Probabilistic Reasoning

Dr. Steven Bethard

Computer and Information Sciences
University of Alabama at Birmingham

15 Mar 2016

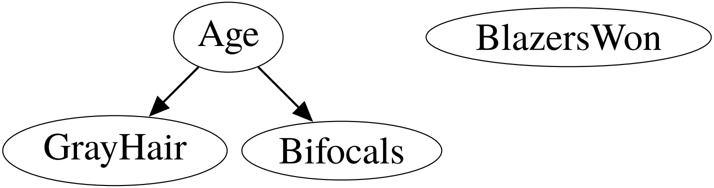
Outline

Outline

Bayesian Networks

Definition

A Bayesian Network is a data structure for representing independence relations among random variables



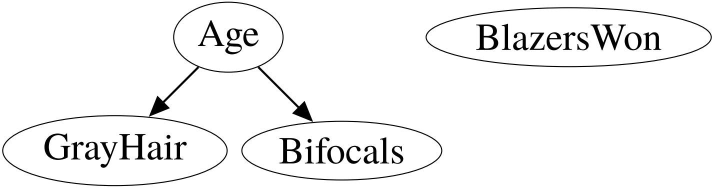
$$P(Age, GrayHair, Bifocals, BlazersWon) = P(Age, GrayHair, Bifocals)P(BlazersWon)$$

$$P(GrayHair, Bifocals|Age) = P(GrayHair|Age)P(Bifocals|Age)$$

Bayesian Networks

Definition

A Bayesian Network is a data structure for representing independence relations among random variables



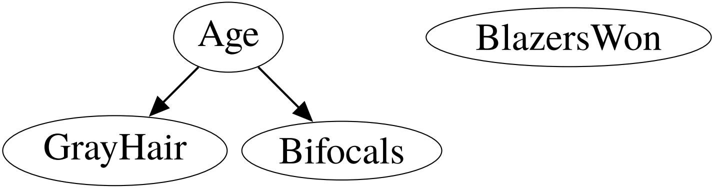
$$P(Age, GrayHair, Bifocals, BlazersWon) = P(Age, GrayHair, Bifocals)P(BlazersWon)$$

$$P(GrayHair, Bifocals|Age) = P(GrayHair|Age)P(Bifocals|Age)$$

Bayesian Networks

Definition

A Bayesian Network is a data structure for representing independence relations among random variables



$$P(Age, GrayHair, Bifocals, BlazersWon) = P(Age, GrayHair, Bifocals)P(BlazersWon)$$

$$P(GrayHair, Bifocals|Age) = P(GrayHair|Age)P(Bifocals|Age)$$

Bayesian Networks

Components

- Random variables (nodes)
- Directed links from *parent* nodes to *child* nodes
- $\mathbf{P}(X_i | Parents(X_i))$ tables for each node
- Links form no cycles

Intuitions

- Links indicate *direct* influence
- Causes usually near top
- Effects usually near bottom

Bayesian Networks

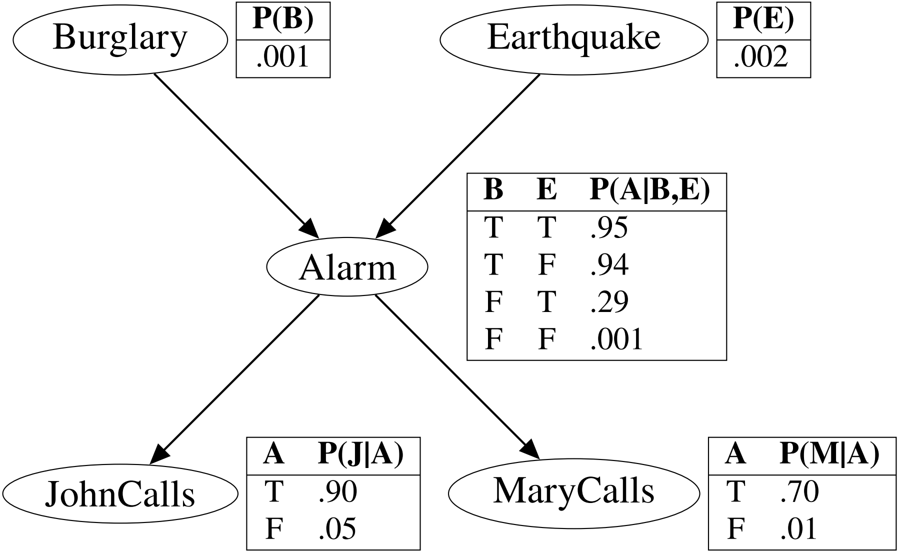
Components

- Random variables (nodes)
- Directed links from *parent* nodes to *child* nodes
- $\mathbf{P}(X_i | Parents(X_i))$ tables for each node
- Links form no cycles

Intuitions

- Links indicate *direct* influence
- Causes usually near top
- Effects usually near bottom

Full Bayesian Network Example



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

Representing the Full Joint Distribution

Key Formula

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

Example

$$\begin{aligned} & P(j, m, a, \neg b, \neg e) \\ = \; & P(j|parents(j)) \cdot P(m|parents(m)) \cdot \ldots \\ = \; & P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\ = \; & 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \\ = \; & 0.00062 \end{aligned}$$

Representing the Full Joint Distribution

Key Formula

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

Example

$$\begin{aligned}
& P(j, m, a, \neg b, \neg e) \\
&= P(j|parents(j)) \cdot P(m|parents(m)) \cdot \ldots \\
&= P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\
&= 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \\
&= 0.00062
\end{aligned}$$

Representing the Full Joint Distribution

Key Formula

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

Example

$$\begin{aligned}
&P(j, m, a, \neg b, \neg e) \\
&\quad = \quad P(j|parents(j)) \cdot P(m|parents(m)) \cdot \ldots \\
&\quad = \quad P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\
&\quad = \quad 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \\
&\quad = \quad 0.00062
\end{aligned}$$

Representing the Full Joint Distribution

Key Formula

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

Example

$$\begin{aligned} & P(j, m, a, \neg b, \neg e) \\ = \; & P(j|parents(j)) \cdot P(m|parents(m)) \cdot \ldots \\ = \; & P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\ = \; & 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \\ = \; & 0.00062 \end{aligned}$$

Representing the Full Joint Distribution

Key Formula

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

Example

$$\begin{aligned}
&P(j, m, a, \neg b, \neg e) \\
&\quad = P(j | parents(j)) \cdot P(m | parents(m)) \cdot \ldots \\
&\quad = P(j|a) \cdot P(m|a) \cdot P(a|\neg b, \neg e) \cdot P(\neg b) \cdot P(\neg e) \\
&\quad = 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \\
&\quad = 0.00062
\end{aligned}$$

A Simple Inference Algorithm

Goal: Answer Queries

- One query variable given some evidence
- $\mathbf{P}(X|y_1, \ldots, y_n)$

Solution: Enumeration

$$
\begin{aligned}
&\mathbf{P}(X|y_1, \ldots, y_n) \\
&= \alpha \mathbf{P}(X, y_1, \ldots, y_n) \\
&= \alpha \sum_{z_1, \ldots, z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X, y_1, \ldots, y_n, z_1, \ldots, z_k) \\
&= \alpha \sum_{z_1, \ldots, z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X| \ldots) P(y_1| \ldots) \ldots P(z_k| \ldots)
\end{aligned}
$$

A Simple Inference Algorithm

Goal: Answer Queries

- One query variable given some evidence
- $\mathbf{P}(X|y_1, \ldots, y_n)$

Solution: Enumeration

$$
\begin{aligned}
&\mathbf{P}(X|y_1, \ldots, y_n) \\
&\quad = \alpha \mathbf{P}(X, y_1, \ldots, y_n) \\
&\quad = \alpha \sum_{z_1, \ldots, z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X, y_1, \ldots, y_n, z_1, \ldots, z_k) \\
&\quad = \alpha \sum_{z_1, \ldots, z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X| \ldots) P(y_1| \ldots) \ldots P(z_k| \ldots)
\end{aligned}
$$

Enumeration Worst Case

Enumeration Formula

$$\mathbf{P}(X|y_1,\ldots,y_n) = \alpha \sum_{z_1,\ldots,z_k\in\overline{\mathbf{XY}}} \mathbf{P}(X|\ldots)P(y_1|\ldots)\ldots P(z_k|\ldots)$$

Query: $P(X_5)$

Worst Case in General

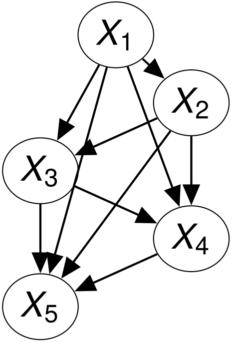
- $\approx n$ parents per node
- $\approx n$ variables not in query

Time Complexity: $O(n \cdot 2^n)$

Enumeration Worst Case

Enumeration Formula

$$\mathbf{P}(X|y_1,\ldots,y_n) = \alpha \sum_{z_1,\ldots,z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X|\ldots)P(y_1|\ldots)\ldots P(z_k|\ldots)$$



Query: $P(X_5)$

Worst Case in General

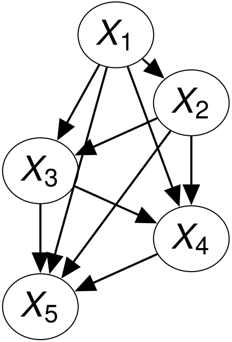
- $\approx n$ parents per node
- $\approx n$ variables not in query

Time Complexity: $O(n \cdot 2^n)$

Enumeration Worst Case

Enumeration Formula

$$\mathbf{P}(X|y_1,\ldots,y_n) = \alpha \sum_{z_1,\ldots,z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X|\ldots)P(y_1|\ldots)\ldots P(z_k|\ldots)$$



Query: $P(X_5)$

Worst Case in General

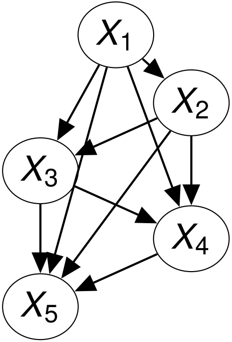
- $\approx n$ parents per node
- $\approx n$ variables not in query

Time Complexity: $O(n \cdot 2^n)$

Enumeration Worst Case

Enumeration Formula

$$\mathbf{P}(X|y_1,\ldots,y_n) = \alpha \sum_{z_1,\ldots,z_k \in \overline{\mathbf{XY}}} \mathbf{P}(X|\ldots)P(y_1|\ldots)\ldots P(z_k|\ldots)$$



Query: $P(X_5)$

Worst Case in General

- $\approx n$ parents per node
- $\approx n$ variables not in query

Time Complexity: $O(n \cdot 2^n)$

Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:
Age, *GrayHair*, *Bifocals*, *ReadDist*

Bad:
ReadDist, *GrayHair*, *Bifocals*, *Age*

Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:
Age, *GrayHair*, *Bifocals*, *ReadDist*

Bad:
ReadDist, *GrayHair*, *Bifocals*, *Age*

Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*

Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

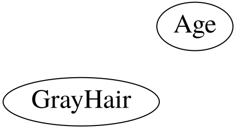
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

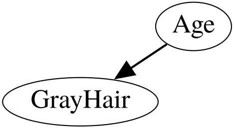
ReadDist, *GrayHair*, *Bifocals*, *Age*

Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:
Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:
ReadDist, *GrayHair*, *Bifocals*, *Age*

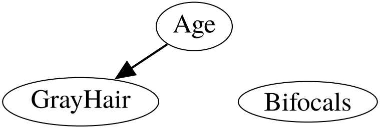
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

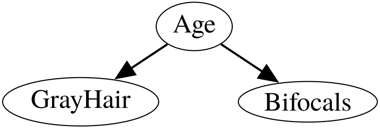
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

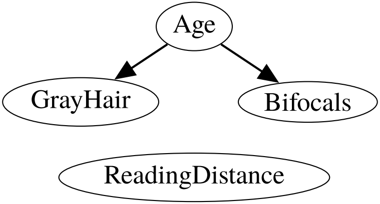
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

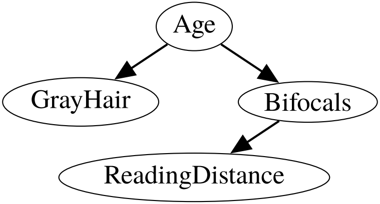
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

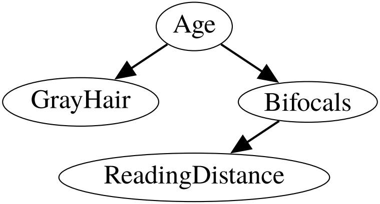
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

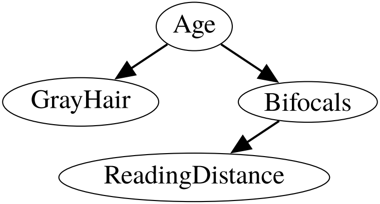
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*

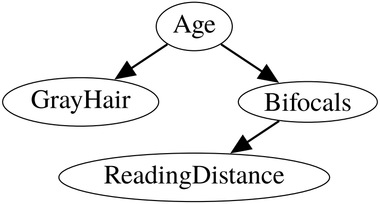
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

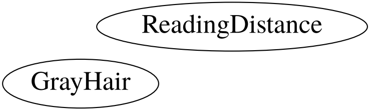
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



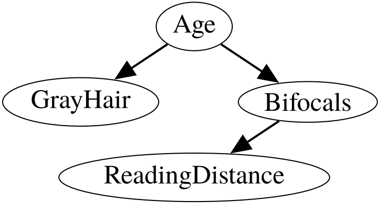
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

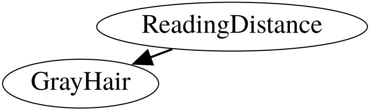
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



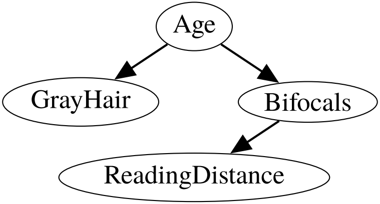
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

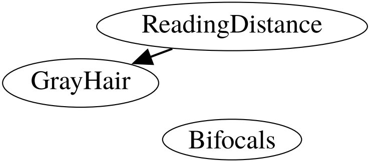
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



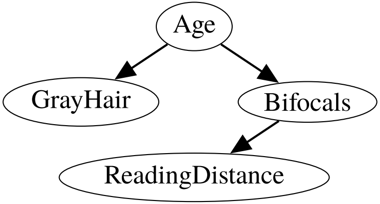
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

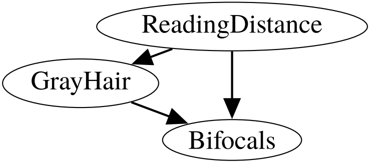
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



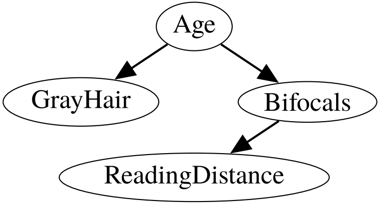
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

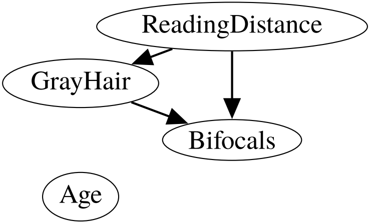
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



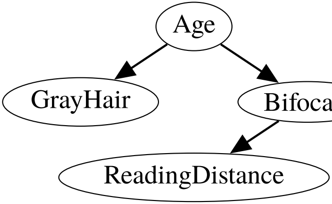
Avoiding Fully Connected Networks

Principles

1. Add root causes
2. Add variables directly influenced by leaves
3. If variables left, goto 2

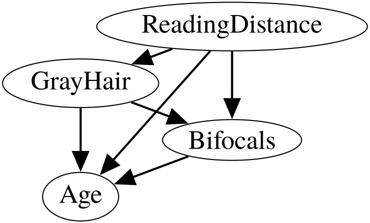
Good:

Age, *GrayHair*, *Bifocals*, *ReadDist*



Bad:

ReadDist, *GrayHair*, *Bifocals*, *Age*



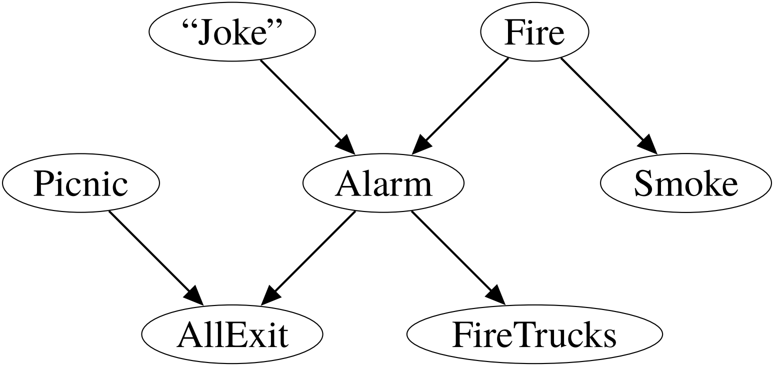
Bayesian Network Exercise

Construct a Network

- The fire alarm usually goes off when there's a fire
- When the alarm rings everyone usually exits together
- Most of the time there's smoke when there's a fire
- Someone sometimes pulls the fire alarm "as a joke"
- The fire trucks usually come when the alarm goes off
- Sometimes everyone exits together for a picnic

Bayesian Network Exercise

One possible solution:



Outline

Drawbacks of Simple Enumeration

Recall: Simple Enumeration

Time Complexity: $O(n \cdot 2^n)$

Example

$$\begin{aligned} P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha(P(b)P(e)P(a|b,e)P(j|a)P(m|a) + \\ &\quad P(b)P(e)P(\neg a|b,e)P(j|\neg a)P(m|\neg a) + \\ &\quad P(b)P(\neg e)P(a|b,\neg e)P(j|a)P(m|a) + \\ &\quad P(b)P(\neg e)P(\neg a|b,\neg e)P(j|\neg a)P(m|\neg a)) \end{aligned}$$

Problem: We calculate $P(b)$, $P(e)$ and $P(\neg e)$ many times!

Drawbacks of Simple Enumeration

Recall: Simple Enumeration

Time Complexity: $O(n \cdot 2^n)$

Example

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha(P(b)P(e)P(a|b,e)P(j|a)P(m|a) + \\
&\quad P(b)P(e)P(\neg a|b,e)P(j|\neg a)P(m|\neg a) + \\
&\quad P(b)P(\neg e)P(a|b,\neg e)P(j|a)P(m|a) + \\
&\quad P(b)P(\neg e)P(\neg a|b,\neg e)P(j|\neg a)P(m|\neg a))
\end{aligned}
$$

Problem: We calculate $P(b)$, $P(e)$ and $P(\neg e)$ many times!

Drawbacks of Simple Enumeration

Recall: Simple Enumeration

Time Complexity: $O(n \cdot 2^n)$

Example

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha(P(b)P(e)P(a|b,e)P(j|a)P(m|a) + \\
&\quad P(b)P(e)P(\neg a|b,e)P(j|\neg a)P(m|\neg a) + \\
&\quad P(b)P(\neg e)P(a|b,\neg e)P(j|a)P(m|a) + \\
&\quad P(b)P(\neg e)P(\neg a|b,\neg e)P(j|\neg a)P(m|\neg a))
\end{aligned}
$$

Problem: We calculate $P(b)$, $P(e)$ and $P(\neg e)$ many times!

Drawbacks of Simple Enumeration

Recall: Simple Enumeration

Time Complexity: $O(n \cdot 2^n)$

Example

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha (P(b)P(e)P(a|b,e)P(j|a)P(m|a) + \\
&\quad P(b)P(e)P(\neg a|b,e)P(j|\neg a)P(m|\neg a) + \\
&\quad P(b)P(\neg e)P(a|b,\neg e)P(j|a)P(m|a) + \\
&\quad P(b)P(\neg e)P(\neg a|b,\neg e)P(j|\neg a)P(m|\neg a))
\end{aligned}
$$

Problem: We calculate $P(b)$, $P(e)$ and $P(\neg e)$ many times!

More Intelligent Summation

Moving Terms in Algebra

$$\begin{aligned} abd + abe + acf + acg \\ &= a(bd + be + cf + cg) \\ &= a(b(d+e) + c(f+g)) \end{aligned}$$

Moving Terms in Bayesian Network Calculations

$$\begin{aligned} P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a') \end{aligned}$$

More Intelligent Summation

Moving Terms in Algebra

$$
\begin{aligned}
& abd + abe + acf + acg \\
& \quad = a(bd + be + cf + cg) \\
& \quad = a(b(d+e) + c(f+g))
\end{aligned}
$$

Moving Terms in Bayesian Network Calculations

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a')
\end{aligned}
$$

More Intelligent Summation

Moving Terms in Algebra

$$\begin{aligned} & abd + abe + acf + acg \\ &\quad = a(bd + be + cf + cg) \\ &\quad = a(b(d + e) + c(f + g)) \end{aligned}$$

Moving Terms in Bayesian Network Calculations

$$\begin{aligned} P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a') \end{aligned}$$

More Intelligent Summation

Moving Terms in Algebra

$$
\begin{aligned}
& abd + abe + acf + acg \\
& \quad = a(bd + be + cf + cg) \\
& \quad = a(b(d + e) + c(f + g))
\end{aligned}
$$

Moving Terms in Bayesian Network Calculations

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a')
\end{aligned}
$$

More Intelligent Summation

Moving Terms in Algebra

$$\begin{aligned} & abd + abe + acf + acg \\ &= a(bd + be + cf + cg) \\ &= a(b(d+e) + c(f+g)) \end{aligned}$$

Moving Terms in Bayesian Network Calculations

$$\begin{aligned} P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\ &= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a') \end{aligned}$$

More Intelligent Summation

Moving Terms in Algebra

$$
\begin{aligned}
& abd + abe + acf + acg \\
& \quad = a(bd + be + cf + cg) \\
& \quad = a(b(d + e) + c(f + g))
\end{aligned}
$$

Moving Terms in Bayesian Network Calculations

$$
\begin{aligned}
P(b|j,m) &= \alpha \sum_{e'} \sum_{a'} P(b)P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} \sum_{a'} P(e')P(a'|b,e')P(j|a')P(m|a') \\
&= \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a')
\end{aligned}
$$

Calculations through Depth First Search

$$P(b|j,m) = \alpha P(b) \sum_{e'} P(e') \sum_{a'} P(a'|b,e')P(j|a')P(m|a')$$

$$\begin{tikzcd}[column sep=small]
& & & P(b) \arrow[dll, dash] \arrow[drr, dash] & & & \\
& P(e) \arrow[dl, dash] \arrow[dr, dash] & & & & P(\neg e) \arrow[dl, dash] \arrow[dr, dash] & \\
P(a|b,e) \arrow[d, dash] & & P(\neg a|b,e) \arrow[d, dash] & & P(a|b,\neg e) \arrow[d, dash] & & P(\neg a|b,\neg e) \arrow[d, dash] \\
P(j|a) \arrow[d, dash] & & P(j|\neg a) \arrow[d, dash] & & P(j|a) \arrow[d, dash] & & P(j|\neg a) \arrow[d, dash] \\
P(m|a) & & P(m|\neg a) & & P(m|a) & & P(m|\neg a)
\end{tikzcd}$$

Factors

Avoid duplicate calculations by using factors, which store:

- A set of variables
- A number for each possible assignment of values

Factors

Avoid duplicate calculations by using factors, which store:

- A set of variables
- A number for each possible assignment of values

$$P(m|A) \quad = f(A) \quad = \begin{pmatrix} a & \rightarrow & 0.70 \\ \neg a & \rightarrow & 0.01 \end{pmatrix}$$

Factors

Avoid duplicate calculations by using factors, which store:

- A set of variables
- A number for each possible assignment of values

$$
P(m|A) = f(A) = \begin{pmatrix} a & \rightarrow & 0.70 \\ \neg a & \rightarrow & 0.01 \end{pmatrix}
$$

$$
P(A|B,E) = g(A,B,E) = \begin{pmatrix} a & b & e & \rightarrow & 0.95 \\ a & b & \neg e & \rightarrow & 0.94 \\ a & \neg b & e & \rightarrow & 0.29 \\ a & \neg b & \neg e & \rightarrow & 0.001 \\ \neg a & b & e & \rightarrow & 0.05 \\ \neg a & b & \neg e & \rightarrow & 0.06 \\ \neg a & \neg b & e & \rightarrow & 0.71 \\ \neg a & b & \neg e & \rightarrow & 0.999 \end{pmatrix}
$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \rightarrow & 0.90 \\ j & \neg a & \rightarrow & 0.05 \\ \neg j & a & \rightarrow & 0.10 \\ \neg j & \neg a & \rightarrow & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \rightarrow & 0.70 \\ m & \neg a & \rightarrow & 0.01 \\ \neg m & a & \rightarrow & 0.30 \\ \neg m & \neg a & \rightarrow & 0.99 \end{pmatrix} =$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \rightarrow & 0.90 \\ j & \neg a & \rightarrow & 0.05 \\ \neg j & a & \rightarrow & 0.10 \\ \neg j & \neg a & \rightarrow & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \rightarrow & 0.70 \\ m & \neg a & \rightarrow & 0.01 \\ \neg m & a & \rightarrow & 0.30 \\ \neg m & \neg a & \rightarrow & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \rightarrow & \\ j & m & \neg a & \rightarrow & \\ j & \neg m & a & \rightarrow & \\ j & \neg m & \neg a & \rightarrow & \\ \neg j & m & a & \rightarrow & \\ \neg j & m & \neg a & \rightarrow & \\ \neg j & \neg m & a & \rightarrow & \\ \neg j & \neg m & \neg a & \rightarrow & \end{pmatrix}$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \to & 0.90 \\ j & \neg a & \to & 0.05 \\ \neg j & a & \to & 0.10 \\ \neg j & \neg a & \to & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \to & 0.70 \\ m & \neg a & \to & 0.01 \\ \neg m & a & \to & 0.30 \\ \neg m & \neg a & \to & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \to & 0.63 \\ j & m & \neg a & \to & \\ j & \neg m & a & \to & \\ j & \neg m & \neg a & \to & \\ \neg j & m & a & \to & \\ \neg j & m & \neg a & \to & \\ \neg j & \neg m & a & \to & \\ \neg j & \neg m & \neg a & \to & \end{pmatrix}$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \rightarrow & 0.90 \\ j & \neg a & \rightarrow & 0.05 \\ \neg j & a & \rightarrow & 0.10 \\ \neg j & \neg a & \rightarrow & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \rightarrow & 0.70 \\ m & \neg a & \rightarrow & 0.01 \\ \neg m & a & \rightarrow & 0.30 \\ \neg m & \neg a & \rightarrow & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & \\ j & \neg m & \neg a & \rightarrow & \\ \neg j & m & a & \rightarrow & \\ \neg j & m & \neg a & \rightarrow & \\ \neg j & \neg m & a & \rightarrow & \\ \neg j & \neg m & \neg a & \rightarrow & \end{pmatrix}$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \to & 0.90 \\ j & \neg a & \to & 0.05 \\ \neg j & a & \to & 0.10 \\ \neg j & \neg a & \to & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \to & 0.70 \\ m & \neg a & \to & 0.01 \\ \neg m & a & \to & 0.30 \\ \neg m & \neg a & \to & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \to & 0.63 \\ j & m & \neg a & \to & 0.0005 \\ j & \neg m & a & \to & 0.27 \\ j & \neg m & \neg a & \to & \\ \neg j & m & a & \to & \\ \neg j & m & \neg a & \to & \\ \neg j & \neg m & a & \to & \\ \neg j & \neg m & \neg a & \to & \end{pmatrix}$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \rightarrow & 0.90 \\ j & \neg a & \rightarrow & 0.05 \\ \neg j & a & \rightarrow & 0.10 \\ \neg j & \neg a & \rightarrow & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \rightarrow & 0.70 \\ m & \neg a & \rightarrow & 0.01 \\ \neg m & a & \rightarrow & 0.30 \\ \neg m & \neg a & \rightarrow & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & \\ \neg j & m & \neg a & \rightarrow & \\ \neg j & \neg m & a & \rightarrow & \\ \neg j & \neg m & \neg a & \rightarrow & \end{pmatrix}$$

Factor Product

The factor product of factors f and g yields a factor with:

- The union of the variables of f and g
- Values that are the product of the corresponding rows

$$\begin{pmatrix} j & a & \rightarrow & 0.90 \\ j & \neg a & \rightarrow & 0.05 \\ \neg j & a & \rightarrow & 0.10 \\ \neg j & \neg a & \rightarrow & 0.95 \end{pmatrix} \times \begin{pmatrix} m & a & \rightarrow & 0.70 \\ m & \neg a & \rightarrow & 0.01 \\ \neg m & a & \rightarrow & 0.30 \\ \neg m & \neg a & \rightarrow & 0.99 \end{pmatrix} = \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_{A} \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow \\ j & \neg m & \rightarrow \\ \neg j & m & \rightarrow \\ \neg j & \neg m & \rightarrow \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_A \begin{pmatrix} j & m & a & \to & 0.63 \\ j & m & \neg a & \to & 0.0005 \\ j & \neg m & a & \to & 0.27 \\ j & \neg m & \neg a & \to & 0.0495 \\ \neg j & m & a & \to & 0.07 \\ \neg j & m & \neg a & \to & 0.0095 \\ \neg j & \neg m & a & \to & 0.03 \\ \neg j & \neg m & \neg a & \to & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \to & \\ j & \neg m & \to & \\ \neg j & m & \to & \\ \neg j & \neg m & \to & \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_A \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow & 0.6305 \\ j & \neg m & \rightarrow & 0.3195 \\ \neg j & m & \rightarrow & 0.0795 \\ \neg j & \neg m & \rightarrow & 0.9705 \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_{A} \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow & 0.6305 \\ j & \neg m & \rightarrow & 0.3195 \\ \neg j & m & \rightarrow & 0.0795 \\ \neg j & \neg m & \rightarrow & 0.9705 \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_A \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow & 0.6305 \\ j & \neg m & \rightarrow & 0.3195 \\ \neg j & m & \rightarrow & 0.0795 \\ \neg j & \neg m & \rightarrow & 0.9705 \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_{A} \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow & 0.6305 \\ j & \neg m & \rightarrow & 0.3195 \\ \neg j & m & \rightarrow & 0.0795 \\ \neg j & \neg m & \rightarrow & 0.9705 \end{pmatrix}$$

Factor Marginalization

The factor marginalization of factor f for variable A yields a factor with:

- The variables of f, minus the variable A
- Values that are the sum of the corresponding rows

$$\sum_A \begin{pmatrix} j & m & a & \rightarrow & 0.63 \\ j & m & \neg a & \rightarrow & 0.0005 \\ j & \neg m & a & \rightarrow & 0.27 \\ j & \neg m & \neg a & \rightarrow & 0.0495 \\ \neg j & m & a & \rightarrow & 0.07 \\ \neg j & m & \neg a & \rightarrow & 0.0095 \\ \neg j & \neg m & a & \rightarrow & 0.03 \\ \neg j & \neg m & \neg a & \rightarrow & 0.9405 \end{pmatrix} = \begin{pmatrix} j & m & \rightarrow & 0.6305 \\ j & \neg m & \rightarrow & 0.3195 \\ \neg j & m & \rightarrow & 0.0795 \\ \neg j & \neg m & \rightarrow & 0.9705 \end{pmatrix}$$

Factor Exercise

$$\sum_A \left(\begin{pmatrix} x & a & \rightarrow & 1 \\ x & \neg a & \rightarrow & 4 \\ y & a & \rightarrow & 3 \\ y & \neg a & \rightarrow & 2 \\ z & a & \rightarrow & 2 \\ z & \neg a & \rightarrow & 5 \end{pmatrix} \times \begin{pmatrix} a & i & \rightarrow & 3 \\ a & j & \rightarrow & 6 \\ \neg a & i & \rightarrow & 2 \\ \neg a & j & \rightarrow & 4 \end{pmatrix} \right) =$$

Factor Exercise

$$\sum_{A}\left(\begin{pmatrix} x & a & \rightarrow & 1 \\ x & \neg a & \rightarrow & 4 \\ y & a & \rightarrow & 3 \\ y & \neg a & \rightarrow & 2 \\ z & a & \rightarrow & 2 \\ z & \neg a & \rightarrow & 5 \end{pmatrix} \times \begin{pmatrix} a & i & \rightarrow & 3 \\ a & j & \rightarrow & 6 \\ \neg a & i & \rightarrow & 2 \\ \neg a & j & \rightarrow & 4 \end{pmatrix}\right) = \begin{pmatrix} x & i & \rightarrow & 11 \\ x & j & \rightarrow & 22 \\ y & i & \rightarrow & 13 \\ y & j & \rightarrow & 26 \\ z & i & \rightarrow & 16 \\ z & j & \rightarrow & 32 \end{pmatrix}$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m) \\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')
\end{aligned}
$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m)\\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')\\
&= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \sum_{E} \begin{pmatrix} e & .002 \\ \neg e & .998 \end{pmatrix} \times \sum_{A} \begin{pmatrix} a & b & e & .95 \\ a & b & \neg e & .94 \\ a & \neg b & e & .29 \\ a & \neg b & \neg e & .001 \\ \neg a & b & e & .05 \\ \neg a & b & \neg e & .06 \\ \neg a & \neg b & e & .71 \\ \neg a & \neg b & \neg e & .999 \end{pmatrix} \times \begin{pmatrix} a & .9 \\ \neg a & .05 \end{pmatrix} \times \begin{pmatrix} a & .7 \\ \neg a & .01 \end{pmatrix}
\end{aligned}
$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m)\\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')\\
&= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \sum_{E} \begin{pmatrix} e & .002 \\ \neg e & .998 \end{pmatrix} \times \sum_{A} \begin{pmatrix} a & b & e & .95 \\ a & b & \neg e & .94 \\ a & \neg b & e & .29 \\ a & \neg b & \neg e & .001 \\ \neg a & b & e & .05 \\ \neg a & b & \neg e & .06 \\ \neg a & \neg b & e & .71 \\ \neg a & \neg b & \neg e & .999 \end{pmatrix} \times \begin{pmatrix} a & .63 \\ \neg a & .0005 \end{pmatrix}
\end{aligned}
$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m) \\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a') \\
&= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \sum_{E} \begin{pmatrix} e & .002 \\ \neg e & .998 \end{pmatrix} \times \sum_{A} \begin{pmatrix} a & b & e & .5985 \\ a & b & \neg e & .5922 \\ a & \neg b & e & .1827 \\ a & \neg b & \neg e & .00063 \\ \neg a & b & e & .000025 \\ \neg a & b & \neg e & .00003 \\ \neg a & \neg b & e & .000355 \\ \neg a & \neg b & \neg e & .0004995 \end{pmatrix}
\end{aligned}
$$

Exact Inference via Factors

$$P(B|j,m)$$
$$= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')$$

$$= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \sum_{E} \begin{pmatrix} e & .002 \\ \neg e & .998 \end{pmatrix} \times \begin{pmatrix} b & e & .598525 \\ b & \neg e & .59223 \\ \neg b & e & .183055 \\ \neg b & \neg e & .0011295 \end{pmatrix}$$

Exact Inference via Factors

$$P(B|j,m)$$
$$= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')$$

$$= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \sum_{E} \begin{pmatrix} b & e & .00119705 \\ b & \neg e & .59104554 \\ \neg b & e & .00036611 \\ \neg b & \neg e & .001127241 \end{pmatrix}$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m) \\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')
\end{aligned}
$$

$$
= \alpha \begin{pmatrix} b & .001 \\ \neg b & .999 \end{pmatrix} \times \begin{pmatrix} b & .59224259 \\ \neg b & .001493351 \end{pmatrix}
$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m)\\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')
\end{aligned}
$$

$$
= \alpha \begin{pmatrix} b & .00059224259 \\ \neg b & .001491857649 \end{pmatrix}
$$

Exact Inference via Factors

$$
\begin{aligned}
&P(B|j,m) \\
&= \alpha P(B) \sum_{e' \in E} P(e') \sum_{a' \in A} P(a'|B,e')P(j|a')P(m|a')
\end{aligned}
$$

$$
\approx \begin{pmatrix} b & .284 \\ \neg b & .716 \end{pmatrix}
$$

Exact Inference Properties

Given a network with:

- v variables
- p parents per variable
- r rows in the conditional probability tables

Worst case time and space complexity:

Singly connected $O(r)$

If p constant-bounded $\Rightarrow O(v)$

Multiply connected $O(2^v)$

Cluster variables into tree $\Rightarrow O(r_{clust})$

Exact Inference Properties

Given a network with:

- v variables
- p parents per variable
- r rows in the conditional probability tables

Worst case time and space complexity:

Singly connected	$O(r)$
	If p constant-bounded $\Rightarrow O(v)$
Multiply connected	$O(2^v)$
	Cluster variables into tree $\Rightarrow O(r_{clust})$

Exact Inference Properties

Given a network with:

- v variables
- p parents per variable
- r rows in the conditional probability tables

Worst case time and space complexity:

Singly connected	$O(r)$
	If p constant-bounded $\Rightarrow O(v)$
Multiply connected	$O(2^v)$
	Cluster variables into tree $\Rightarrow O(r_{clust})$

Exact Inference Properties

Given a network with:

- v variables
- p parents per variable
- r rows in the conditional probability tables

Worst case time and space complexity:

Singly connected	$O(r)$
	If p constant-bounded $\Rightarrow O(v)$
Multiply connected	$O(2^v)$
	Cluster variables into tree $\Rightarrow O(r_{clust})$

Outline

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```python
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```python
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x | Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Approximate Inference

Given query $P(X = x|Y_1 = z_1, \ldots, Y_n = z_n)$:

- Generate k assignments of all variables in network
- Drop assignments inconsistent with $Y_1 = z_1, \ldots, Y_n = z_n$
- Count assignments where $X = x$, and divide by k

```python
def prior_sample(bayes_net):
    # generate a value for each variable in the network
    # variables are sorted from parents to children
    sample = {}
    for variable in bayes_net:
        # find the values assigned to the parents
        parent_values = [sample[parent] for parent in variable.parents]
        # find the probability for this assignment from the table
        probability = variable.probability_of(True, *parent_values)
        # add True or False according to the distribution
        sample[variable] = random.random() < probability
    # return the complete sample
    return sample
```

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$

Query: $\mathsf{P}(Rain|Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$false$	$false$	$false$	$false$		
$true$	$true$	$false$	$true$		

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler=true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler{=}true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		
false	*true*	*true*	*true*		

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler{=}true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		
false	*true*	*true*	*true*	1	

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		
false	*true*	*true*	*true*	1	
false	*true*	*false*	*true*		

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		
false	*true*	*true*	*true*	1	
false	*true*	*false*	*true*		1

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain|Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$false$	$false$	$false$	$false$		
$true$	$true$	$false$	$true$		1
$true$	$false$	$false$	$false$		
$false$	$true$	$true$	$true$	1	
$false$	$true$	$false$	$true$		1

$$P(Rain = true|Sprinkler = true) =$$

Rejection Sampling

Key Idea

Throw away samples inconsistent with the evidence

Variables:	*Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query:	$P(Rain \mid Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*false*	*false*	*false*		
true	*true*	*false*	*true*		1
true	*false*	*false*	*false*		
false	*true*	*true*	*true*	1	
false	*true*	*false*	*true*		1

$$P(Rain = true \mid Sprinkler = true) = \frac{1}{1+1+1} = \frac{1}{3}$$

Rejection Sampling Code

```
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

Rejection Sampling Code

```python
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

Rejection Sampling Code

```
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

Rejection Sampling Code

```
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

Rejection Sampling Code

```
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

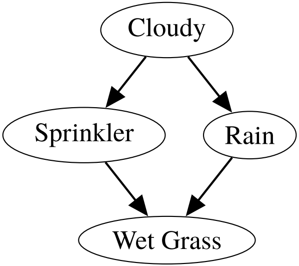
Rejection Sampling Code

```python
def rejection_sampling(query, evidence, bayes_net, samples):
    # generate a bunch of samples, counting query values
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample = prior_sample(bayes_net)
        # if the sample is consistent with the evidence, count it
        if all(sample[variable] == evidence[variable] for variable in evidence):
            counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)

def normalize(counts):
    # divide all counts by the total
    total = sum(counts.values())
    for value in counts:
        counts[value] /= total
    return counts
```

Rejection Sampling Exercise

P(C)
.50



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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

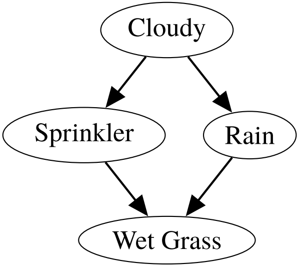
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
0.6	0.4	0.3	0.8
0.7	0.3	0.8	0.6
0.3	0.2	0.7	0.3
0.9	0.2	0.4	0.1
0.8	0.4	0.1	0.9

Rejection Sampling Exercise

P(C)
.50



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	.00

Calculate:

$P(rain|sprinkler)$

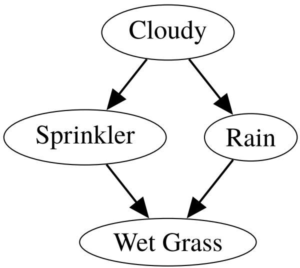
5 samples

$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	0.4	0.3	0.8
0.7	0.3	0.8	0.6
0.3	0.2	0.7	0.3
0.9	0.2	0.4	0.1
0.8	0.4	0.1	0.9

Rejection Sampling Exercise



P(C)
.50

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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

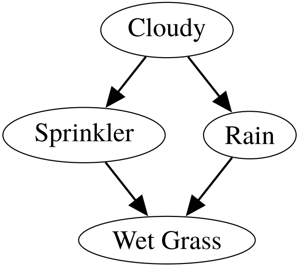
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	T	0.3	0.8
0.7	0.3	0.8	0.6
0.3	0.2	0.7	0.3
0.9	0.2	0.4	0.1
0.8	0.4	0.1	0.9

Rejection Sampling Exercise

P(C)
.50



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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

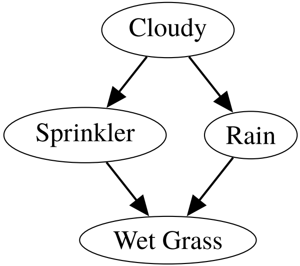
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	*W*
F	*T*	*F*	0.8
0.7	0.3	0.8	0.6
0.3	0.2	0.7	0.3
0.9	0.2	0.4	0.1
0.8	0.4	0.1	0.9

Rejection Sampling Exercise

P(C)
.50



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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

$P(\ldots)$ is *true* if random $< P(\ldots)$

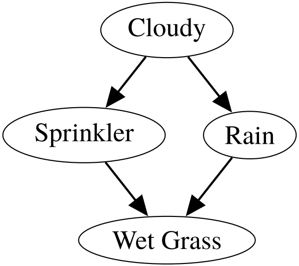
Random numbers:

C	S	R	W
F	T	F	T
0.7	0.3	0.8	0.6
0.3	0.2	0.7	0.3
0.9	0.2	0.4	0.1
0.8	0.4	0.1	0.9

Rejection Sampling Exercise

P(C)
.50

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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

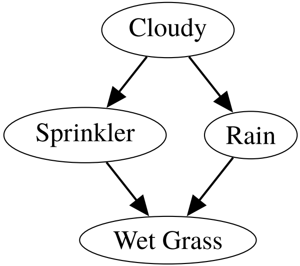
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	T	F	T
F	T	F	T
T	F	T	T
F	T	F	T
F	T	T	T

Rejection Sampling Exercise

P(C)
.50



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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

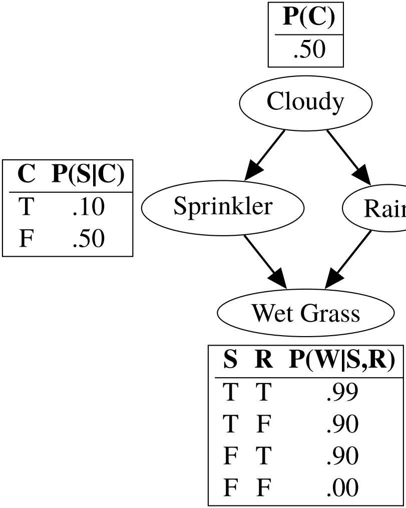
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	T	F	T
F	T	F	T
T	F	T	T
F	T	F	T
F	T	T	T

$P(rain|sprinkler) =$

Rejection Sampling Exercise



Calculate:

$P(rain|sprinkler)$

5 samples

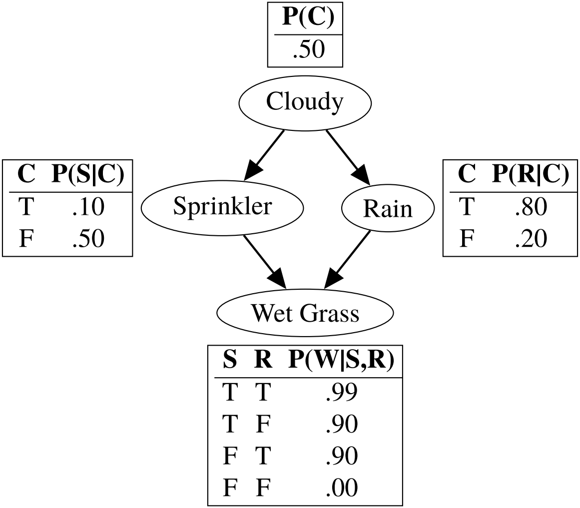
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	T	F	T
F	T	F	T
T	F	T	T
F	T	F	T
F	T	T	T

$P(rain|sprinkler) =$

Rejection Sampling Exercise



Calculate:

$P(rain|sprinkler)$

5 samples

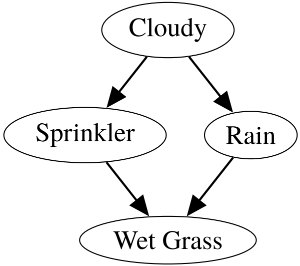
$P(\ldots)$ is *true* if random $< P(\ldots)$

Random numbers:

C	S	R	W
F	T	F	T
F	T	F	T
T	F	T	T
F	T	F	T
F	T	T	T

$P(rain|sprinkler) =$

Rejection Sampling Exercise



P(C)
.50

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	.00

Calculate:

$P(rain|sprinkler)$

5 samples

$P(\dots)$ is *true* if random $< P(\dots)$

Random numbers:

C	S	R	W
F	T	F	T
F	T	F	T
T	F	T	T
F	T	F	T
F	T	T	T

$P(rain|sprinkler) = \frac{1}{4}$

Rejection Sampling

Properties

Given n variables, at most d parents each, s samples drawn and u samples used:

- Time Complexity: $O(nds)$
- Standard deviation of error proportional to $\frac{1}{\sqrt{u}}$ i.e. it approximates the true probability

Problems

- Generates and throws away many samples
- More thrown away for lower probability evidence
- More evidence variables means lower probability

Rejection Sampling

Properties

Given n variables, at most d parents each, s samples drawn and u samples used:

- Time Complexity: $O(nds)$
- Standard deviation of error proportional to $\frac{1}{\sqrt{u}}$
  i.e. it approximates the true probability

Problems

- Generates and throws away many samples
- More thrown away for lower probability evidence
- More evidence variables means lower probability

Rejection Sampling

Properties

Given n variables, at most d parents each, s samples drawn and u samples used:

- Time Complexity: $O(nds)$
- Standard deviation of error proportional to $\frac{1}{\sqrt{u}}$

  i.e. it approximates the true probability

Problems

- Generates and throws away many samples
- More thrown away for lower probability evidence
- More evidence variables means lower probability

Rejection Sampling

Properties

Given n variables, at most d parents each, s samples drawn and u samples used:

- Time Complexity: $O(nds)$
- Standard deviation of error proportional to $\frac{1}{\sqrt{u}}$
  i.e. it approximates the true probability

Problems

- Generates and throws away many samples
- More thrown away for lower probability evidence
- More evidence variables means lower probability

Rejection Sampling

Properties

Given n variables, at most d parents each, s samples drawn and u samples used:

- Time Complexity: $O(nds)$
- Standard deviation of error proportional to $\frac{1}{\sqrt{u}}$
  i.e. it approximates the true probability

Problems

- Generates and throws away many samples
- More thrown away for lower probability evidence
- More evidence variables means lower probability

So Why Not Just Fix The Evidence?

Query: $P(A|b)$
Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

So Why Not Just Fix The Evidence?

Query: $P(A|b)$
Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$P(A|b) = \langle 0.4, 0.6 \rangle$

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$P(A|b) = \langle 0.4, 0.6 \rangle$

Using full joint distribution

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$P(A|b) = \langle 0.4, 0.6 \rangle$

Using full joint distribution

A	B	$P(A, B)$
T	T	

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$P(A|b) = \langle 0.4, 0.6 \rangle$

Using full joint distribution

A	B	$P(A, B)$
T	T	$0.4 \cdot 0.2 = 0.08$

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$P(A|b) = \langle 0.4, 0.6 \rangle$

Using full joint distribution

A	B	$P(A, B)$
T	T	$0.4 \cdot 0.2 = 0.08$
T	F	$0.4 \cdot 0.8 = 0.32$

So Why Not Just Fix The Evidence?

Query: $P(A|b)$

Given: $P(A) = \langle 0.4, 0.6\rangle$, $P(b|A) = \langle 0.2, 0.4\rangle$

Fixing evidence (wrong)

$\langle a, b\rangle$ 40% of the time

$\langle \neg a, b\rangle$ 60% of the time

$P(A|b) = \langle 0.4, 0.6\rangle$

Using full joint distribution

A	B	$P(A, B)$
T	T	$0.4 \cdot 0.2 = 0.08$
T	F	$0.4 \cdot 0.8 = 0.32$
F	T	$0.6 \cdot 0.4 = 0.24$
F	F	$0.6 \cdot 0.6 = 0.36$

So Why Not Just Fix The Evidence?

Query: $P(A|b)$
Given: $P(A) = \langle 0.4, 0.6 \rangle$, $P(b|A) = \langle 0.2, 0.4 \rangle$

Fixing evidence (wrong)

$\langle a, b \rangle$	40% of the time
$\langle \neg a, b \rangle$	60% of the time

$$P(A|b) = \langle 0.4, 0.6 \rangle$$

Using full joint distribution

A	B	$P(A, B)$
T	T	$0.4 \cdot 0.2 = 0.08$
T	F	$0.4 \cdot 0.8 = 0.32$
F	T	$0.6 \cdot 0.4 = 0.24$
F	F	$0.6 \cdot 0.6 = 0.36$

$$\begin{aligned} P(A|b) &= \alpha P(A, b) \\ &= \alpha \langle 0.08, 0.24 \rangle \\ &= \langle 0.25, 0.75 \rangle \end{aligned}$$

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables:	*Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query:	$P(Rain \mid Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain | Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
true	*true*	*true*	*true*		

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables:	*Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query:	$P(Rain \mid Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
true	*true*	*true*	*true*	0.1	

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$		

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	
$false$	$true$	$false$	$true$		

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	
$false$	$true$	$false$	$true$		0.5

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	
$false$	$true$	$false$	$true$		0.5
$true$	$true$	$true$	$true$		

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain | Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	¬*rain*
true	*true*	*true*	*true*	0.1	
true	*true*	*true*	*true*	0.1	
false	*true*	*false*	*true*		0.5
true	*true*	*true*	*true*	0.1	

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X=x|parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain|Sprinkler=true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	
$false$	$true$	$false$	$true$		0.5
$true$	$true$	$true$	$true$	0.1	

$P(Rain=true|Sprinkler=true) =$

Likelihood Weighting

Key Ideas

- Only generate samples consistent with evidence
- Use $P(X = x | parents(X))$ to assign weights

Variables: $Cloudy$, $Sprinkler$, $Rain$, $WetGrass$
Query: $P(Rain | Sprinkler = true)$

$Cloudy$	$Sprinkler$	$Rain$	$WetGrass$	$rain$	$\neg rain$
$true$	$true$	$true$	$true$	0.1	
$true$	$true$	$true$	$true$	0.1	
$false$	$true$	$false$	$true$		0.5
$true$	$true$	$true$	$true$	0.1	

$$P(Rain = true | Sprinkler = true) = \frac{0.1+0.1+0.1}{0.1+0.1+0.1+0.5} = \frac{3}{8}$$

Likelihood Weighting Code

```python
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```python
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```python
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```python
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting Code

```python
def likelihood_weighting(query, evidence, bayes_net, samples):
    # generate samples, adding up weights for each query value
    counts = {False: 0, True: 0}
    for _ in range(samples):
        sample, weight = weighted_sample(bayes_net, evidence)
        counts[sample[query]] += weight
    # normalize the counts and return the probabilities
    return normalize(counts)

def weighted_sample(bayes_net, evidence):
    # generate a value for each variable, from parents to children
    sample, weight = {}, 1.0
    for variable in bayes_net:
        parent_values = [sample[parent] for parent in variable.parents]
        # if the value is given, add it and update the weight
        if variable in evidence:
            sample[variable] = value = evidence[variable]
            weight *= variable.probability_of(value, *parent_values)
        # otherwise, add True or False using the distribution
        else:
            probability = variable.probability_of(True, *parent_values)
            sample[variable] = random.random() < probability
    return sample, weight
```

Likelihood Weighting

Properties

Given n variables, $\leq d$ parents each, and s samples:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used

Problems

- More evidence variables
  $\rightarrow$ each sample has lower probability
- Evidence late in the node ordering
  $\rightarrow$ earlier node selections may not match evidence

Likelihood Weighting

Properties

Given n variables, $\leq d$ parents each, and s samples:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used

Problems

- More evidence variables
  $\rightarrow$ each sample has lower probability
- Evidence late in the node ordering
  $\rightarrow$ earlier node selections may not match evidence

Likelihood Weighting

Properties

Given n variables, $\leq d$ parents each, and s samples:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used

Problems

- More evidence variables
  $\rightarrow$ each sample has lower probability
- Evidence late in the node ordering
  $\rightarrow$ earlier node selections may not match evidence

Likelihood Weighting

Properties

Given n variables, $\leq d$ parents each, and s samples:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used

Problems

- More evidence variables
  $\rightarrow$ each sample has lower probability
- Evidence late in the node ordering
  $\rightarrow$ earlier node selections may not match evidence

Likelihood Weighting

Properties

Given n variables, $\leq d$ parents each, and s samples:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used

Problems

- More evidence variables
  $\rightarrow$ each sample has lower probability
- Evidence late in the node ordering
  $\rightarrow$ earlier node selections may not match evidence

Gibbs Sampling

Gibbs Sampling (Markov chain Monte Carlo)

- Start by randomly assigning values to variables
- Iteratively update values given current assignment
  - Assign new values given "surrounding" distribution

Gibbs Sampling for Bayesian Networks

Define "surrounding" as the Markov Blanket:

a node's parents, children and children's parents

Gibbs Sampling

Gibbs Sampling (Markov chain Monte Carlo)

- Start by randomly assigning values to variables
- Iteratively update values given current assignment
  - Assign new values given “surrounding” distribution

Gibbs Sampling for Bayesian Networks

Define “surrounding” as the Markov Blanket:

a node’s parents, children and children’s parents

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*true*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*true*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*true*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*true*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*false*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*false*	*false*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches “dynamic equilibrium”
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*false*	*true*		

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*false*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*true*	*false*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*true*	*false*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*false*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches “dynamic equilibrium”
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler=true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*true*	*true*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*true*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*true*	*true*		1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
false	*true*	*true*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
false	*true*	*true*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*true*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*true*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*false*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*false*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*

Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	¬*rain*
true	*true*	*false*	*true*	1	1

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*false*	*true*	1	2

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	*¬rain*
true	*true*	*false*	*true*	1	2

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Example

Variables: *Cloudy*, *Sprinkler*, *Rain*, *WetGrass*
Query: $P(Rain|Sprinkler = true)$

Cloudy	*Sprinkler*	*Rain*	*WetGrass*	*rain*	$\neg$*rain*
true	*true*	*false*	*true*	1	2

$$P(Rain = true|Sprinkler = true) = \frac{1}{1+2} = \frac{1}{3}$$

Why Gibbs Sampling Works

- Over time, reaches "dynamic equilibrium"
- Time spent in each state proportional to its probability

Gibbs Sampling Code

```
def gibbs_sampling(query, evidence, bayes_net, samples):
    # initialize the sample with random values for non-evidence
    sample = {}
    for variable in bayes_net:
        if variable in evidence:
            sample[variable] = evidence[variable]
        else:
            sample[variable] = random.random() < 0.5
    # generate samples by changing non-evidence values
    counts = {False:0, True:0}
    non_evidence = [var for var in bayes_net if var not in evidence]
    for _ in range(samples):
        for variable in non_evidence:
            # get the prob distribution given the markov blanket
            probability = markov_blanket_probability_of(variable, sample)
            # select a new value according to that distribution
            sample[variable] = random.random() < probability
        # increment the count for the current query value
        counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)
```

Gibbs Sampling Code

```python
def gibbs_sampling(query, evidence, bayes_net, samples):
    # initialize the sample with random values for non-evidence
    sample = {}
    for variable in bayes_net:
        if variable in evidence:
            sample[variable] = evidence[variable]
        else:
            sample[variable] = random.random() < 0.5
    # generate samples by changing non-evidence values
    counts = {False:0, True:0}
    non_evidence = [var for var in bayes_net if var not in evidence]
    for _ in range(samples):
        for variable in non_evidence:
            # get the prob distribution given the markov blanket
            probability = markov_blanket_probability_of(variable, sample)
            # select a new value according to that distribution
            sample[variable] = random.random() < probability
        # increment the count for the current query value
        counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)
```

Gibbs Sampling Code

```python
def gibbs_sampling(query, evidence, bayes_net, samples):
    # initialize the sample with random values for non-evidence
    sample = {}
    for variable in bayes_net:
        if variable in evidence:
            sample[variable] = evidence[variable]
        else:
            sample[variable] = random.random() < 0.5
    # generate samples by changing non-evidence values
    counts = {False:0, True:0}
    non_evidence = [var for var in bayes_net if var not in evidence]
    for _ in range(samples):
        for variable in non_evidence:
            # get the prob distribution given the markov blanket
            probability = markov_blanket_probability_of(variable, sample)
            # select a new value according to that distribution
            sample[variable] = random.random() < probability
        # increment the count for the current query value
        counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)
```

Gibbs Sampling Code

```python
def gibbs_sampling(query, evidence, bayes_net, samples):
    # initialize the sample with random values for non-evidence
    sample = {}
    for variable in bayes_net:
        if variable in evidence:
            sample[variable] = evidence[variable]
        else:
            sample[variable] = random.random() < 0.5
    # generate samples by changing non-evidence values
    counts = {False:0, True:0}
    non_evidence = [var for var in bayes_net if var not in evidence]
    for _ in range(samples):
        for variable in non_evidence:
            # get the prob distribution given the markov blanket
            probability = markov_blanket_probability_of(variable, sample)
            # select a new value according to that distribution
            sample[variable] = random.random() < probability
        # increment the count for the current query value
        counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)
```

Gibbs Sampling Code

```
def gibbs_sampling(query, evidence, bayes_net, samples):
    # initialize the sample with random values for non-evidence
    sample = {}
    for variable in bayes_net:
        if variable in evidence:
            sample[variable] = evidence[variable]
        else:
            sample[variable] = random.random() < 0.5
    # generate samples by changing non-evidence values
    counts = {False:0, True:0}
    non_evidence = [var for var in bayes_net if var not in evidence]
    for _ in range(samples):
        for variable in non_evidence:
            # get the prob distribution given the markov blanket
            probability = markov_blanket_probability_of(variable, sample)
            # select a new value according to that distribution
            sample[variable] = random.random() < probability
        # increment the count for the current query value
        counts[sample[query]] += 1
    # normalize the counts and return the probabilities
    return normalize(counts)
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Markov Blanket Code

Markov Blanket Probability

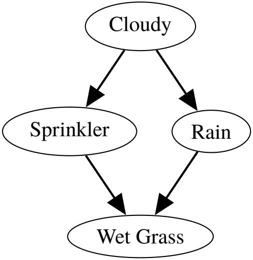
$$P(x|mb(X)) = \alpha P(x|parents(X)) \prod_{Y \in Children(X)} P(y|parents(Y))$$

```python
def markov_blanket_probability_of(variable, sample):
    # get the probabilities for each value of the variable
    counts = {}
    for value in [True, False]:
        # change the variable's value in the sample
        sample[variable] = value
        # add the probability of the variable given its parents
        parent_values = [sample[parent] for parent in variable.parents]
        counts[value] = variable.probability_of(value, *parent_values)
        # times the probabilities of the children given their parents
        for child in variable.children:
            child_value = sample[child]
            parent_values = [sample[parent] for parent in child.parents]
            counts[value] *= child.probability_of(child_value, *parent_values)
    # normalize the counts and return the probability of True
    return normalize(counts)[True]
```

Gibbs Sampling Exercise

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

P(C)
.50



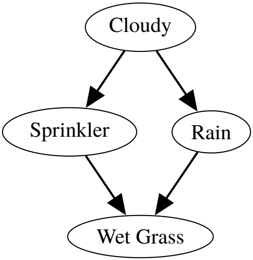
C	P(S|C)
T	.10
F	.50

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	.00

Gibbs Sampling Exercise

P(C)
.50



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	.00

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

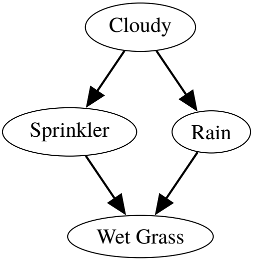
Updating C:

Gibbs Sampling Exercise

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:
$P(c|mb(C))$

P(C)
.50

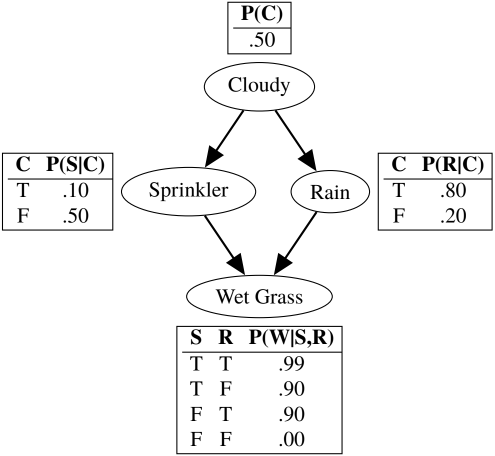


C	P(S|C)
T	.10
F	.50

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	.00

Gibbs Sampling Exercise



P(C)
.50

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	.00

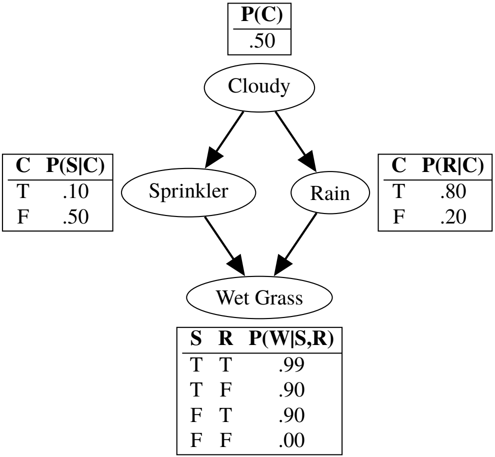
Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$$P(c|mb(C))$$
$$= \alpha P(c) P(s|c) P(\neg r|c)$$

Gibbs Sampling Exercise



P(C)
.50

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	.00

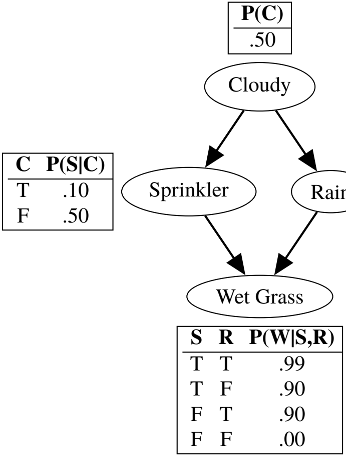
Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$$P(c|mb(C))$$
$$= \alpha P(c)P(s|c)P(\neg r|c)$$
$$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$$

Gibbs Sampling Exercise



P(C)
.50

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	.00

Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

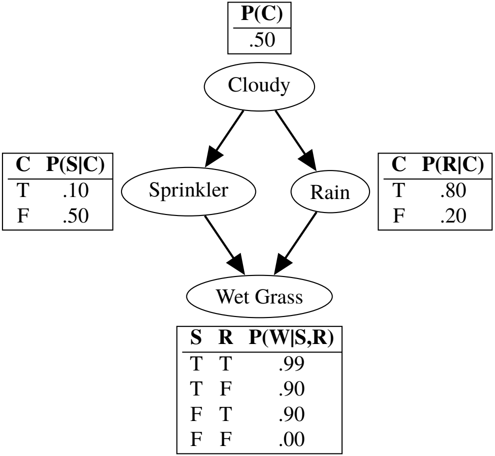
$P(c|mb(C))$

$= \alpha P(c)P(s|c)P(\neg r|c)$

$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$

$P(\neg c|mb(C))$

Gibbs Sampling Exercise



P(C)
.50

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

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

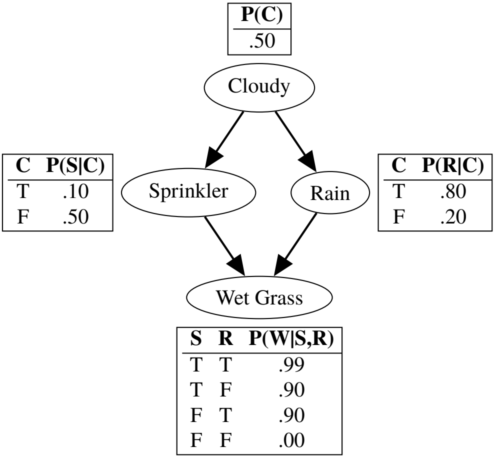
S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.00

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$
$= \alpha P(c)P(s|c)P(\neg r|c)$
$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$
$P(\neg c|mb(C))$
$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$

Gibbs Sampling Exercise



P(C)
.50

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

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

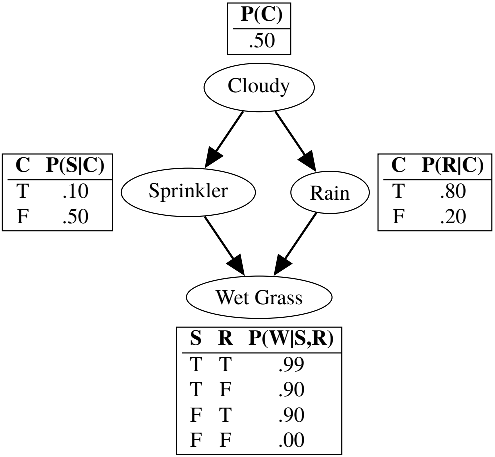
S	R	P(W|S,R)
T	T	.99
T	F	.90
F	T	.90
F	F	.00

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$
$= \alpha P(c)P(s|c)P(\neg r|c)$
$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$
$P(\neg c|mb(C))$
$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$
$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$

Gibbs Sampling Exercise



P(C)
.50

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	.00

Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$

$= \alpha P(c)P(s|c)P(\neg r|c)$

$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$

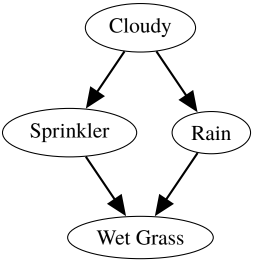
$P(\neg c|mb(C))$

$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$

$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$

$\mathbf{P}(C|mb(C)) = \langle 0.048, 0.952 \rangle$

Gibbs Sampling Exercise



P(C)
.50

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	.00

Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$

$= \alpha P(c)P(s|c)P(\neg r|c)$

$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$

$P(\neg c|mb(C))$

$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$

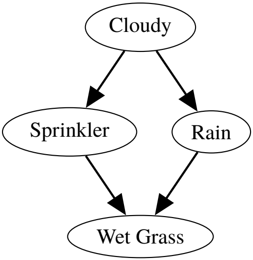
$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$

$\mathbf{P}(C|mb(C)) = \langle 0.048, 0.952 \rangle$

Random 0.03

Gibbs Sampling Exercise

P(C)
.50



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	.00

Query: $P(r|s)$

Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$

$= \alpha P(c)P(s|c)P(\neg r|c)$

$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$

$P(\neg c|mb(C))$

$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$

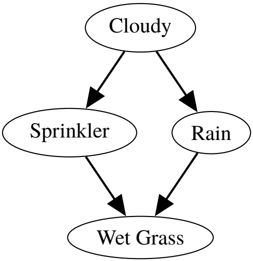
$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$

$\mathbf{P}(C|mb(C)) = \langle 0.048, 0.952 \rangle$

Random 0.03: $\langle c, s, \neg r, w \rangle$

Gibbs Sampling Exercise

P(C)
.50



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	.00

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

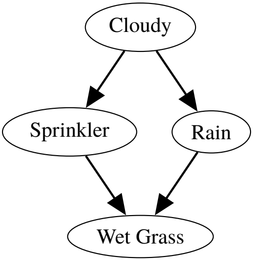
$P(c|mb(C))$
$= \alpha P(c)P(s|c)P(\neg r|c)$
$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$
$P(\neg c|mb(C))$
$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$
$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$
$\mathbf{P}(C|mb(C)) = \langle 0.048, 0.952 \rangle$

Random 0.03: $\langle c, s, \neg r, w \rangle$

Update	R	W	C
Random	0.48	0.63	0.83

Gibbs Sampling Exercise

P(C)
.50



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	.00

Query: $P(r|s)$
Initial Sample: $\langle \neg c, s, \neg r, w \rangle$

Updating C:

$P(c|mb(C))$
$= \alpha P(c)P(s|c)P(\neg r|c)$
$= \alpha \cdot 0.5 \cdot 0.1 \cdot 0.2 = 0.01\alpha$
$P(\neg c|mb(C))$
$= \alpha P(\neg c)P(s|\neg c)P(\neg r|\neg c)$
$= \alpha \cdot 0.5 \cdot 0.5 \cdot 0.8 = 0.2\alpha$
$\mathbf{P}(C|mb(C)) = \langle 0.048, 0.952 \rangle$

Random 0.03: $\langle c, s, \neg r, w \rangle$

Update	R	W	C
Random	0.48	0.63	0.83
Result	$\langle \neg c, s, r, w \rangle$		

Gibbs Sampling Exercise

previous sample $\rightarrow \langle c, s, \neg r, w\rangle$

$$
\begin{aligned}
P(r|mb(R)) &= \alpha P(r|c)P(w|s,r) = \alpha \cdot 0.8 \cdot 0.99 = \alpha \cdot 0.792 \\
P(\neg r|mb(R)) &= \alpha P(\neg r|c)P(w|s,\neg r) = \alpha \cdot 0.2 \cdot 0.9 = \alpha \cdot 0.18 \\
P(R|mb(R)) &= \alpha\langle 0.792, 0.18\rangle = \langle 0.815, 0.185\rangle \\
0.48 &< 0.815 \rightarrow \langle c, s, r, w\rangle
\end{aligned}
$$

$$
\begin{aligned}
P(w|mb(W)) &= \alpha P(w|s,r) = \alpha \cdot 0.99 \\
P(\neg w|mb(W)) &= \alpha P(\neg w|s,r) = \alpha \cdot 0.01 \\
P(W|mb(W)) &= \alpha\langle 0.99, 0.01\rangle = \langle 0.99, 0.01\rangle \\
0.63 &< 0.99 \rightarrow \langle c, s, r, w\rangle
\end{aligned}
$$

$$
\begin{aligned}
P(c|mb(C)) &= \alpha P(c)P(s|c)P(r|c) = \alpha \cdot 0.5 \cdot 0.1 \cdot 0.8 = \alpha \cdot 0.04 \\
P(\neg c|mb(C)) &= \alpha P(\neg c)P(s|\neg c)P(r|\neg c) = \alpha \cdot 0.5 \cdot 0.5 \cdot 0.2 = \alpha \cdot 0.05 \\
P(C|mb(C)) &= \alpha\langle 0.04, 0.05\rangle = \langle 0.444, 0.555\rangle \\
0.83 &\not< 0.444 \rightarrow \langle \neg c, s, r, w\rangle
\end{aligned}
$$

Gibbs Sampling

Properties

Given n variables, s samples, and $\leq d$ nodes $\in mb(X)$:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used
- Performs well in practice

Problems

- Difficult to tell when convergence is achieved
- Performs worse when Markov blankets are large

Gibbs Sampling

Properties

Given n variables, s samples, and $\leq d$ nodes $\in mb(X)$:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used
- Performs well in practice

Problems

- Difficult to tell when convergence is achieved
- Performs worse when Markov blankets are large

Gibbs Sampling

Properties

Given n variables, s samples, and $\leq d$ nodes $\in mb(X)$:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used
- Performs well in practice

Problems

- Difficult to tell when convergence is achieved
- Performs worse when Markov blankets are large

Gibbs Sampling

Properties

Given n variables, s samples, and $\leq d$ nodes $\in mb(X)$:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used
- Performs well in practice

Problems

- Difficult to tell when convergence is achieved
- Performs worse when Markov blankets are large

Gibbs Sampling

Properties

Given n variables, s samples, and $\leq d$ nodes $\in mb(X)$:

- Time Complexity: $O(nds)$
- Unlike rejection sampling, all samples are used
- Performs well in practice

Problems

- Difficult to tell when convergence is achieved
- Performs worse when Markov blankets are large

Key Ideas

Bayesian Networks

- Variables linked by conditional independence
- Put causes on top, add direct effects below
- $P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | parents(X_i))$

Inference Methods

- Exact inference via factors
- Rejection sampling requires many samples
- Likelihood weighting poor with a lot of evidence
- Gibbs Sampling updates based on Markov blanket