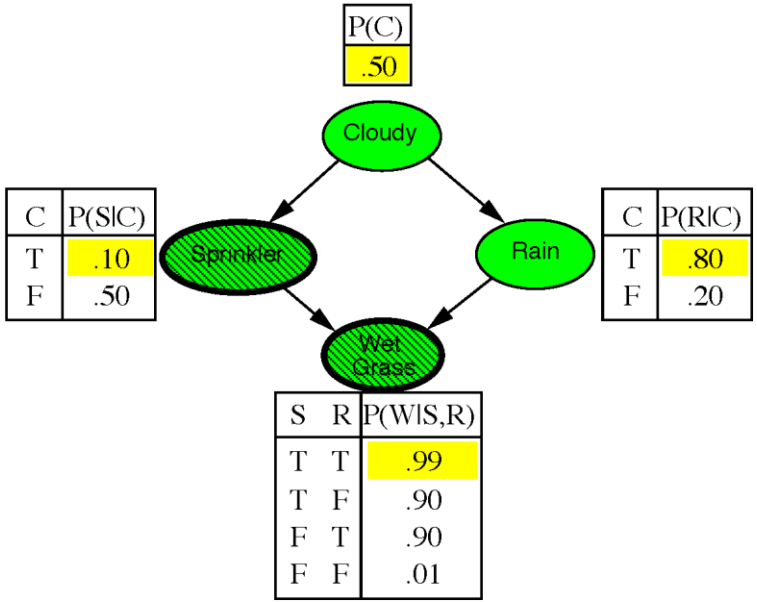
Sample Rain: say, true



$w = 1.0 \times 0.1$

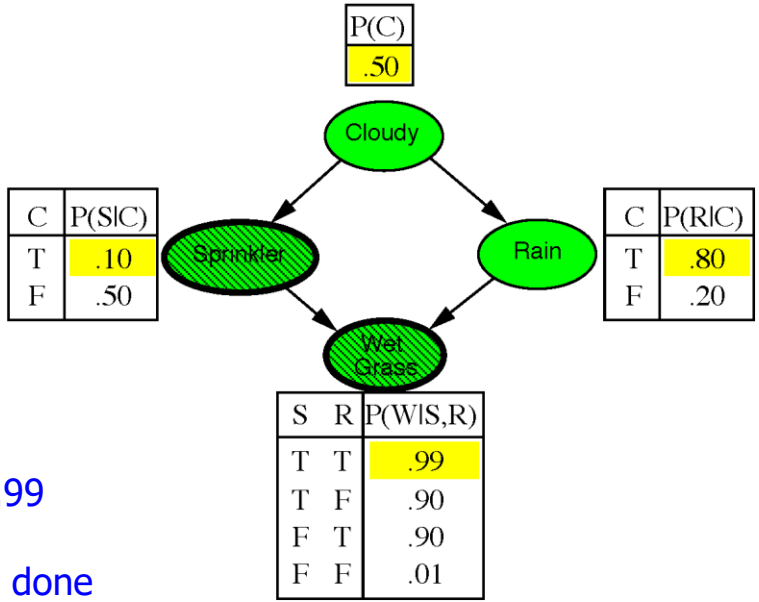
Likelihood weighting example

Rain=true now given for the remainder of this sample



$w = 1.0 \times 0.1$

Likelihood weighting example



Wet Grass=true was given.

So, update w with
P(Wet Grass=true|parents) =
P(Wet Grass=true|
Sprinkler=true,Rain=true) = 0.99

All variable assigned or sampled: done for this sample.

$$w = 1.0 \times 0.1 \times 0.99 = 0.099$$

Return event (true, true, true, true) with weight 0.099

Likelihood weighting analysis

Sampling probability for WEIGHTEDSAMPLE is

$$S_{WS}(\mathbf{y}, \mathbf{e}) = \prod_{i=1}^{l} P(y_i | Parents(Y_i))$$

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

$$w(\mathbf{y}, \mathbf{e}) = \prod_{i=1}^{m} P(e_i | Parents(E_i))$$

Weighted sampling probability is

$$\begin{aligned} & S_{WS}(\mathbf{y}, \mathbf{e}) w(\mathbf{y}, \mathbf{e}) \\ & = \prod_{i=1}^{l} P(y_i | Parents(Y_i)) \prod_{i=1}^{m} P(e_i | Parents(E_i)) \\ & = P(\mathbf{y}, \mathbf{e}) \text{ (by standard global semantics of network)} \end{aligned}$$

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

Approximate inference using Markov Chain Monte Carlo (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 $P(X|\mathbf{e})$
  local variables: $\mathbf{N}[X]$, a vector of counts over X, initially zero
    $\mathbf{Y}$, the nonevidence variables in bn
    $\mathbf{x}$, the current state of the network, initially copied from **e**

  initialize $\mathbf{x}$ with random values for the variables in $\mathbf{Y}$
  for $j = 1$ to N **do**
    $\mathbf{N}[x] \leftarrow \mathbf{N}[x] + 1$ where x is the value of X in $\mathbf{x}$
    for each Y_i in $\mathbf{Y}$ **do**
      sample the value of Y_i in $\mathbf{x}$ from $\mathbf{P}(Y_i|MB(Y_i))$ given the values of $MB(Y_i)$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{N}[X]$)

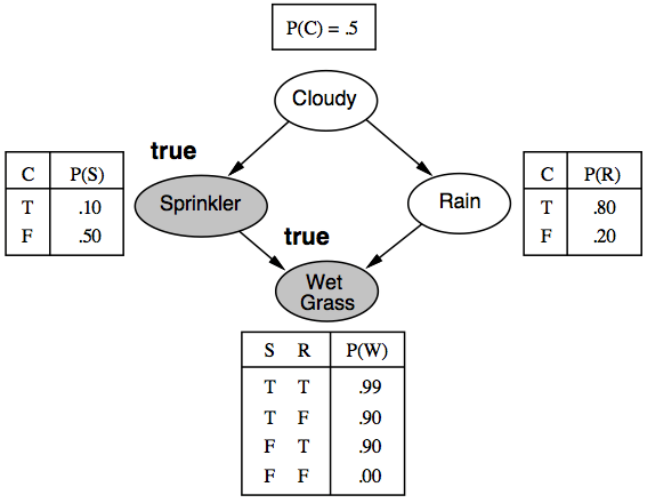
Approaches stationary distribution: long-run fraction of time spent in each state is exactly proportional to its posterior probability

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 $WetGrass$

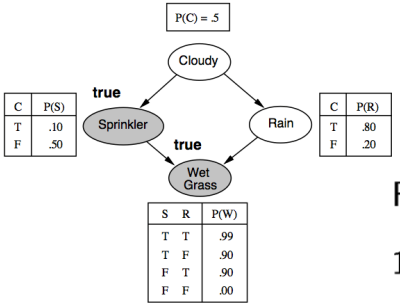


MCMC example

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 $WetGrass$



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

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

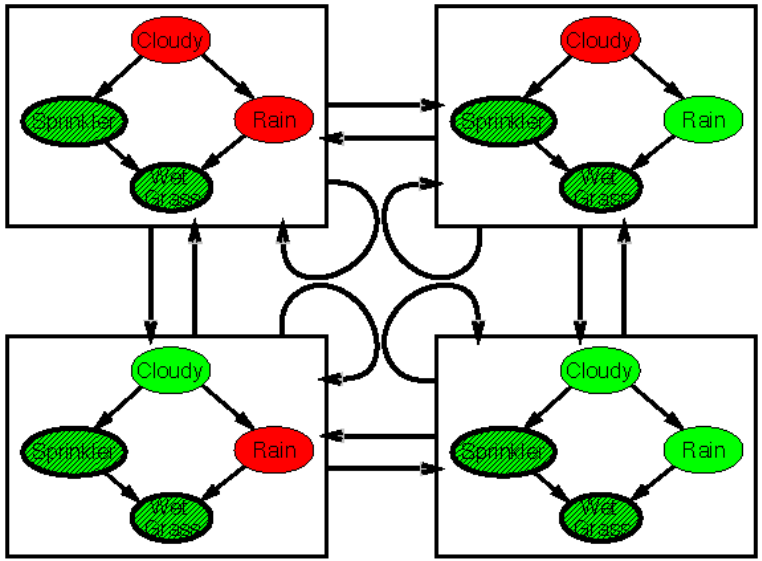
2. $\mathbf{P}(Rain|MB(Rain)) = \mathbf{P}(Rain|\neg Cloudy, Sprinkler, WetGrass)$
   sample $\rightarrow true$

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

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

The Markov chain

With $Sprinkler = true, WetGrass = true$, there are four states:



Wander about for a while, average what you see

MCMC analysis

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 $q(\mathbf{y} \to \mathbf{y}')$

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

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

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

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

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

Detailed balance

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

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

Detailed balance $\Rightarrow$ stationarity:

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

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

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

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

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

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:

$$P(y_i'|MB(Y_i)) = P(y_i'|Parents(Y_i))\Pi_{Z_j \in Children(Y_i)} P(z_j|Parents(Z_j))$$

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)