



Principles of Machine Learning

Probabilistic Graphical Models

Vasant Honavar

Artificial Intelligence Research Laboratory
College of Information Sciences and Technology
Bioinformatics and Genomics Graduate Program
The Huck Institutes of the Life Sciences
Pennsylvania State University

vhonavar@ist.psu.edu

<http://vhonavar.ist.psu.edu>

<http://faculty.ist.psu.edu/vhonavar>

Probabilistic Graphical Models

- The assumption that variables are independent (e.g., Naïve Bayes assumption that the variables are independent given the class) can be too restrictive
- But representing the joint distributions is intractable without some independence assumptions
- Probabilistic graphical models e.g., Bayes networks, explicitly model conditional independence among subsets of variables to yield a graphical representation of probability distributions that admit such independence

Bayesian network

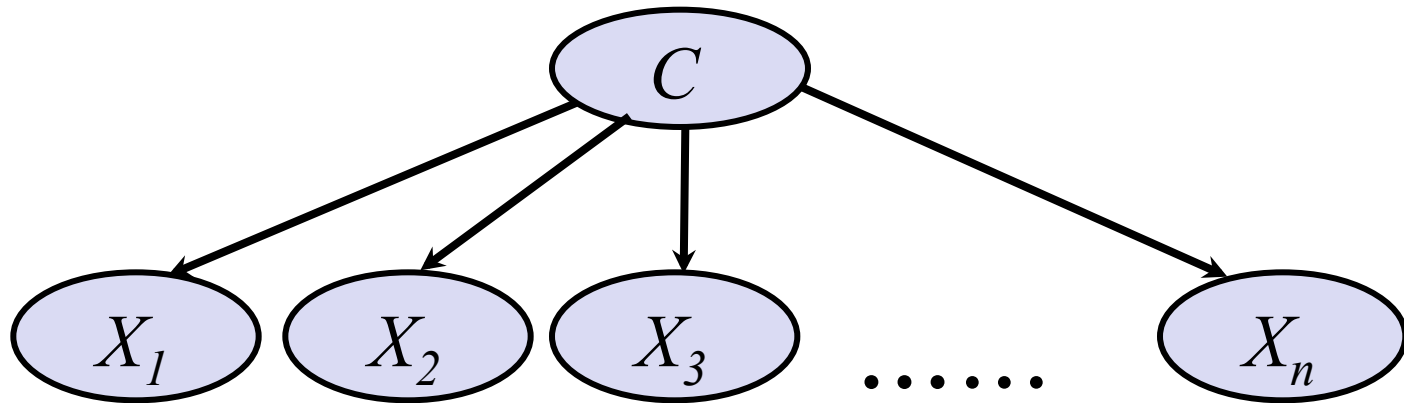
- Bayesian network is a directed acyclic graph (DAG) in which the nodes represent random variables
- Each node is annotated with a probability distribution $P(X_i / Parents(X_i))$ representing the dependency of that node on its parents in the DAG
- Each node is asserted to be conditionally independent of its non-descendants, given its immediate predecessors
- Arcs represent direct dependencies

Conditional Independence

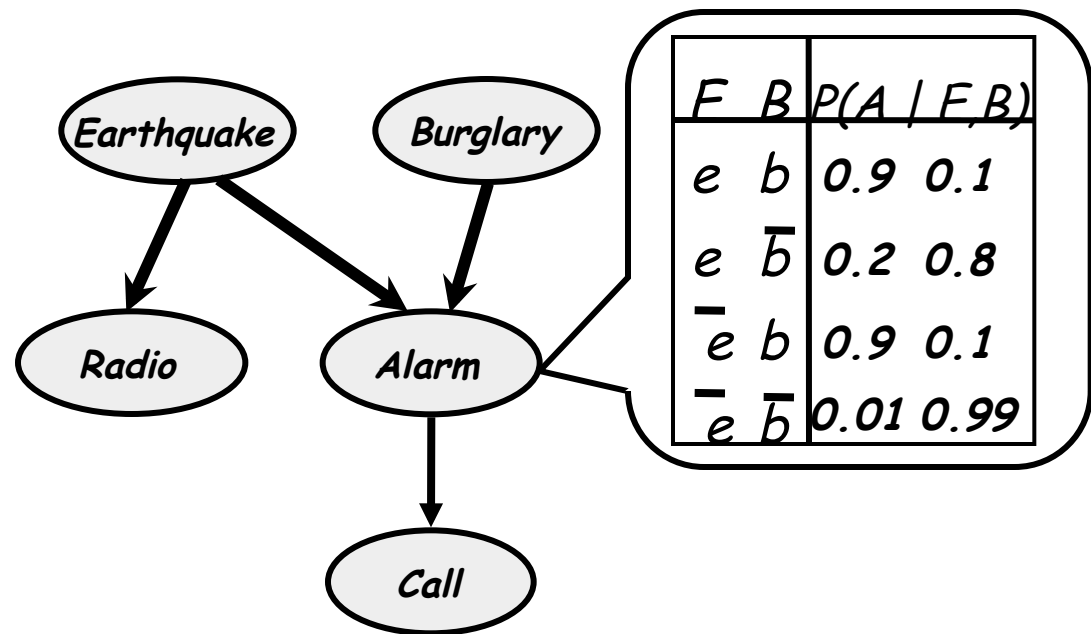
- X is **conditionally independent** of Y given Z if the probability distribution governing X is independent of the value of Y given the value of Z :
- $P(X | Y, Z) = P(X | Z)$ that is,

$$(\forall x_i, y_j, z_k) P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

Naïve Bayes



Bayesian Networks



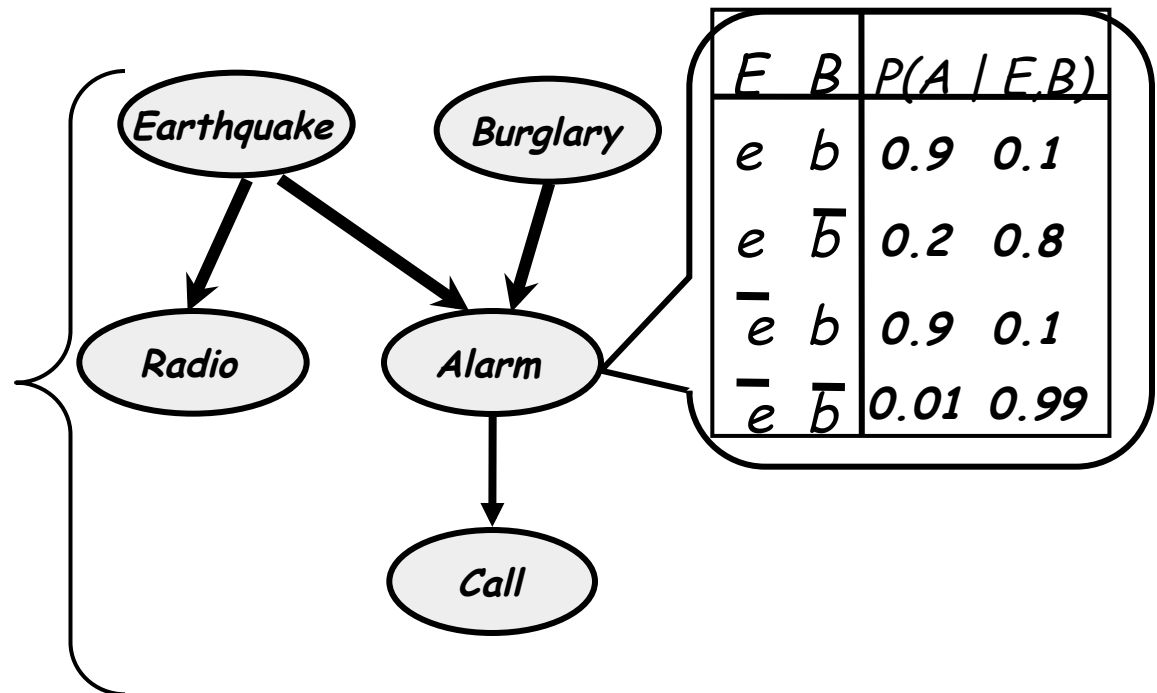
Efficient factorized representation of probability distributions via conditional independence

Bayesian Networks

- Qualitative part**
 statistical independence statements represented in the form of a directed acyclic graph (DAG)
 - Nodes - random variables
 - Edges – direct influence

Quantitative part

Conditional probability distributions – one for each random variable conditioned on its parents

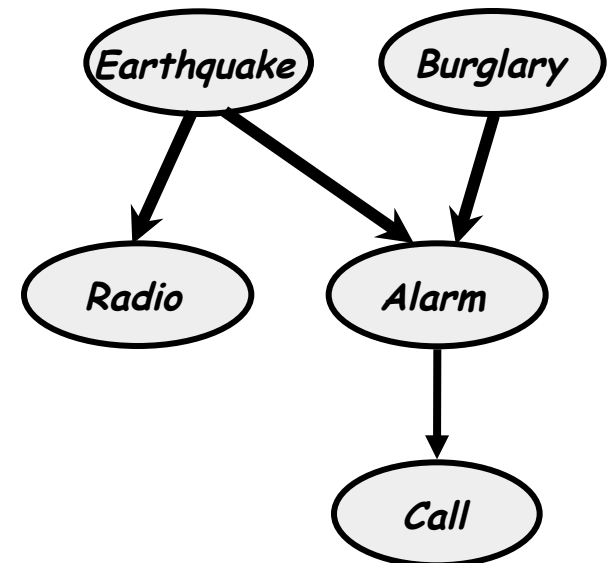


Qualitative part

- Nodes are independent of non-descendants given their parents

d-separation:

- a graph theoretic criterion for reading independence statements
- can be computed in linear time (in the number of edges)



Directed graphs and joint probabilities

- Let $\{X_1, X_2, \dots, X_n\}$ be a set of random variables
- Let $\text{parents}(X_i)$ be the set of parents of X_i
- Associate a vertex in the directed a-cyclic graph with a random variable and a function of the form $f_i(x_i, x_{\pi_i})$

- Then
$$p(x_1 \dots x_n) = \prod_{i=1}^n f_i(x_i, x_{\pi_i})$$

What independences does a Bayes Net model?

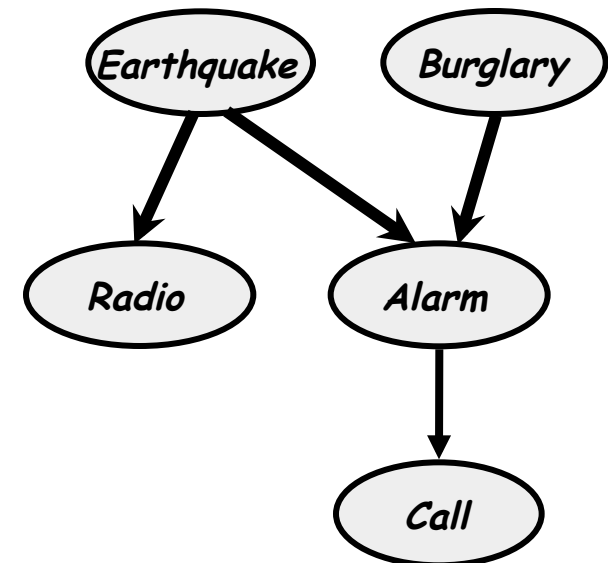
- In order for a Bayesian network to model a probability distribution, the following must be true by definition:
- Each variable is conditionally independent of all its non-descendants in the graph given the value of all its parents.

This implies

$$P(X_1 \dots X_n) = \prod_{i=1}^n P(X_i \mid \text{parents}(X_i))$$

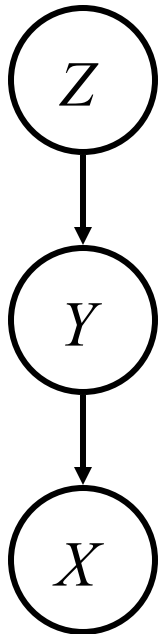
$$P(E, B, R, A, C) = P(E)P(B)P(R \mid E)P(A \mid E, B)P(C \mid A)$$

But what else does it imply?



What Independences does a Bayes Network model?

Example:



Given Y , does learning the value of Z tell us nothing new about X ?

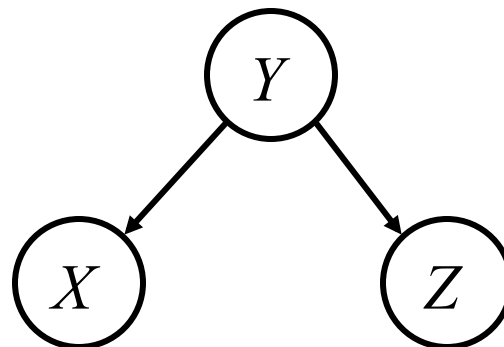
i.e., is $P(X|Y, Z)$ equal to $P(X | Y)$?

Yes. Since we know the value of all of X 's parents (namely, Y), and Z is not a descendant of X , X is conditionally independent of Z .

Also, since independence is symmetric, $P(Z|Y, X) = P(Z|Y)$.

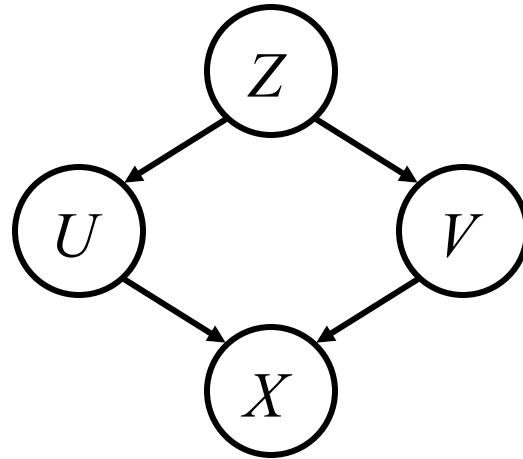
What Independences does a Bayes Network model?

- Let $I(X,Y,Z)$ represent X and Z being conditionally independent given Y .



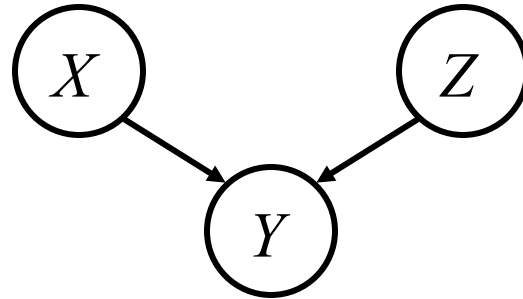
- $I(X,Y,Z)$? Yes, just as in previous example: All X 's parents given, and Z is not a descendant.

What Independences does a Bayes Network model?



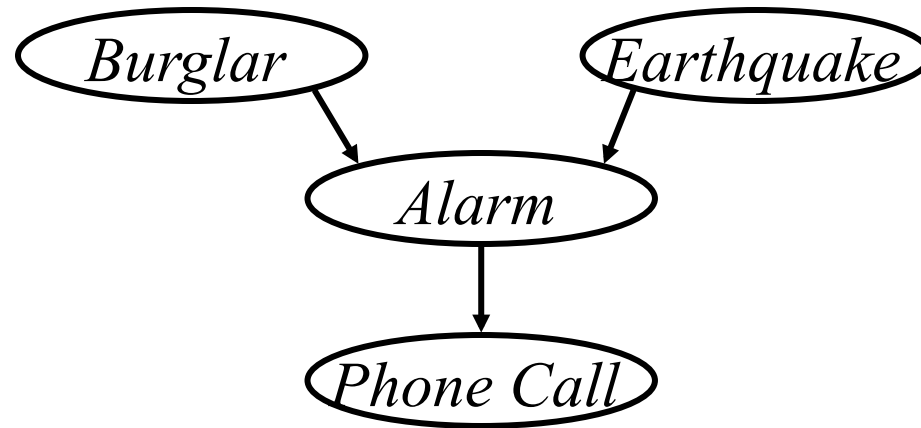
- $I(X, \{U\}, Z)$? No.
- $I(X, \{U, V\}, Z)$? Yes.

Dependency induced by V-structures

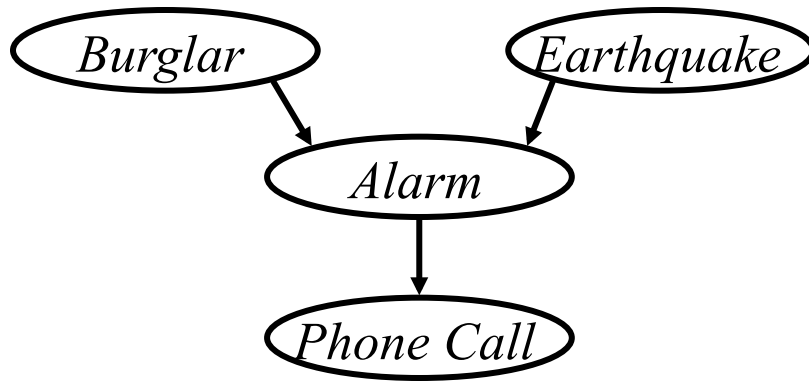


- X has no parents, so we know all its parents' values trivially
- Z is not a descendant of X
- So, $I(X, \{\}, Z)$, even though there is a undirected path from X to Z through an unknown variable Y .
- What if we do know the value of Y ? Or one of its descendants?

The Burglar Alarm example



- Your house has a twitchy burglar alarm that is also sometimes triggered by earthquakes.
- Earth arguably doesn't care whether your house is currently being burgled
- While you are on vacation, one of your neighbors calls and tells you your home's burglar alarm is ringing.



- But now suppose you learn that there was a medium-sized earthquake in your neighborhood. ...Probably not a burglar after all.
- Earthquake “explains away” the hypothetical burglar.
- But then it must NOT be the case that $I(\text{Burglar}, \{\text{Phone Call}\}, \text{Earthquake})$, even though $I(\text{Burglar}, \{\}, \text{Earthquake})$!

d-separation

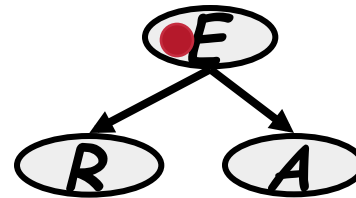
- Fortunately, there is a relatively simple algorithm for determining whether two variables in a Bayesian network are conditionally independent given some other variables:
 - *d-separation*.
- Two variables are independent if all paths between them are blocked by evidence
- Three cases:
 - Common cause
 - Intermediate cause
 - Common Effect

d-separation

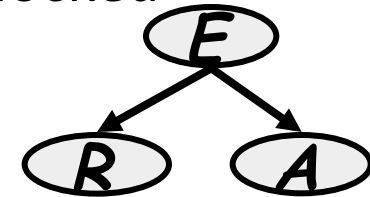
- Two variables are independent if all paths between them are blocked by evidence
- Three cases:
 - Common cause
 - Intermediate cause
 - Common Effect

Evidence may be transmitted through a diverging connection unless it is instantiated.

Blocked



Unblocked

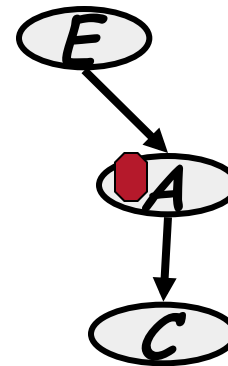


- If we do not know whether an earthquake occurred, then radio announcement can influence our belief about the alarm having gone off.
- If we know that earthquake occurred, then radio announcement gives no information about the alarm

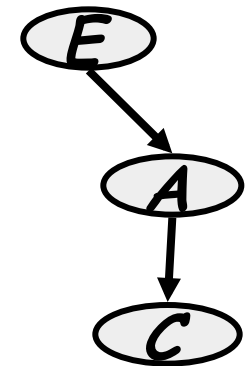
d-separation

- Common cause
- Intermediate cause
- Common Effect

Blocked



Unblocked

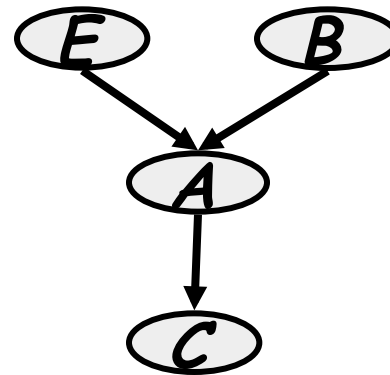


Evidence may be transmitted through a serial connection unless it is blocked

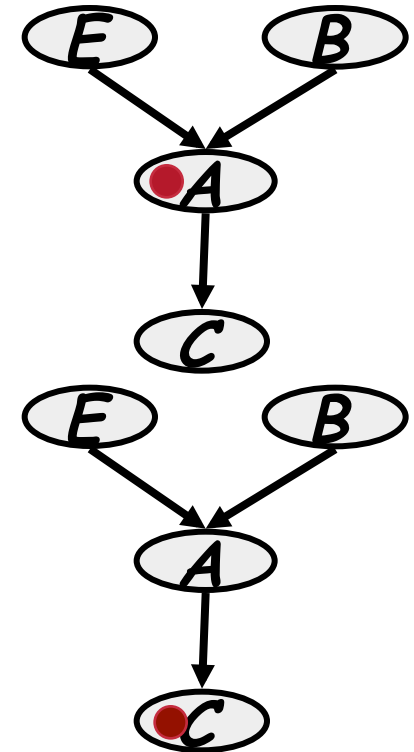
d-separation

- Common cause
- Intermediate cause
- Common Effect

Blocked



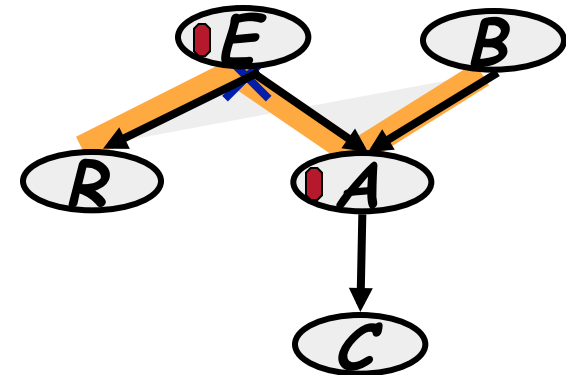
Unblocked



Evidence may be transmitted through a converging connection only if either the variable or one of its descendants has received evidence

Example

- $I(X, Y, Z)$ denotes X and Z are independent given Y
 - Dot indicates evidence available
 - Surely $I(R, \{E, A\} B)$
 - Possibly $\neg I(R, A, B)$
 - Possibly $\neg I(R, B, C)$
 - Surely $I(R, B)$



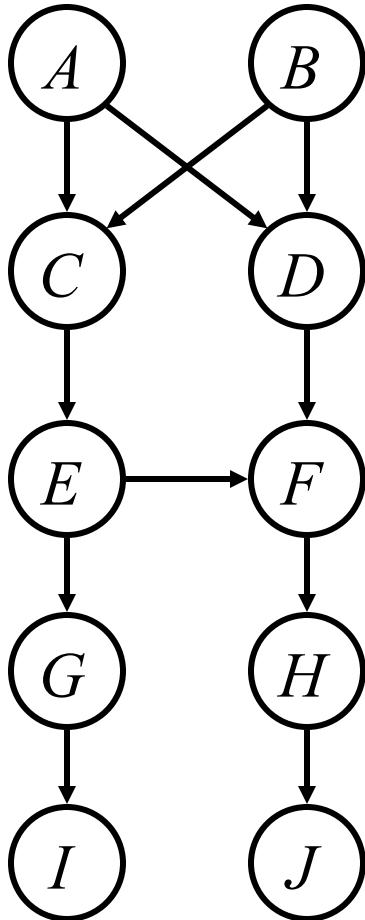
d-separation

- **Definition:** X and Z are *d-separated* by a set of evidence variables E iff every undirected path from X to Z is “blocked” by evidence E

d-separation

- Theorem [Verma & Pearl, 1998]: If a set of evidence variables E d -separates X and Z in a Bayesian network's graph, then $I(X, E, Z)$.
- d -separation can be computed in linear time using a depth-first search like algorithm.
- We now have a fast algorithm for automatically inferring whether finding out about the value of one variable might give us any additional hints about some other variable, given what we already know.
- d -separation of X and Z by E is sufficient for asserting $I(X, E, Z)$, but not necessary.
 - Variables may actually be independent when they are not d -separated, depending on the actual probabilities involved

d-separation



$I(C, \{\}, D)?$

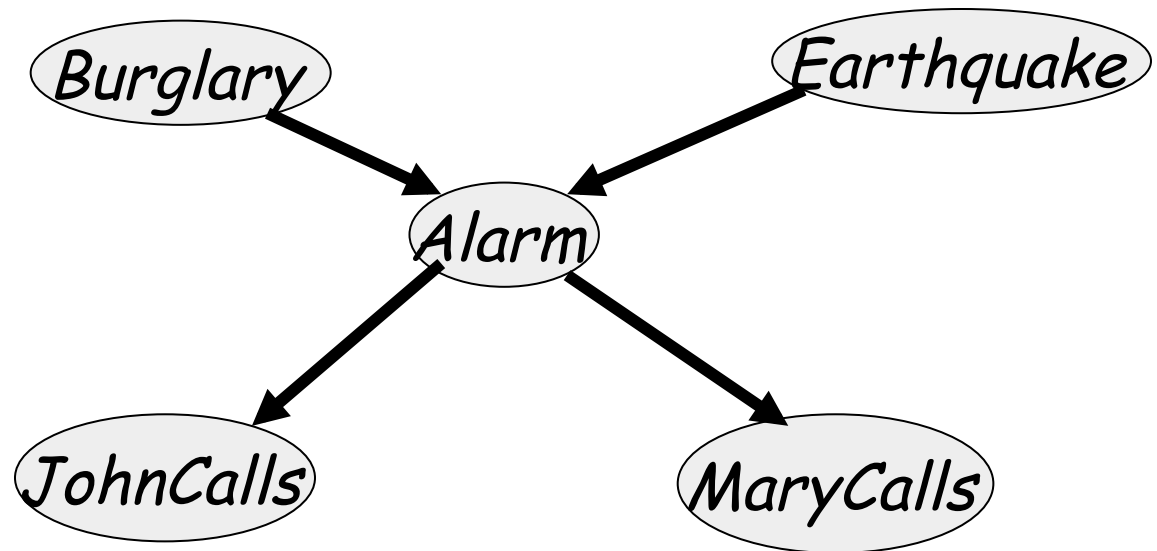
$I(C, \{A\}, D)?$

$I(C, \{A, B\}, D)?$

$I(C, \{A, B, J\}, D)?$

Markov Blanket

- A node is conditionally independent of all other nodes in the network given its parents, children, and children's parents -



Burglary is independent of John Calls and Mary Calls given Alarm and Earth Quake

Bayesian Networks: Summary

- Bayesian networks offer an efficient representation of probability distributions
- Efficient:
 - Local models
 - Independence (d -separation)
- Effective: Algorithms take advantage of structure to
 - Compute posterior probabilities
 - Compute most probable instantiation
 - Decision making

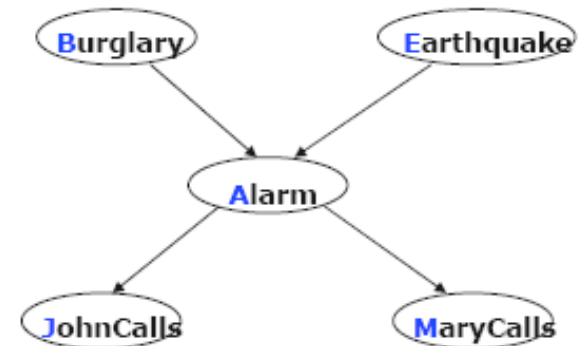
Inference in Bayesian network

Bad news:

- – Exact inference problem in BNs is NP-hard (Cooper)
- – Approximate inference is NP-hard (Dagum, Luby)

In practice, things are not so bad

- Exact inference
 - Inference in Simple Chains
 - Variable elimination
 - Clustering / join tree algorithms
- Approximate inference
 - Stochastic simulation / sampling methods
 - Markov chain Monte Carlo methods
 - Mean field theory



Computing joint probability distributions using a Bayesian network

- Any entry in the joint probability distribution can be calculated from the Bayesian network.
- We're just using the chain rule and conditional independence.

$$\begin{aligned}
 P(J, M, A, \neg B, \neg E) &= P(J \mid M, A, \neg B, \neg E)P(M, A, \neg B, \neg E) \\
 &= P(J \mid A)P(M \mid A, \neg B, \neg E)P(A, \neg B, \neg E) \\
 &= P(J \mid A)P(M \mid A)P(A \mid \neg B, \neg E)P(\neg B, \neg E) \\
 &= P(J \mid A)P(M \mid A)P(A \mid \neg B, \neg E)P(\neg B)P(\neg E)
 \end{aligned}$$

Computing joint probabilities

General formula:

$$P(X_1, \dots, X_n) = P(X_1) \prod_{i=2}^n P(X_i \mid \text{Parents}(X_i))$$

- Joint distribution can be used to answer any query about the domain.
- Bayesian network represents the joint distribution
- Any query about the domain can be answered using a BN
- Tradeoff: A BN can be much more concise, but you need to calculate, rather than look up in a table, probabilities from the joint distribution

Inference in Bayesian Networks

- Bayesian networks are a compact encoding of the full joint probability distribution over N variables that makes conditional independence assumptions between these variables explicit.
- We can use Bayesian networks to compute any probability of interest over the given variables.
- Now we look at Inference in more detail

Inference in Bayesian Networks

Find $P(Q=q|E=e)$

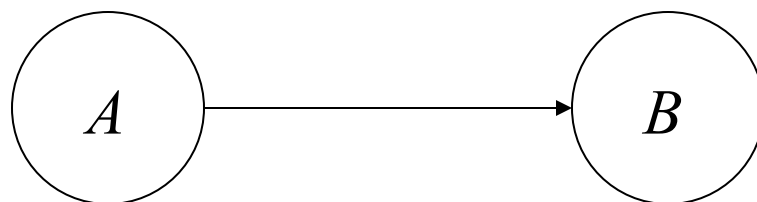
- Q the query variable(s)
- E set of evidence variables

$$P(q|e) = P(q,e) / P(e)$$

X_1, \dots, X_n are network variables except Q, E

$$P(q,e) = \sum_{x_1, x_2, \dots, x_n} (q, e, X_1, X_2, \dots, X_n)$$

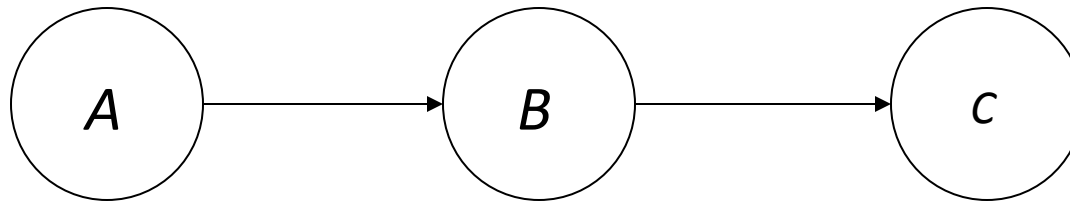
Basic Inference



$$P(b) = ?$$

$$P(b) = \sum_a P(a, b) = \sum_a P(b|a) P(a)$$

Basic Inference

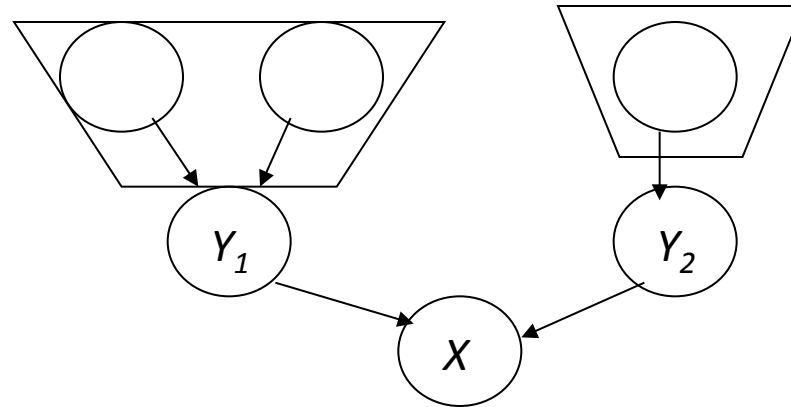


$$P(b) = \sum_a P(a, b) = \sum_a P(b | a) P(a)$$

$$P(c) = \sum_b P(c | b) P(b)$$

$$\begin{aligned}
 P(c) &= \sum_{a,b} P(a, b, c) = \sum_{a,b} P(c | b, a) P(b | a) P(a) \\
 &= \sum_{a,b} P(c | b) P(b | a) P(a) \\
 &= \sum_{a,b} P(c | b) P(b)
 \end{aligned}$$

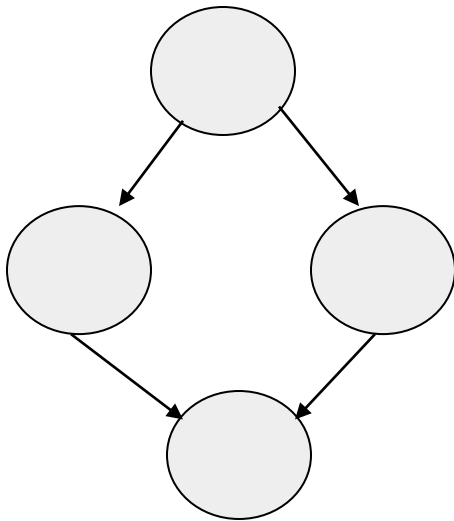
Inference in trees



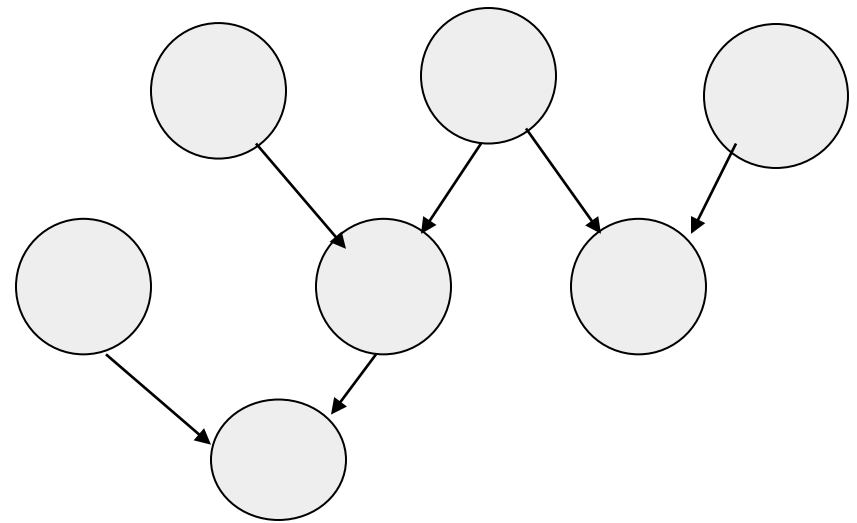
$$P(X) = \sum_{y_1, y_2} P(X, Y_1, Y_2) = \sum_{y_1, y_2} P(X | Y_1, Y_2) P(Y_1, Y_2) = \sum_{y_1, y_2} P(X | Y_1, Y_2) P(Y_1) P(Y_2)$$

Polytrees

- A network is *singly connected* (a polytree) if it contains no undirected loops.



Not a polytree



Polytree

Inference in polytrees

- **Theorem:** Inference in polytrees can be performed in time that is polynomial in the number of variables.
- **Main idea:** in variable elimination, need only maintain distributions over single nodes.

Inference with Bayesian Networks

- Inference in polytrees can be performed efficiently
- Inference with DAG is NP-Hard
 - Proof by reduction of SAT to Bayesian network inference

Approaches to inference

- Exact inference
 - Inference in Simple Chains
 - Variable elimination
 - Clustering / join tree algorithms
- Approximate inference
 - Stochastic simulation / sampling methods
 - Markov chain Monte Carlo methods
 - Mean field theory

Variable Elimination

- General idea:
- Write query in the form

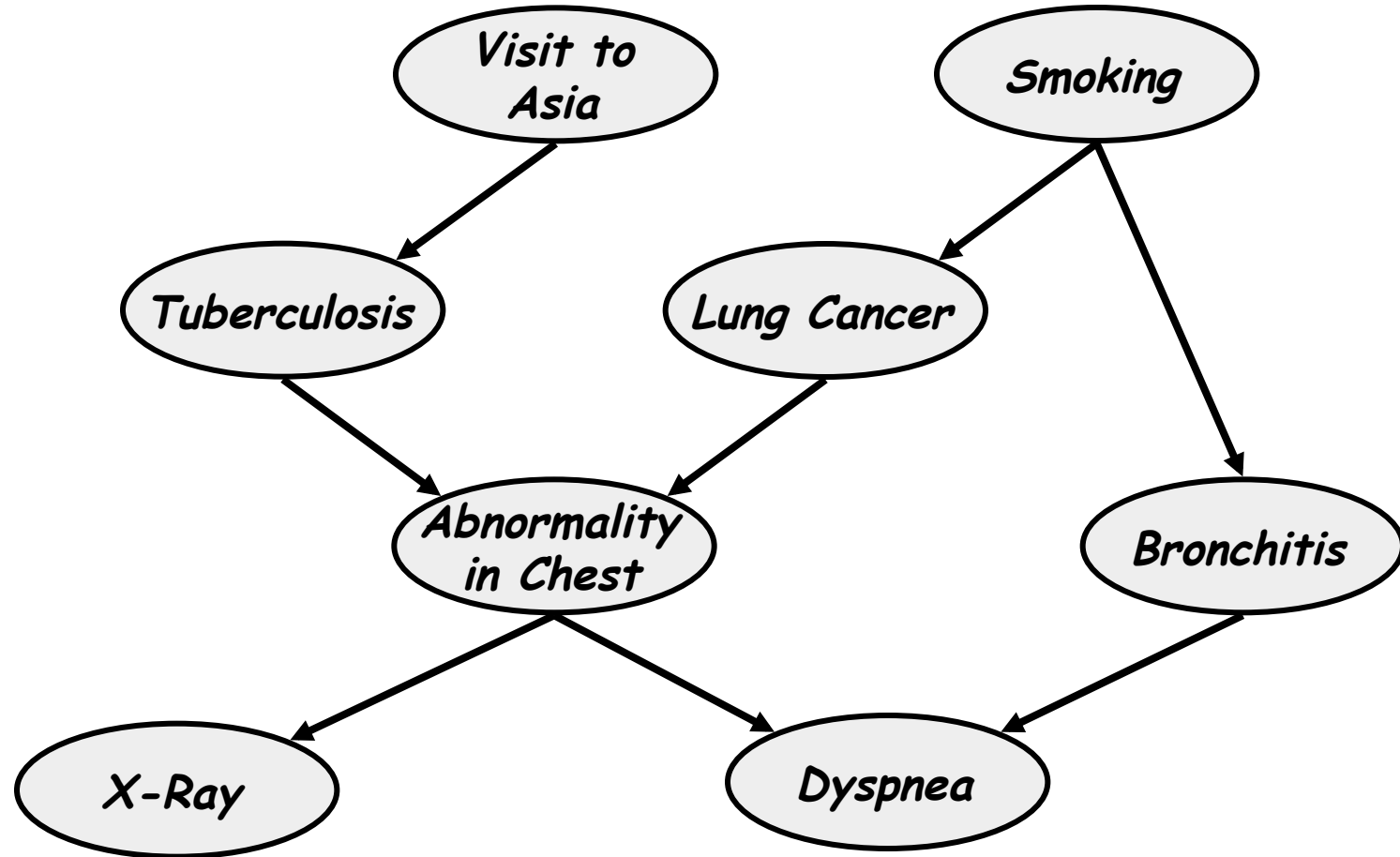
$$P(X_n, \mathbf{e}) = \sum_{x_k} \cdots \sum_{x_3} \sum_{x_2} \prod_i P(x_i | pa_i)$$

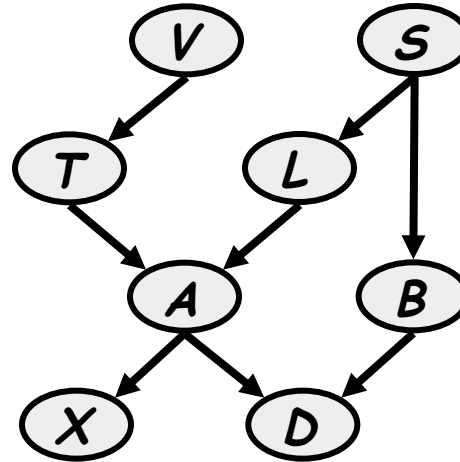
- Iteratively
 - Move all irrelevant terms outside of innermost sum
 - Perform innermost sum, getting a new term
 - Insert the new term into the product

Variable Elimination

- A factor over X is a function from $Domain(X)$ to numbers in the interval $[0,1]$
- A conditional probability table is a factor
- A joint distribution is a factor
- Bayesian network inference
 - Factors are multiplied to generate new ones
 - Variables in factors are summed out (marginalization)
 - A variable can be summed out as soon as all the factors in which the variable appears have been multiplied

A More Complex Example

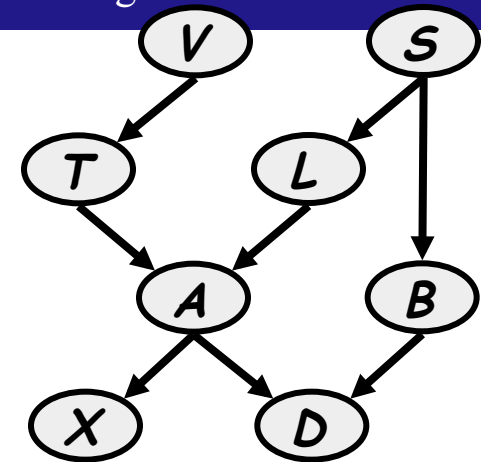




- We want to compute $P(d)$
- Need to eliminate: v, s, x, t, l, a, b

$$P(v) P(s) P(t | v) P(l | s) P(b | s) P(a | t, l) P(x | a) P(d | a, b)$$

- We want to compute $P(d)$
- Need to eliminate: v, s, x, t, l, a, b
- Initial factors



$$\cancel{P(v)}P(s)\cancel{P(t|v)}P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

Eliminate: v

Compute:

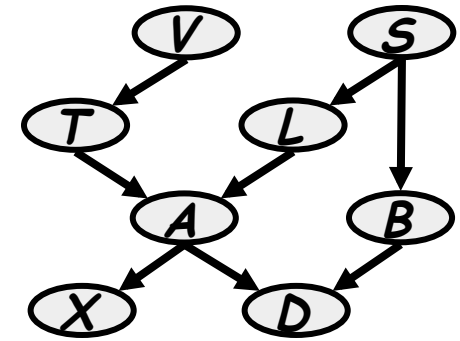
$$f_v(t) = \sum_v P(v)P(t|v)$$

$$\Rightarrow \underline{f_v(t)}P(s)P(l|s)P(b|s)P(a|t,l)P(x|a)P(d|a,b)$$

Note: $f_v(t) = P(t)$

In general, result of elimination is not necessarily a probability term

- We want to compute $P(d)$
- Need to eliminate: s, x, t, l, a, b
- Initial factors



$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)P(s)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

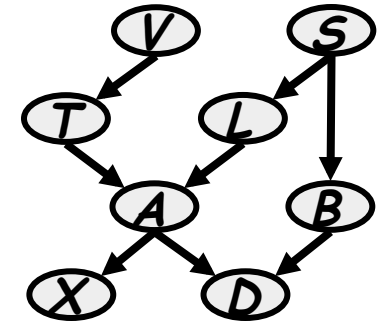
Eliminate: s

Compute: $f_s(b, l) = \sum_s P(s)P(b | s)P(l | s)$

$$\Rightarrow f_v(t)f_s(b, l)P(a | t, l)P(x | a)P(d | a, b)$$

Summing on s results in a factor with two arguments $f_s(b, l)$
 In general, result of elimination may be a function of several variables.

- We want to compute $P(d)$
- Need to eliminate: x, t, l, a, b
- Initial factors



$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)P(s)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)f_s(b, l)P(a | t, l)\underline{P(x | a)}P(d | a, b)$$

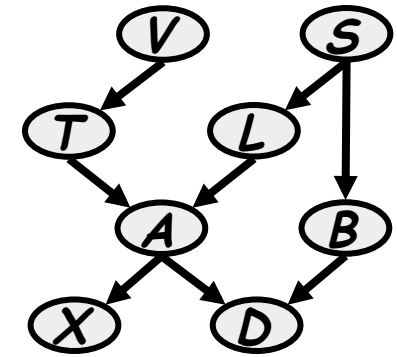
Eliminate: x

Compute: $f_x(a) = \sum_x P(x | a)$

$$\Rightarrow f_v(t)f_s(b, l)\underline{f_x(a)}P(a | t, l)P(d | a, b)$$

Note: $f_x(a) = 1$ for all values of a

- We want to compute $P(d)$
- Need to eliminate: t, l, a, b



Initial factors

$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)P(s)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)f_s(b, l)P(a | t, l)P(x | a)P(d | a, b)$$

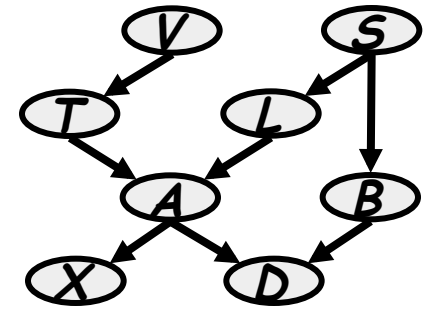
$$\Rightarrow \underline{f_v(t)}f_s(b, l)\underline{f_x(a)}P(a | t, l)P(d | a, b)$$

Eliminate: t

Compute: $f_t(a, l) = \sum_t f_v(t)P(a | t, l)$

$$\Rightarrow f_s(b, l)f_x(a)\underline{f_t(a, l)}P(d | a, b)$$

- We want to compute $P(d)$
- Need to eliminate: l, a, b
- Initial factors



$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)P(s)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)f_s(b, l)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)f_s(b, l)f_x(a)P(a | t, l)P(d | a, b)$$

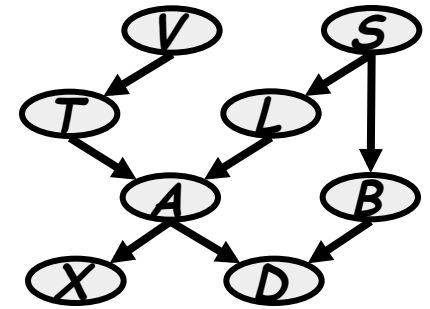
$$\Rightarrow \underline{f_s(b, l)}f_x(a)\underline{f_t(a, l)}P(d | a, b)$$

Eliminate: l

Compute: $f_l(a, b) = \sum_T f_s(b, l)f_t(a, l)$

$$\Rightarrow \underline{f_l(a, b)}f_x(a)P(d | a, b)$$

- We want to compute $P(d)$
- Need to eliminate: b
- Initial factors



$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

$$\Rightarrow f_v(t)f_s(b, l)f_x(a)P(a | t, l)P(d | a, b)$$

$$\Rightarrow f_s(b, l)f_x(a)f_t(a, l)P(d | a, b)$$

$$\Rightarrow \underline{f_t(a, b)}\underline{f_x(a)}\underline{P(d | a, b)} \Rightarrow \underline{f_a(b, d)} \Rightarrow \underline{f_b(d)}$$

Eliminate: a, b

Compute:

$$f_a(b, d) = \sum_a f_t(a, b)f_x(a)p(d | a, b) \quad f_b(d) = \sum_b f_a(b, d)$$

Basic operations

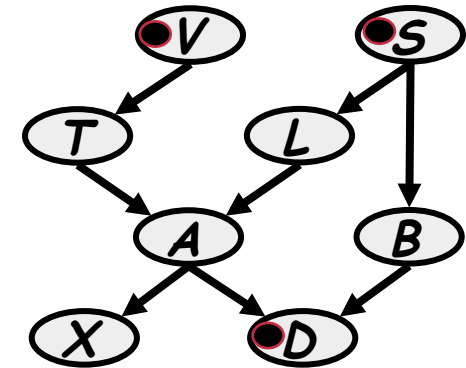
- Multiplying two factors
- Summing out a variable from a product of factors – marginalization

Multiplying factors: Pointwise product

A	B	$f_1(A,B)$	B	C	$f_2(B,C)$	A	B	C	$f_3(A,B,C)$
T	T	0.3	T	T	0.2	T	T	T	$(0.3)(0.2)$
T	F	0.7	T	F	0.8	T	T	F	$(0.3)(0.8)$
F	T	0.9	F	T	0.6	T	F	T	$(0.7)(0.6)$
F	F	0.1	F	F	0.4				

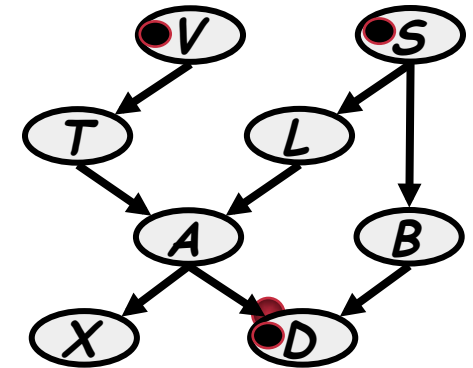
- Pointwise product is NOT
 - matrix multiplication
 - element by element multiplication

Dealing with evidence



- How do we deal with evidence?
- Suppose get evidence $V = 1, S = 0, D = 1$
- We want to compute $P(L, V = 1, S = 0, D = 1)$

Dealing with Evidence



- We start by writing the factors:

$$P(v)P(s)P(t | v)P(l | s)P(b | s)P(a | t, l)P(x | a)P(d | a, b)$$

- Since we know that $V = 1$, we don't need to eliminate V
- Instead, we can replace the factors $P(V)$ and $P(T|V)$ with

$$f_{P(V)} = P(V = 1) \quad f_{P(T|V)}(T) = P(T | V = 1)$$

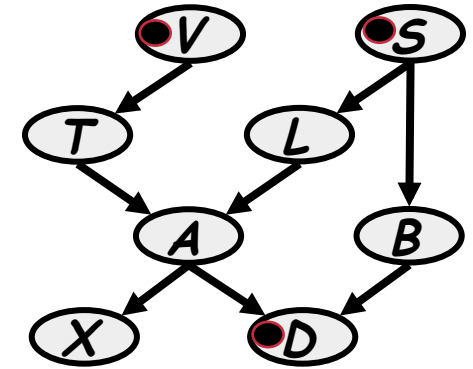
- These “select” the appropriate parts of the original factors given the evidence
- Note that $f_{P(V)}$ is a constant, and thus does not appear in elimination of other variables

Variable Elimination

- We now understand variable elimination as a sequence of rewriting operations
- Actual computation is done in elimination step
- Computation depends on order of elimination

Dealing with Evidence

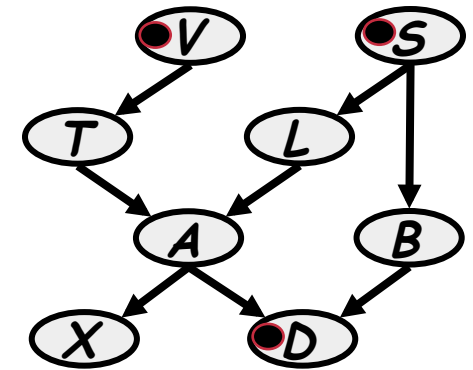
- Given evidence $V = 1, S = 0, D = 1$
- Compute $P(L, V = 1, S = 0, D = 1)$
- Initial factors, after setting evidence:



$$f_{P(V)} f_{P(S)} f_{P(T|V)}(t) f_{P(L|S)}(l) f_{P(B|S)}(b) P(a | t, l) P(x | a) f_{P(D|A,B)}(a, b)$$

Dealing with Evidence

- Given evidence $V = 1, S = 0, D = 1$
- Compute $P(L, V = 1, S = 0, D = 1)$
- Initial factors, after setting evidence:

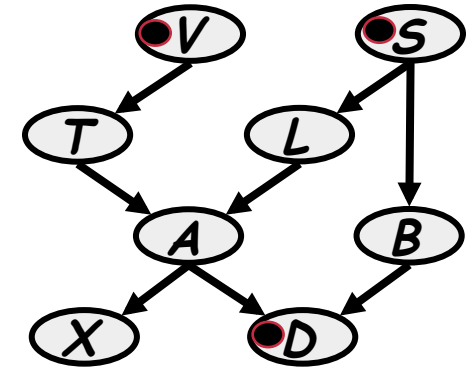


$$f_{P(V)} f_{P(S)} f_{P(T|V)}(t) f_{P(L|S)}(l) f_{P(B|S)}(b) P(a | t, l) P(x | a) f_{P(D|A,B)}(a, b)$$

- Eliminating X , we get

$$f_{P(V)} f_{P(S)} f_{P(T|V)}(t) f_{P(L|S)}(l) f_{P(B|S)}(b) P(a | t, l) f_x(a) f_{P(D|A,B)}(a, b)$$

Dealing with Evidence



- Given evidence $V = 1, S = 0, D = 1$
- Compute $P(L, V = 1, S = 0, D = 1)$
- Initial factors, after setting evidence:

- Eliminating X , we get

$$f_{P(V)} f_{P(S)} f_{P(T|V)}(t) f_{P(L|S)}(l) f_{P(B|S)}(b) P(a | t, l) P(x | a) f_{P(D|a,b)}(a, b)$$

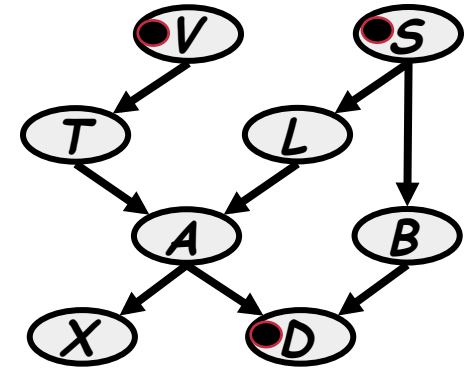
- Eliminating t , we get

$$f_{P(V)} f_{P(S)} \underline{f_{P(T|V)}(t)} f_{P(L|S)}(l) f_{P(B|S)}(b) P(a | t, l) f_x(a) f_{P(D|a,b)}(a, b)$$

$$f_{P(V)} f_{P(S)} f_{P(L|S)}(l) f_{P(B|S)}(b) f_t(a, l) f_x(a) f_{P(D|a,b)}(a, b)$$

Dealing with Evidence

- Given evidence $V = 1, S = 0, D = 1$
- Compute $P(L, V = 1, S = 0, D = 1)$
- Initial factors, after setting evidence:



$$f_{P(v)} f_{P(s)} f_{P(t|v)}(t) f_{P(l|s)}(l) f_{P(b|s)}(b) P(a | t, l) P(x | a) f_{P(d|a,b)}(a, b)$$

- Eliminating x , we get

$$f_{P(v)} f_{P(s)} f_{P(t|v)}(t) f_{P(l|s)}(l) f_{P(b|s)}(b) P(a | t, l) f_x(a) f_{P(d|a,b)}(a, b)$$

- Eliminating t , we get

$$f_{P(v)} f_{P(s)} f_{P(l|s)}(l) f_{P(b|s)}(b) f_t(a, l) f_x(a) f_{P(d|a,b)}(a, b)$$

- Eliminating a , we get

$$f_{P(v)} f_{P(s)} f_{P(l|s)}(l) f_{P(b|s)}(b) f_a(b, l)$$

Variable Elimination Algorithm

- Let X_1, \dots, X_m be an ordering on the non-query variables

$$\sum_{X_1} \sum_{X_2} \dots \sum_{X_m} \prod_j P(X_j \mid Parents(X_j))$$

- For $i = m, \dots, 1$
 - Leave in the summation for X_i only factors mentioning X_i
 - Multiply the factors, getting a factor that contains a number for each value of the variables mentioned, including X_i
 - Sum out X_i , getting a factor f that contains a number for each value of the variables mentioned, not including X_i
 - Replace the multiplied factor in the summation

Complexity of variable elimination

- Suppose in one elimination step we compute

$$f_x(y_1, \dots, y_k) = \sum f'_x(x, y_1, \dots, y_k)$$

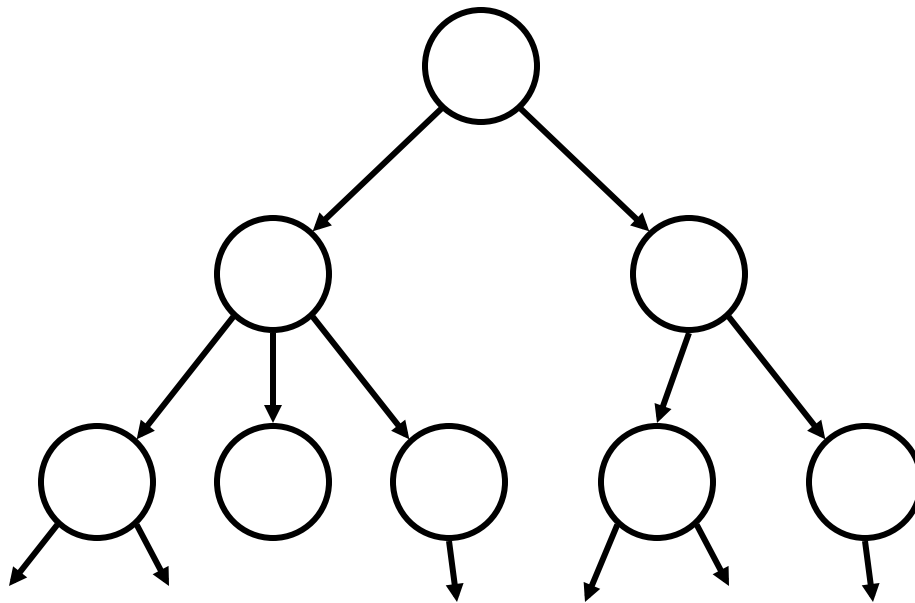
$$f'_x(x, y_1, \dots, y_k) = \prod_{i=1}^m f_i(x, y_{1,1}, \dots, y_{1,i})$$

- This requires $m \cdot |Domain(X)| \cdot \prod_i |Domain(Y_i)|$ multiplications
- Complexity is (not surprisingly) exponential in number of variables in the intermediate factor!

Variable Elimination

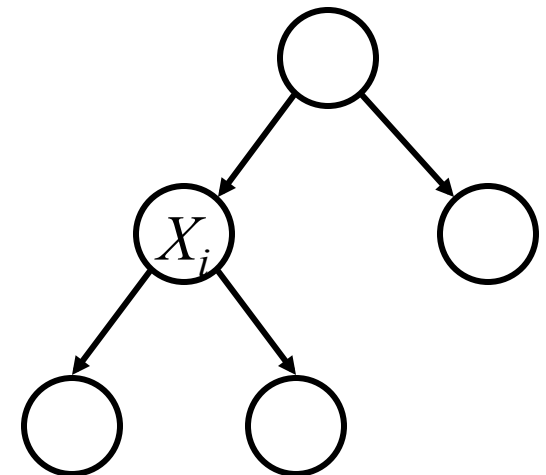
- We want to select “good” elimination orderings that reduce complexity
- This can be done by examining a graph theoretic property of the “induced” graph.
- This reduces the problem of finding good ordering to graph-theoretic operation that is well-understood—unfortunately computing it is NP-hard!

Bayesian Network Inference in polytrees – Message Passing algorithm



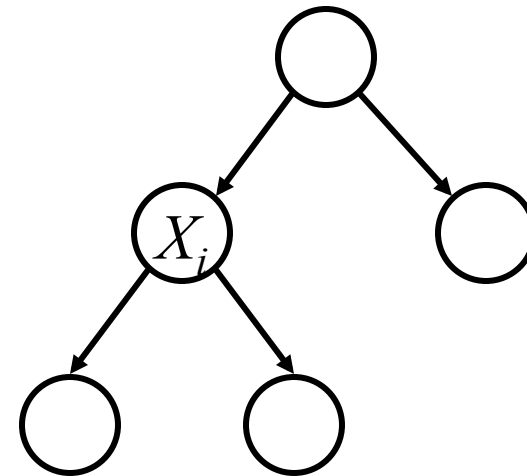
Decomposing the probabilities

- Suppose we want $P(X_i | E)$ where E is some set of evidence variables.
- Let's split E into two parts:
 - E_i^- is the part consisting of assignments to variables in the subtree rooted at X_i
 - E_i^+ is the rest of the variables in E



Decomposing the probabilities

$$\begin{aligned}
 P(X_i | E) &= P(X_i | E_i^-, E_i^+) \\
 &= \frac{P(E_i^- | X_i, E_i^+) P(X_i | E_i^+)}{P(E_i^- | E_i^+)} \\
 &= \frac{P(E_i^- | X_i) P(X_i | E_i^+)}{P(E_i^- | E_i^+)} \\
 &= \alpha \pi(X_i) \lambda(X_i)
 \end{aligned}$$



Where:

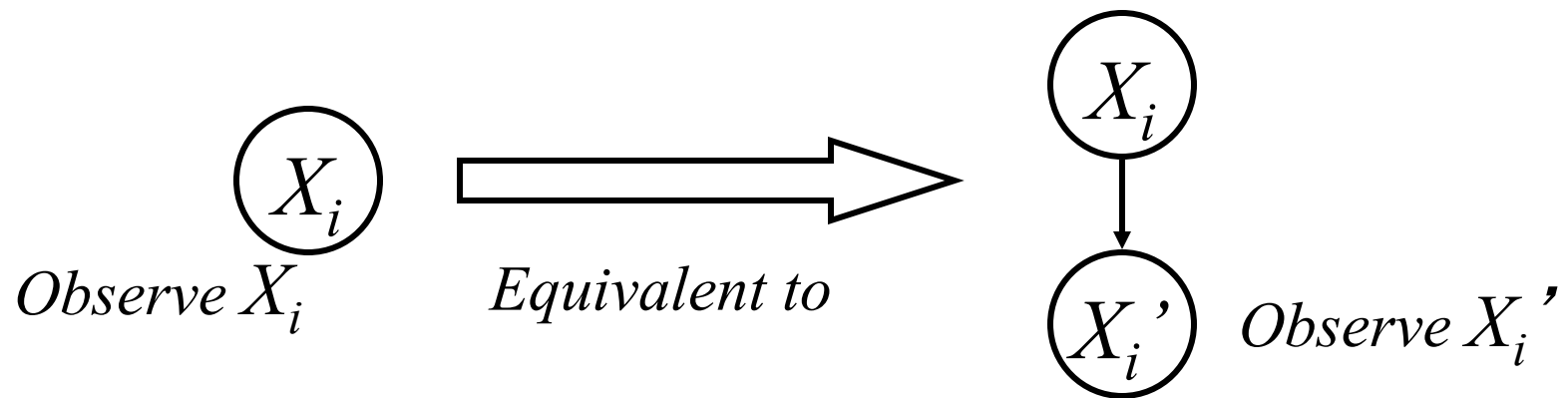
- α is a constant independent of X_i
- $\pi(X_i) = P(X_i | E_i^+)$
- $\lambda(X_i) = P(E_i^- | X_i)$

Using the decomposition for inference

- We can use this decomposition to do inference as follows. First, compute $\lambda(X_i) = P(E_i^- | X_i)$ for all X_i recursively, using the leaves of the tree as the base case.

Quick aside: “Virtual evidence”

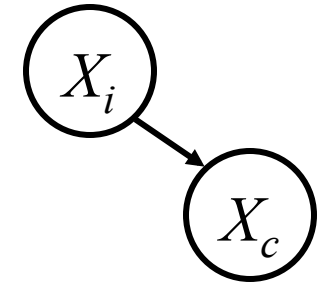
- For theoretical simplicity, but without loss of generality, let us assume that *all* variables in E (the evidence set) are leaves in the tree.



Where $P(X_i' | X_i) = 1$ if $X_i' = X_i$, 0 otherwise

Calculating $\lambda(X_i)$ for non-leaves

- Suppose X_i has one child, $X_j = X_c$.
- Then:



$$\begin{aligned}
 \lambda(X_i) &= P(E_i^- \mid X_i) = \sum_{X_j} P(E_i^-, X_j \mid X_i) \\
 &= \sum_{X_j} P(X_j \mid X_i) P(E_i^- \mid X_i, X_j) \\
 &= \sum_{X_j} P(X_j \mid X_i) P(E_i^- \mid X_j) \\
 &= \sum_{X_j} P(X_j \mid X_i) \lambda(X_j)
 \end{aligned}$$

Calculating $\lambda(X_i)$ for non-leaves

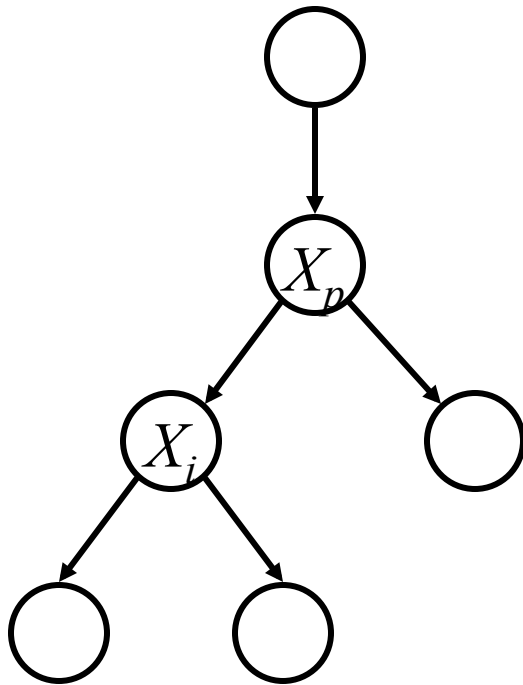
- Now, suppose X_i has a set of children, C .
- Since X_i *d-separates* each of its subtrees, the contribution of each subtree to $\lambda(X_i)$ is independent:

$$\lambda(X_i) = P(E_i^- | X_i) = \prod_{X_j \in C} \lambda_j(X_i)$$

$$= \prod_{X_j \in C} \left[\sum_{X_j} P(X_j | X_i) \lambda(X_j) \right]$$

- where $\lambda_j(X_i)$ is the contribution to $P(E_i^- | X_i)$ of the part of the evidence lying in the subtree rooted at one of X_i 's children X_j .

Computing $\pi(X_i)$



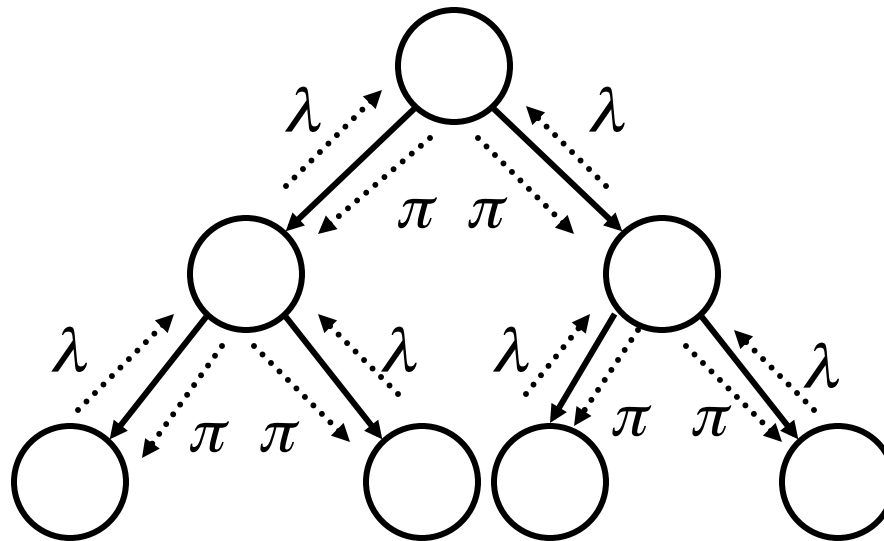
$$\begin{aligned} \pi(X_i) &= P(X_i | E_i^+) = \sum_{X_p} P(X_i, X_p | E_i^+) \\ &= \sum_{X_p} P(X_i | X_p, E_i^+) P(X_p | E_i^+) \\ &= \sum_{X_p} P(X_i | X_p) P(X_p | E_i^+) \\ &= \sum_{X_p} P(X_i | X_p) \frac{P(X_p | E)}{\lambda_i(X_p)} \\ &= \sum_{X_p} P(X_i | X_p) \pi_i(X_p) \end{aligned}$$

- Where $\pi_i(X_p)$ is defined as

$$\frac{P(X_p | E)}{\lambda_i(X_p)}$$

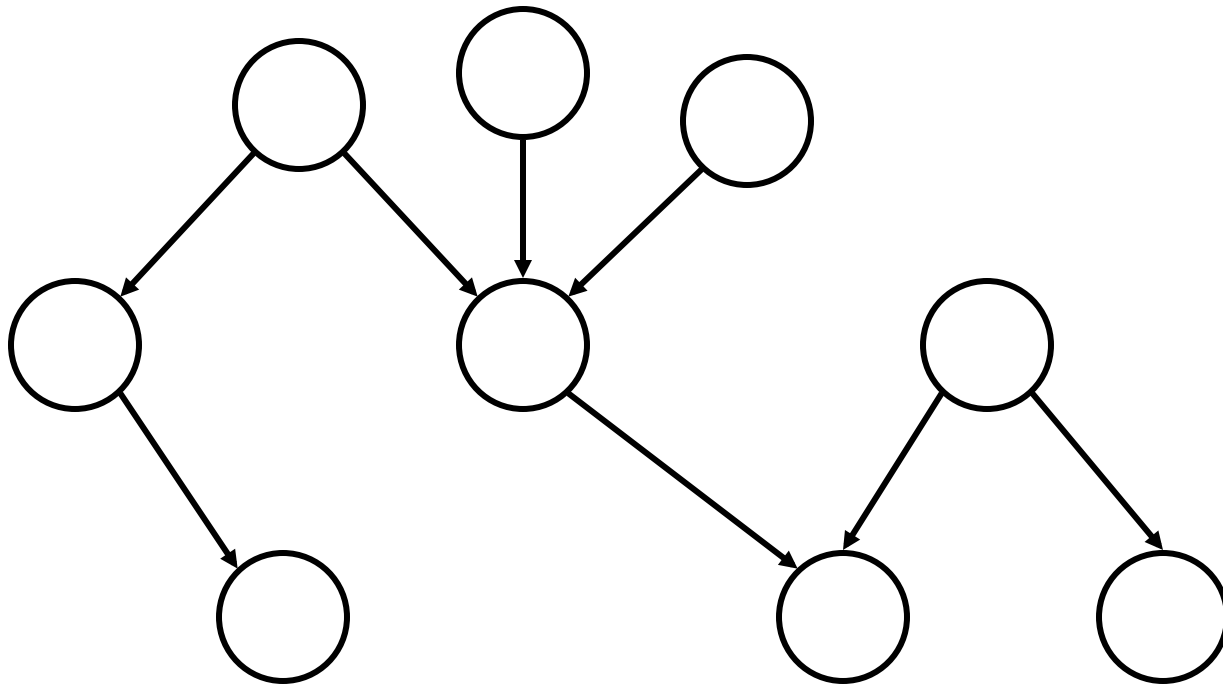
Bayesian network inference in trees

- Thus we can compute all the $\pi(X_i)$'s, and, in turn, all the $P(X_i | E)$'s.
- Can think of nodes as autonomous processors passing λ and π messages to their neighbors



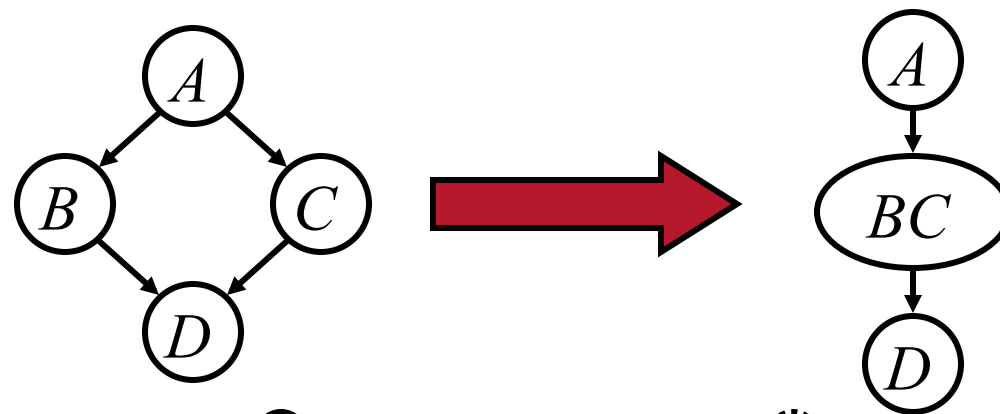
Polytrees

- Previous technique can be generalized to *polytrees*: undirected versions of the graphs are still trees, but nodes can have more than one parent

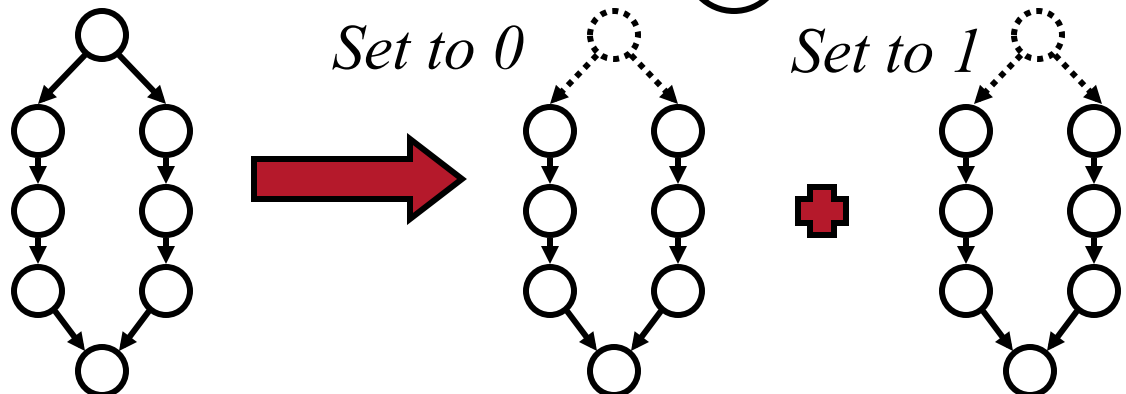


Dealing with cycles

- Can deal with undirected cycles in graph by
- clustering variables together

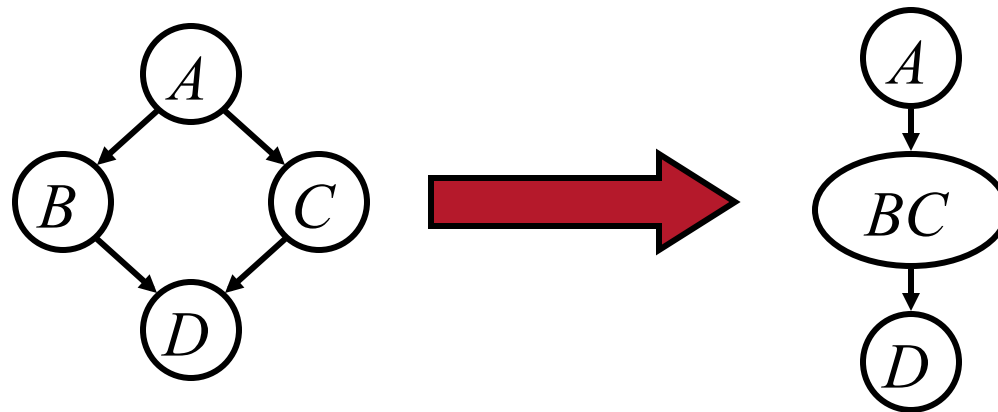


- Conditioning



Dealing with cycles

- Can deal with undirected cycles in graph by
- clustering variables together

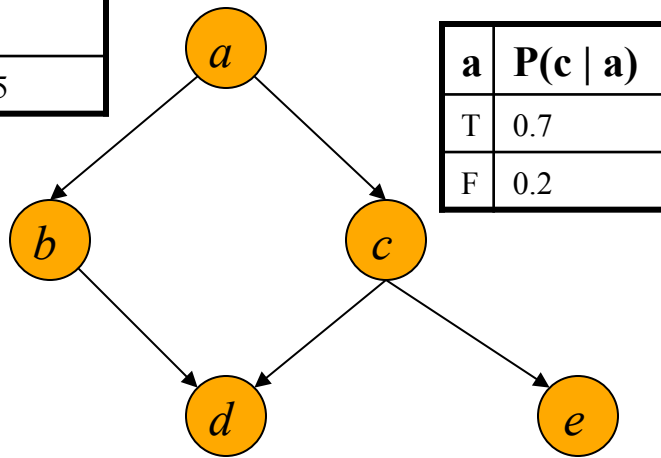


Clustering Methods

- Clustering methods transform an multiply connected BN (MCBN) into a “probabilistically equivalent” poly-tree
 - Such a transformation is done by merging several RVs in MCBNs into a single compound RV in order to break the information flow over multiple paths
 - “Probabilistic equivalence” is guaranteed by computing the joint probability distribution of the RVs that are merged into a compound RV

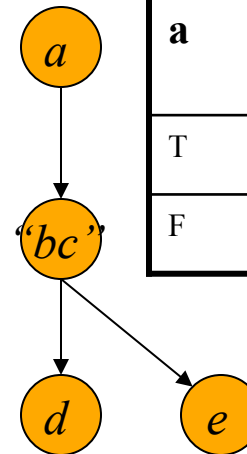
Ad hoc clustering example

a	P(b a)
T	0.9
F	0.25



a	P(c a)
T	0.7
F	0.2

b,c	P(d b,c)
T, T	0.8
T, F	0.6
F, T	0.6
F, F	0.1



a	P("bc" a)			
	TT	TF	FT	FF
T	0.63	0.27	0.07	0.03
F	0.45	0.2	0.15	0.6

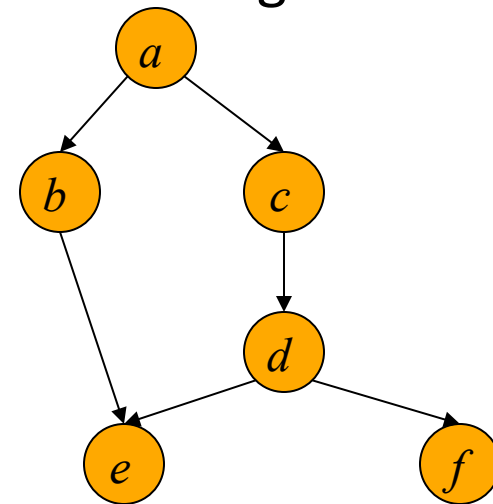
"bc"	P(d "bc")
TT	0.8
TF	0.6
FT	0.6
FF	0.1

After Clustering

- Once we cluster the events of an MCN, we can use any exact inference algorithms developed for singly-connected networks
- Clustering reduces the size of the network, sometimes exponentially
- However the computation required for inference is not necessarily reduced
 - Building the compound CPTs may still take exponential time in the worst case

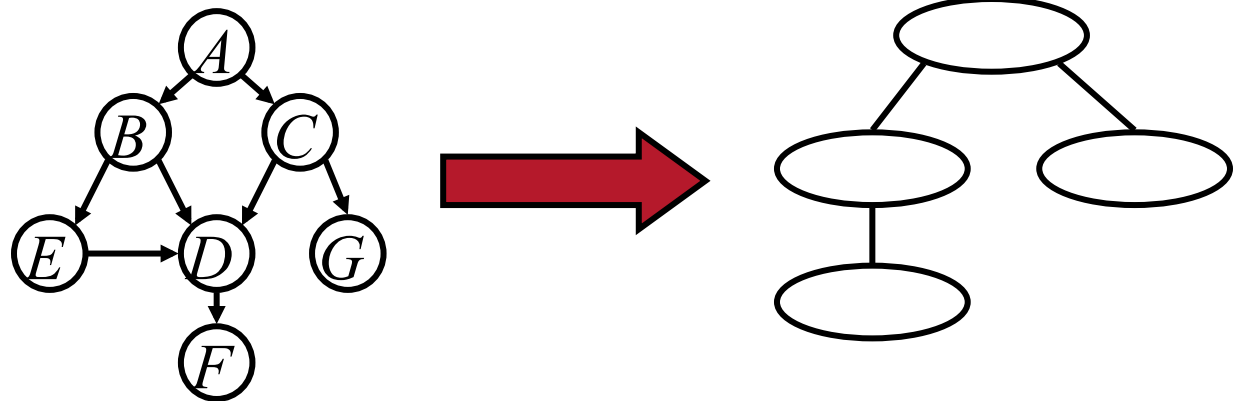
Junction Trees

- The transformation in the clustering example we have discussed is “ad hoc”
 - We just looked at the network and merged RVs such that we avoided information flow to the same RV through multiple paths
- The motivation behind the junction tree methods is to provide a systematic and an efficient way to do clustering
 - Moralization
 - Triangulation
 - Restructuring
 - Belief Update



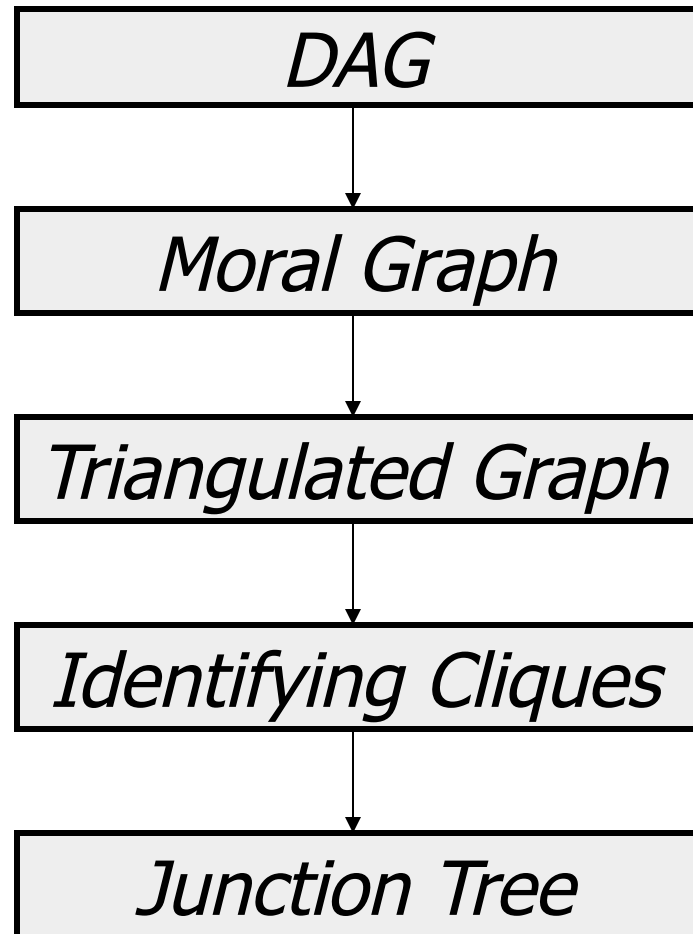
Join trees or junction trees

Arbitrary Bayesian network can be transformed via a graph-theoretic trick into a *join tree* (also used in databases).



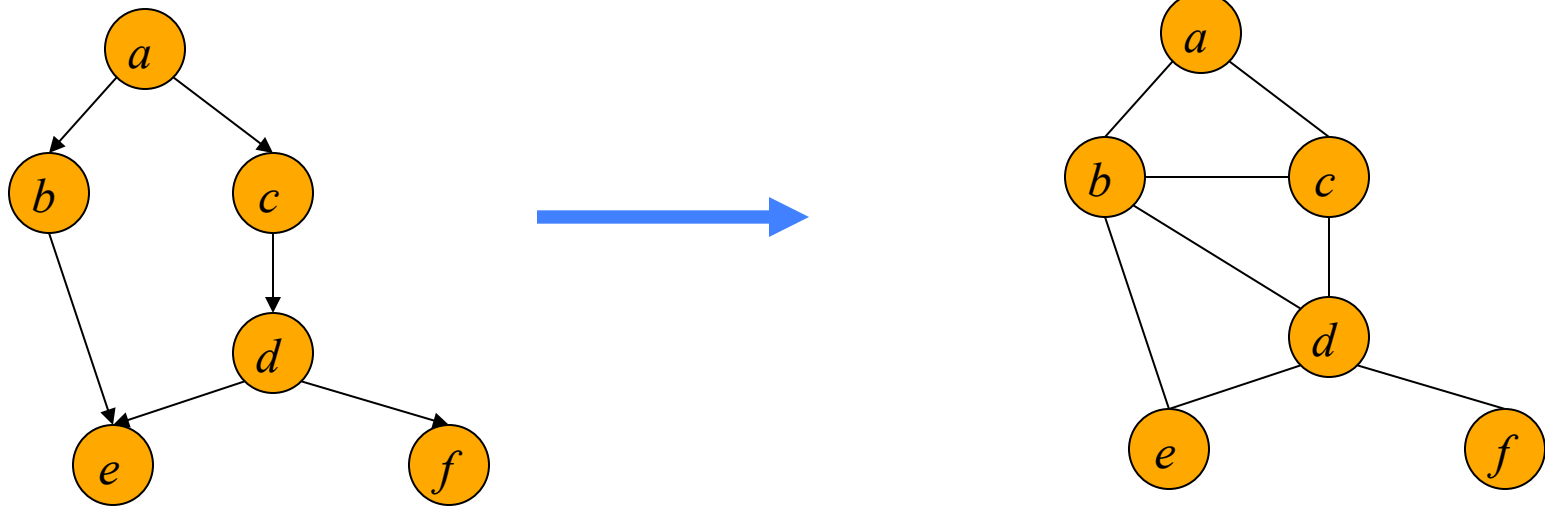
In the worst case the join tree nodes must take on values whose number grows exponentially with the number of nodes that are clustered together, but this often works well in practice when the number of nodes per cluster is small

Building Junction Trees



Moralization & Triangulation

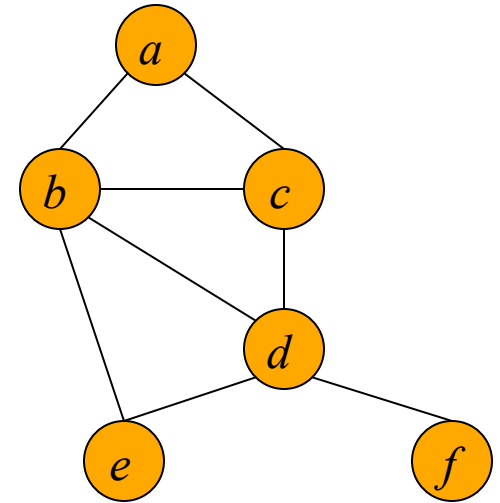
- Considering the undirected network, “marry” the parent nodes that have a common child



- Triangulate every cycle produced from marriages

Restructuring

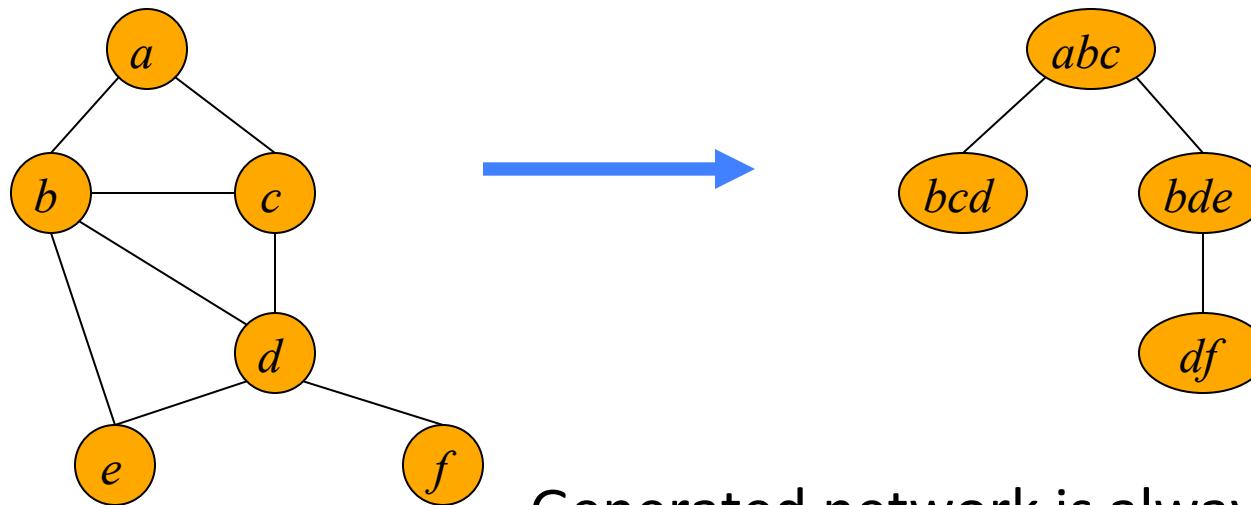
- Identify all maximal cliques in the network
 - In this example, we have
 - “abc”, “bcd”, “bde”, and “df”



- Identify the “separators” between the maximal cliques
 - “bc” between “abc” and “bcd”
 - “bd” between “bcd” and “bde”
 - “d” between (1) “bde” and “df”, and (2) “bcd” and “df”

Restructuring

- Create a new network where the cliques of the original networks are compound nodes



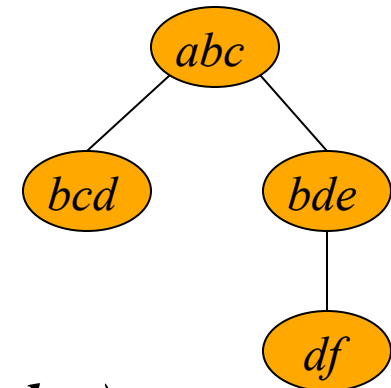
Generated network is always a poly-tree (or a poly-graph), called a **Junction Tree** (or a **Junction graph**)

Belief Update in Junction Trees

- The CPTs for the nodes in the junction tree are computed by the cross-products of the CPTs from the original network

Example:

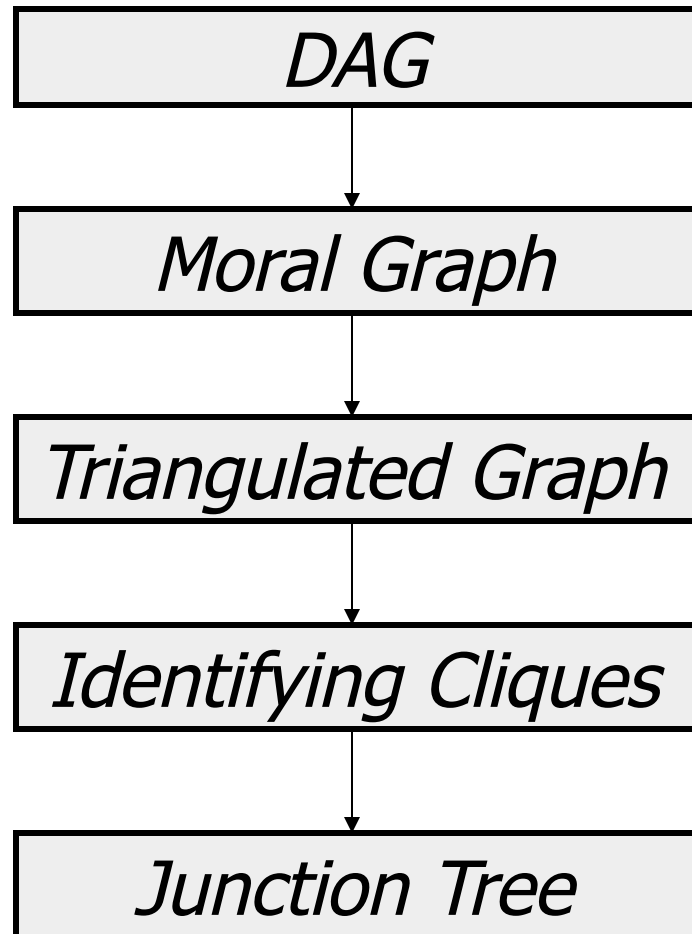
“abc”	P(“bcd” “abc”)
TTT	0.8
TTF	0.6
TFT	0.6
TFF	0.1
FTT	0.8
FTF	0.6
FFT	0.6
FFF	0.1



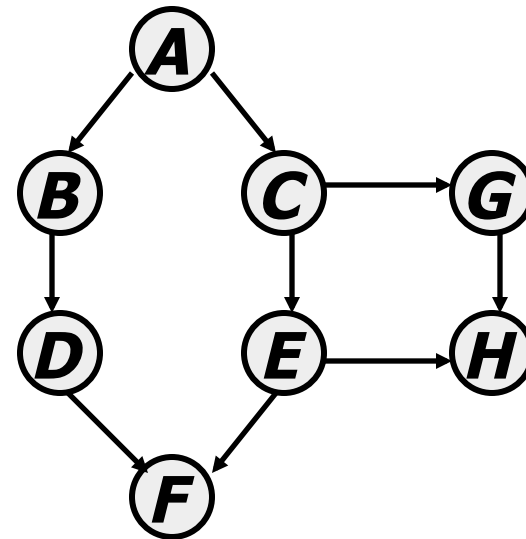
← $P(b | a) P(c | a) P(d | b c)$

- Then, the belief update can be done by using exact inference methods for singly-connected trees

Building Junction Trees

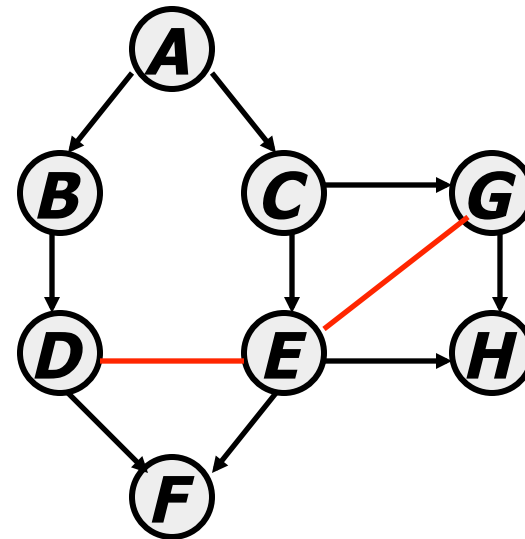


Constructing the Moral Graph



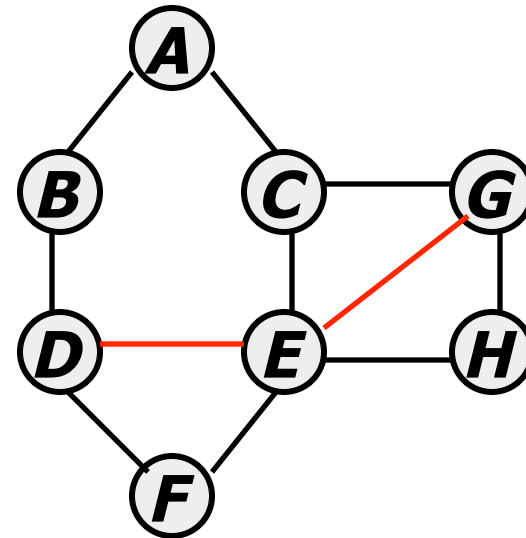
Constructing The Moral Graph

- Add undirected edges to all co-parents which are not currently joined – *Marrying parents*



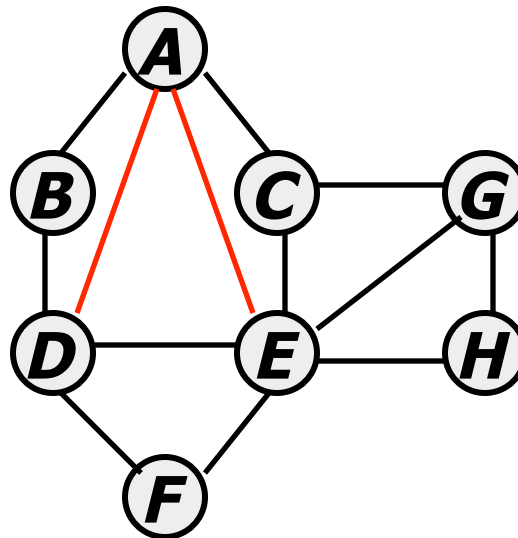
Constructing The Moral Graph

- Add undirected edges to all co-parents which are not currently joined – *Marrying parents*
- Drop the directions of the arcs



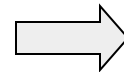
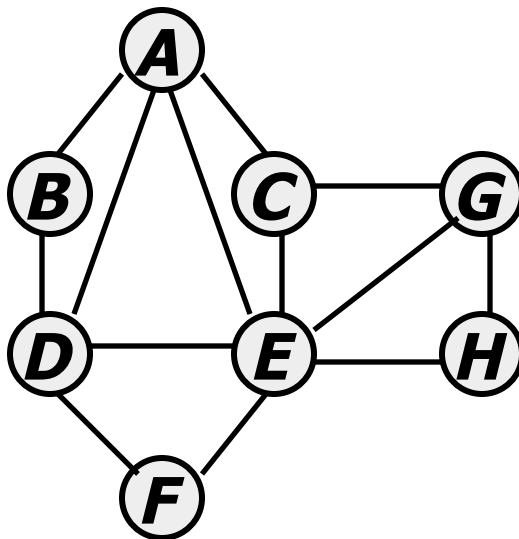
Triangulating

- An undirected graph is triangulated iff every cycle of length >3 contains an edge that connects two nonadjacent nodes



Identifying Cliques

- A clique is a subgraph of an undirected graph that is complete (has an edge between each pair of vertices) and maximal



EGH

CEG

DEF

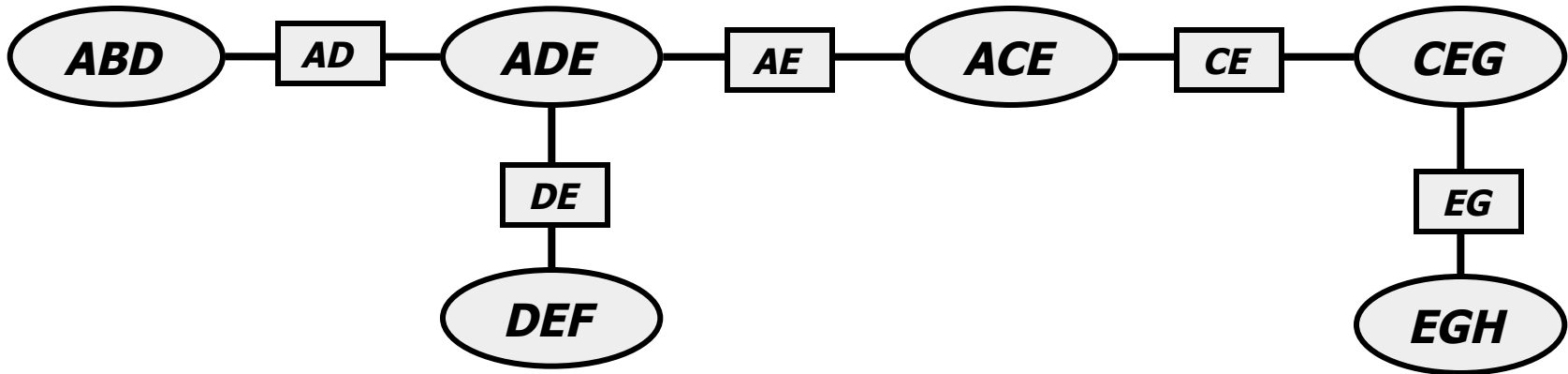
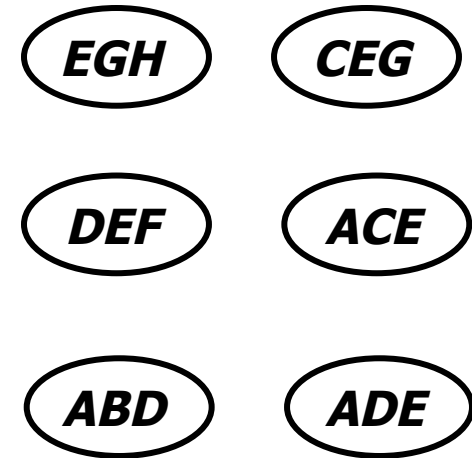
ACE

ABD

ADE

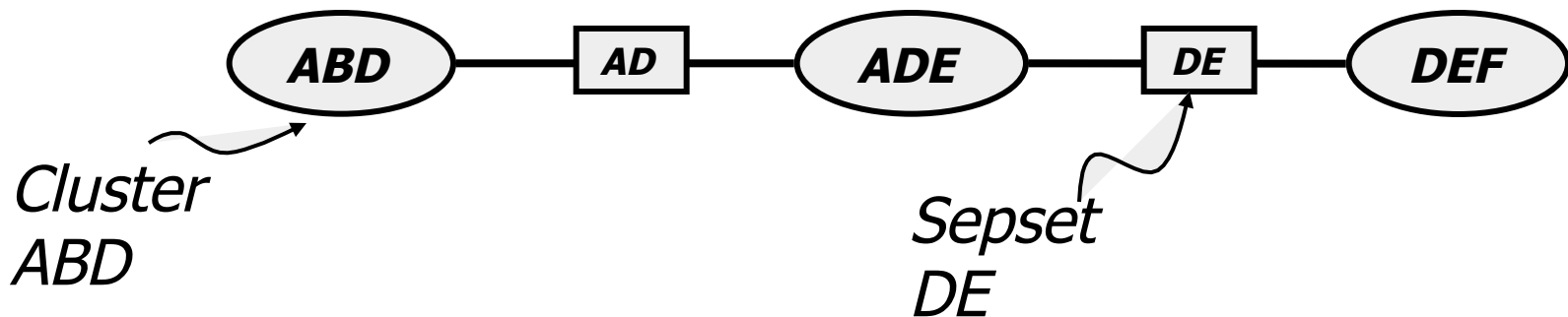
Junction Tree

- A junction tree is a subgraph of the clique graph that
 - is a tree
 - contains all the cliques
 - satisfies the junction tree property
- *Junction tree property:* For each pair U, V of cliques with intersection S , all cliques on the path between U and V contain S .



Properties of Junction Tree

- An undirected tree
- Each node is a cluster (nonempty set) of variables
- **Running intersection property:**
 - Given two clusters X and Y clusters on the path between X and Y contain $X \cap Y$
- Separator sets (sepsets):
 - Intersection of the adjacent cluster



Properties of Junction Tree

- Belief potentials:
 - Map each instantiation of clusters or sepsets into a real number
- Constraints:
 - Consistency: for each cluster \mathbf{X} and neighboring sepset \mathbf{S}

– The joint distribution

$$\sum_{\mathbf{X} \setminus \mathbf{S}} \phi_{\mathbf{X}} = \phi_{\mathbf{S}}$$

$$P(\mathbf{U}) = \frac{\prod_i \phi_{\mathbf{X}_i}}{\prod_j \phi_{\mathbf{S}_j}}$$

Properties of Junction Tree

- If a junction tree satisfies these properties, it follows that:
 - For each cluster (or sepset) \mathbf{X}
 - The probability distribution of any variable V using any cluster (or sepset) \mathbf{X} at contains V

$$\phi_{\mathbf{X}} = P(\mathbf{X})$$

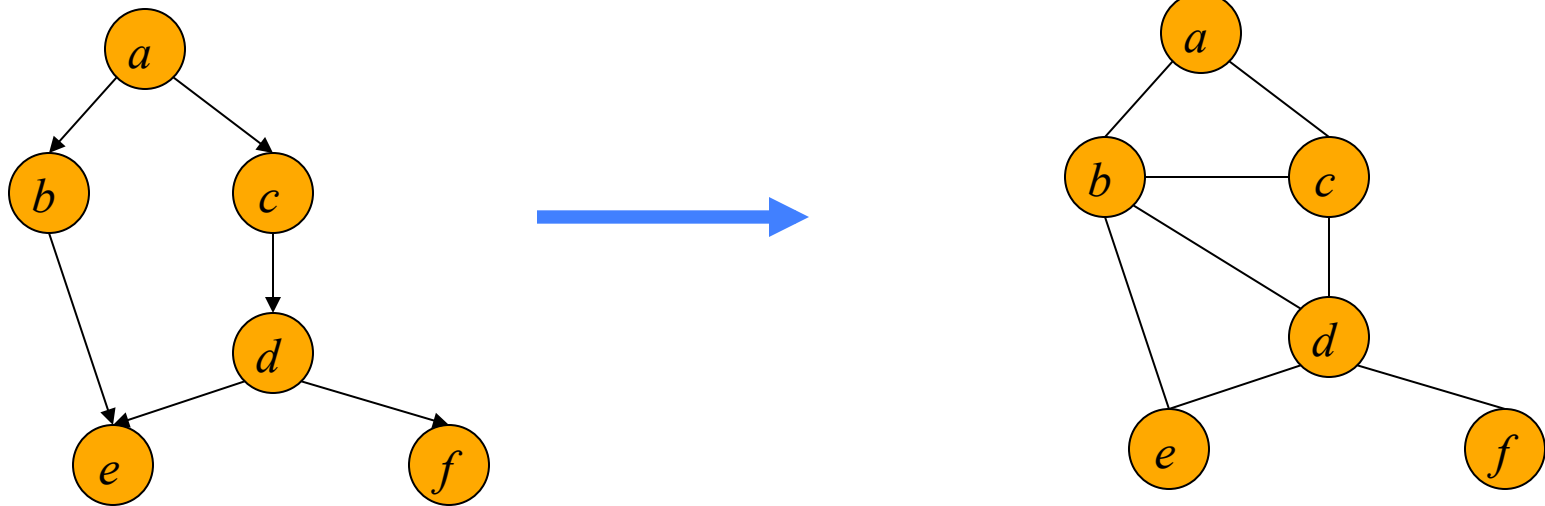
$$P(V) = \sum_{\mathbf{X} \setminus \{V\}} \phi_{\mathbf{X}}$$

Inference

- Choose a root
- For each distribution (CPT) in the original Bayes Net, put this distribution into one of the clique nodes that contains all the variables referenced by the CPT. (At least one such node must exist because of the moralization step).
- For each clique node, take the product of the distributions (as in variable elimination).

Moralization & Triangulation

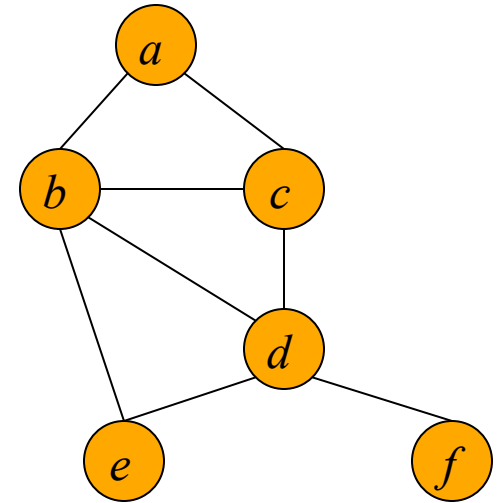
- Considering the undirected network, “marry” the parent nodes that have a common child



- Triangulate every cycle produced from marriages

Restructuring

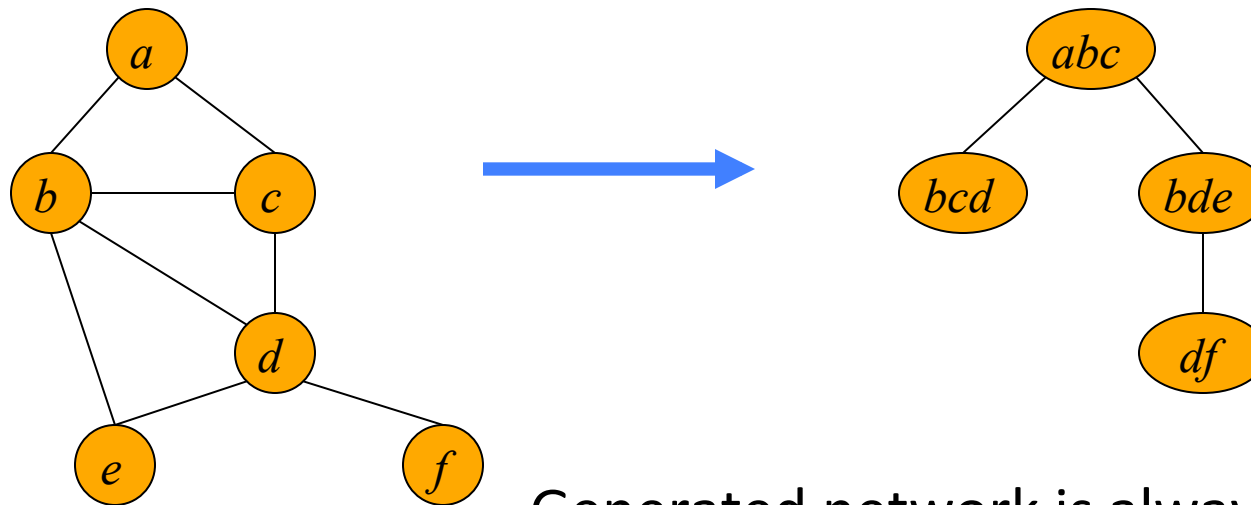
- Identify all maximal cliques in the network
 - In this example, we have
 - “abc”, “bcd”, “bde”, and “df”



- Identify the “separators” between the maximal cliques
 - “bc” between “abc” and “bcd”
 - “bd” between “bcd” and “bde”
 - “d” between (1) “bde” and “df”, and (2) “bcd” and “df”

Restructuring

- Create a new network where the cliques of the original networks are compound nodes



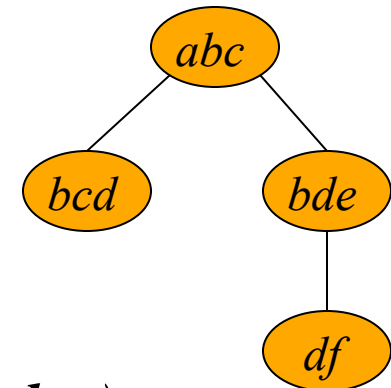
Generated network is always a poly-tree (or a poly-graph), called a **Junction Tree** (or a **Junction graph**)

Belief Update in Junction Trees

- The CPTs for the nodes in the junction tree are computed by the cross-products of the CPTs from the original network

Example:

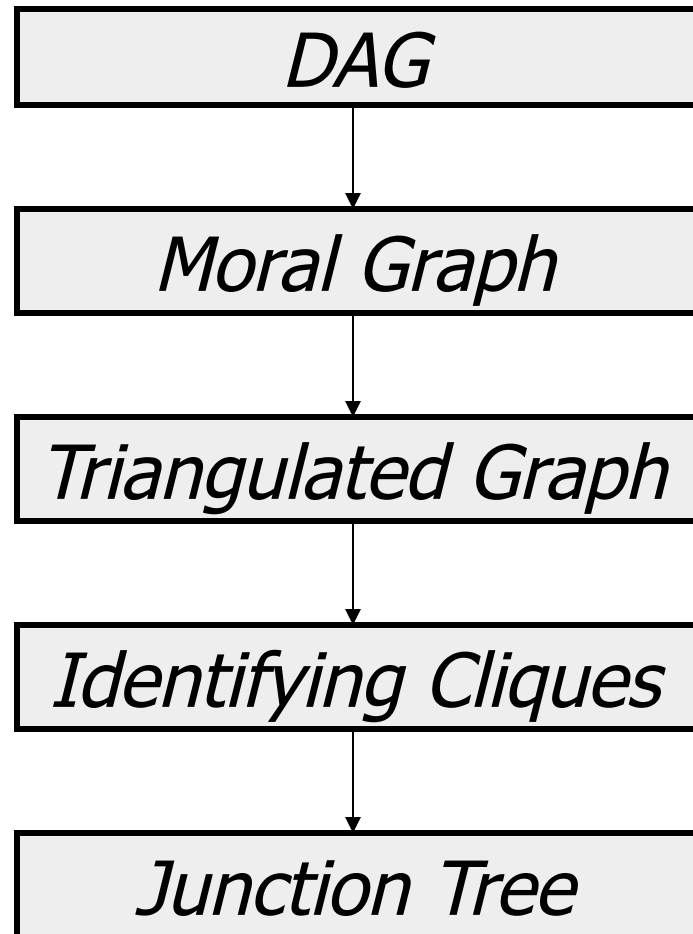
“abc”	P(“bcd” “abc”)
TTT	0.8
TTF	0.6
TFT	0.6
TFF	0.1
FTT	0.8
FTF	0.6
FFT	0.6
FFF	0.1



← $P(b | a) P(c | a) P(d | b c)$

- Then, the belief update can be done by using exact inference methods for singly-connected trees

Building Junction Trees



Inference in Bayesian network

Exact inference algorithms:

- Variable elimination
- Symbolic inference (D' Ambrosio)
- Message passing algorithm (Pearl)
- Clustering and join tree approach (Lauritzen, Spiegelhalter)

Approximate inference algorithms:

- Monte Carlo methods:
- Forward sampling, Likelihood sampling
- Variational methods

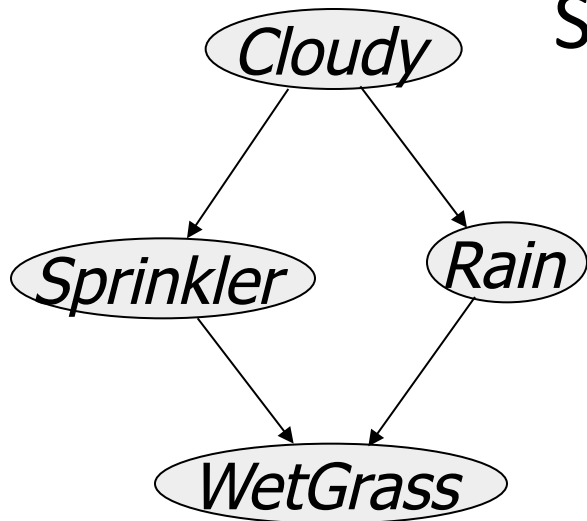
Approximate Inference

- With large and highly connected graphical models, the associated cliques for the junction tree algorithm or the intermediate factors in the variable elimination algorithm will grow in size, generating an exponential blowup in the number of computations performed

Approximate Inference: Stochastic simulation

- Suppose you are given values for some subset of the variables, G , and want to infer values for unknown variables, U
- Randomly generate a very large number of instantiations from the BN
 - Generate instantiations for **all** variables – start at root variables and work your way “forward”
- Only keep those instantiations that are consistent with the values for G
- Use the frequency of values for U to get estimated probabilities
- Accuracy of the results depends on the size of the sample (asymptotically approaches exact results)

Stochastic Simulation



$$P(\text{WetGrass} | \text{Cloudy})?$$

$$P(\text{WetGrass} | \text{Cloudy}) = P(\text{WetGrass}, \text{Cloudy}) / P(\text{Cloudy})$$

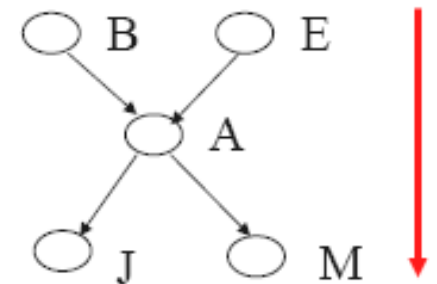
1. Draw N samples from the BN by repeating 1.1 and 1.2
 - 1.1. Guess Cloudy at random according to $P(\text{Cloudy})$
 - 1.2. For each guess of Cloudy, guess Sprinkler and Rain, then WetGrass
2. Compute the ratio of the # runs where WetGrass and Cloudy are True over the # runs where Cloudy is True

Stochastic simulation

- The probability is approximated using sample frequencies

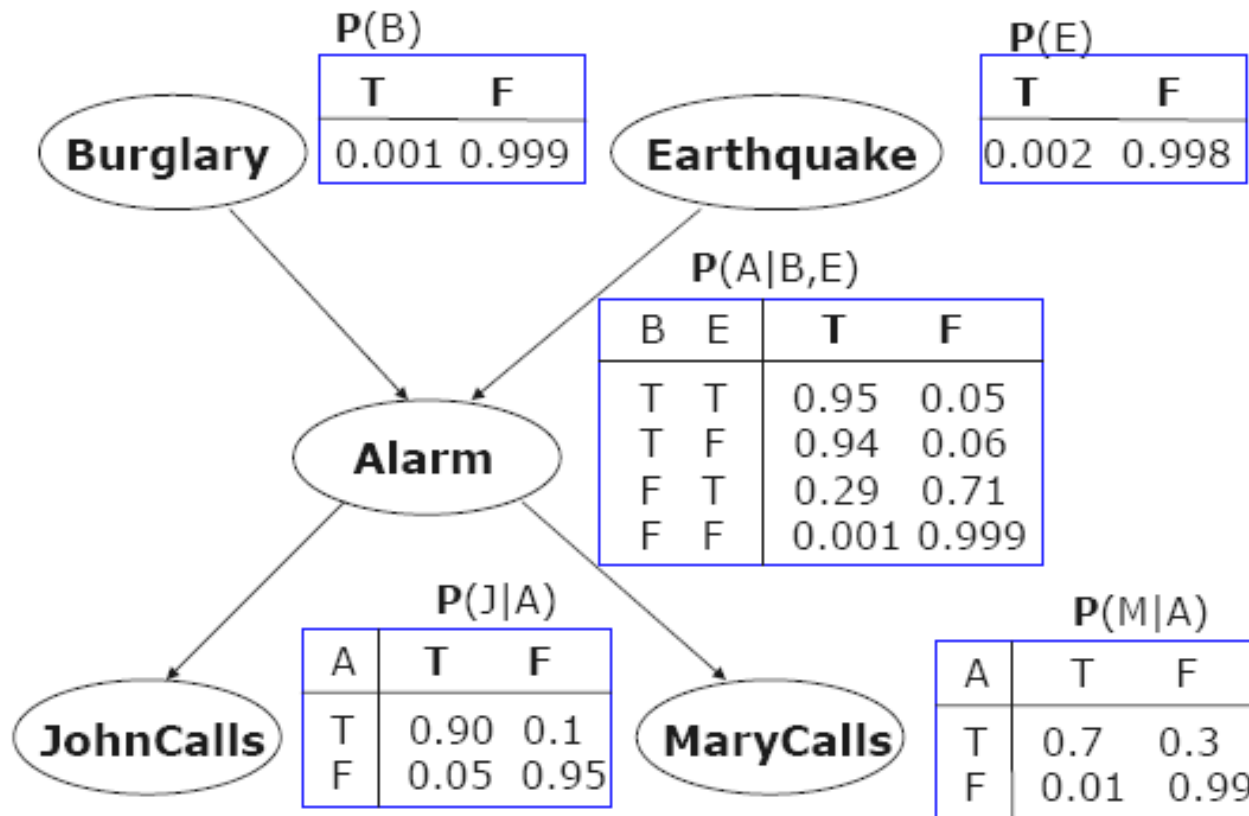
BN sampling:

- Generate sample in a top down manner, following the links in BN
- A sample is an assignment of values to all variables

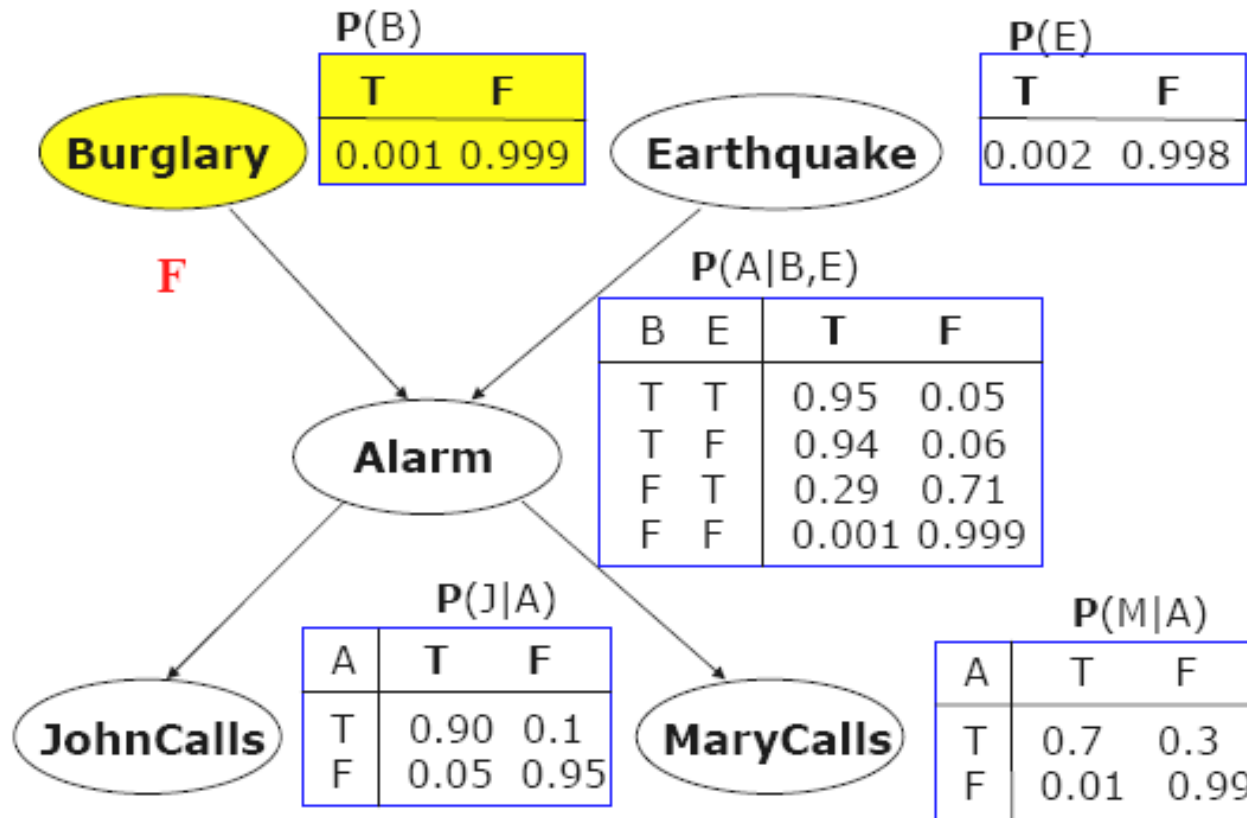


BN Sampling Example

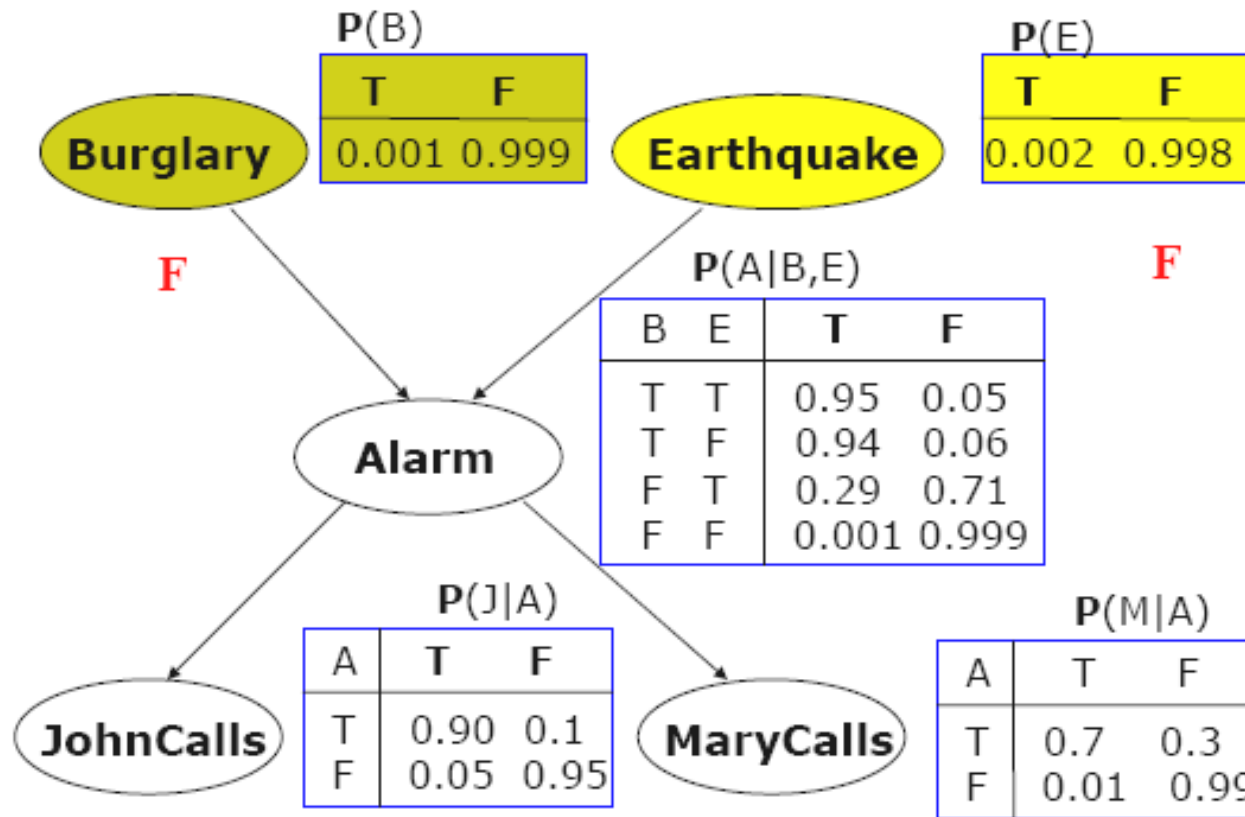
Goal: To infer $P(B | J = T, M = F)$



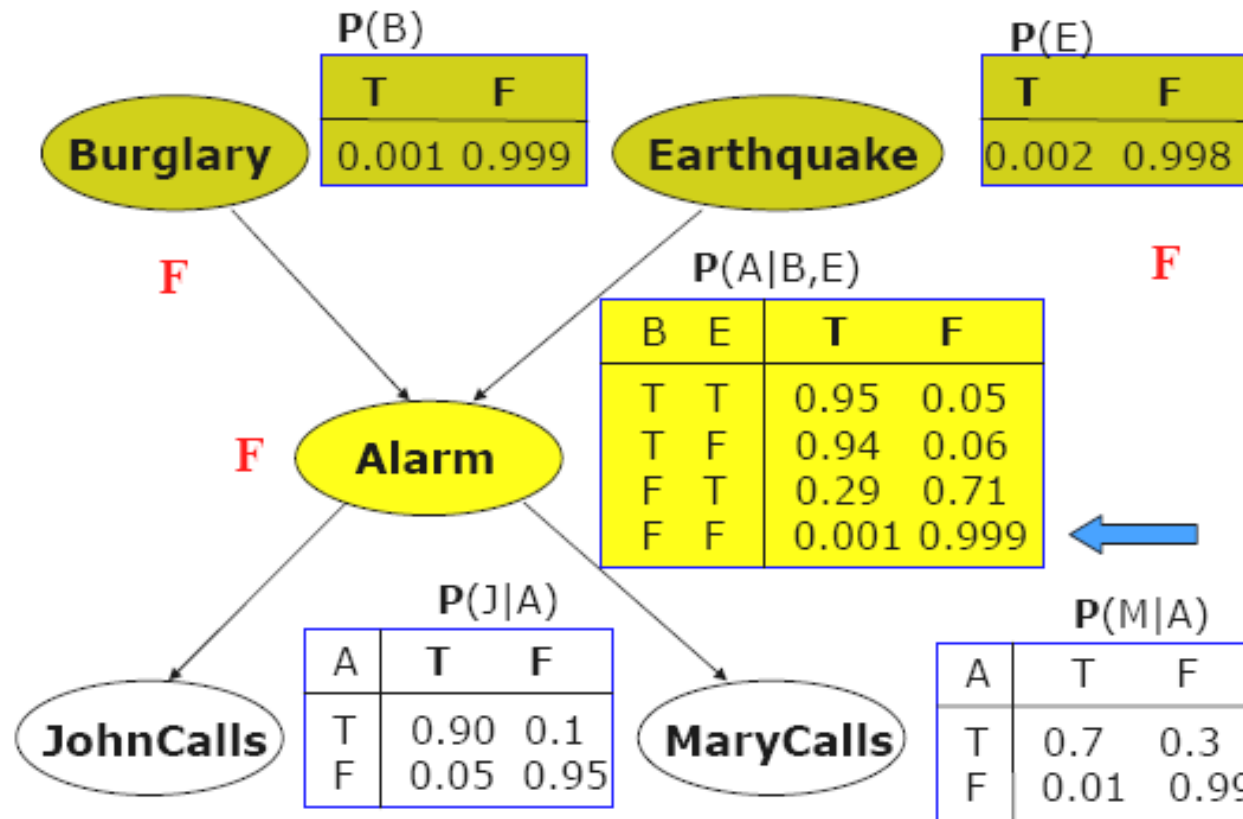
BN Sampling Example



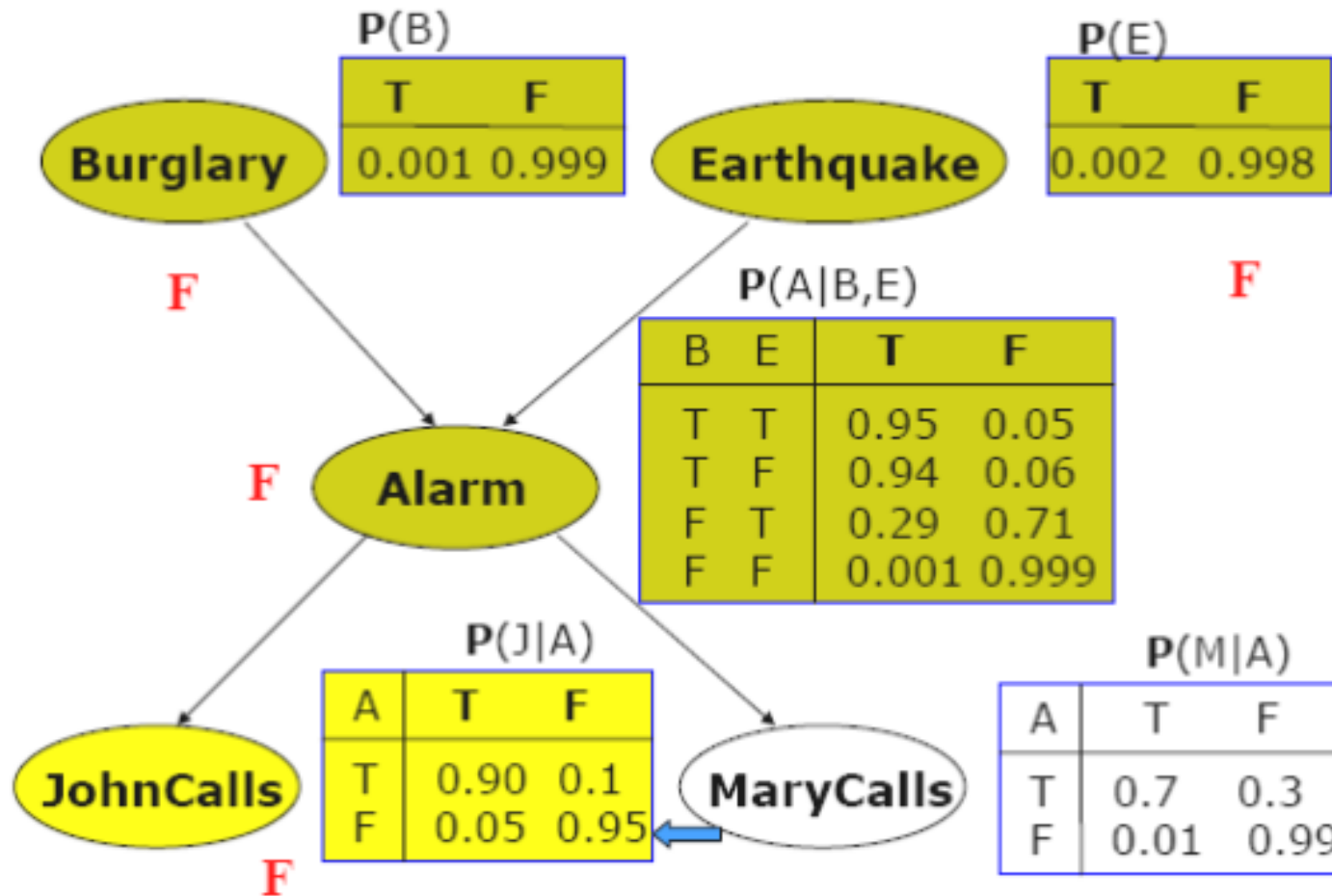
BN Sampling Example



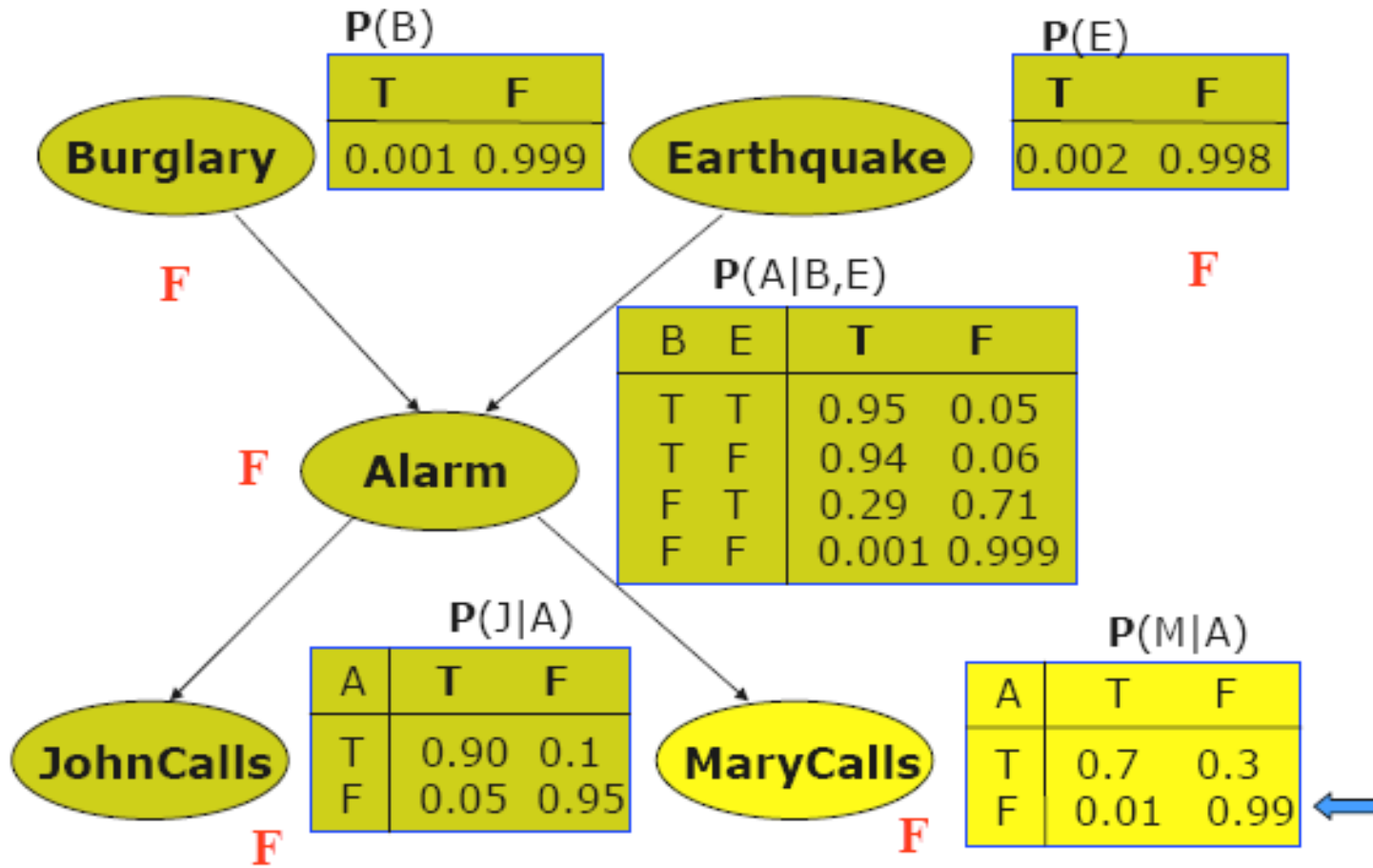
BN Sampling Example



BN Sampling Example



BN Sampling Example



Rejection Sampling

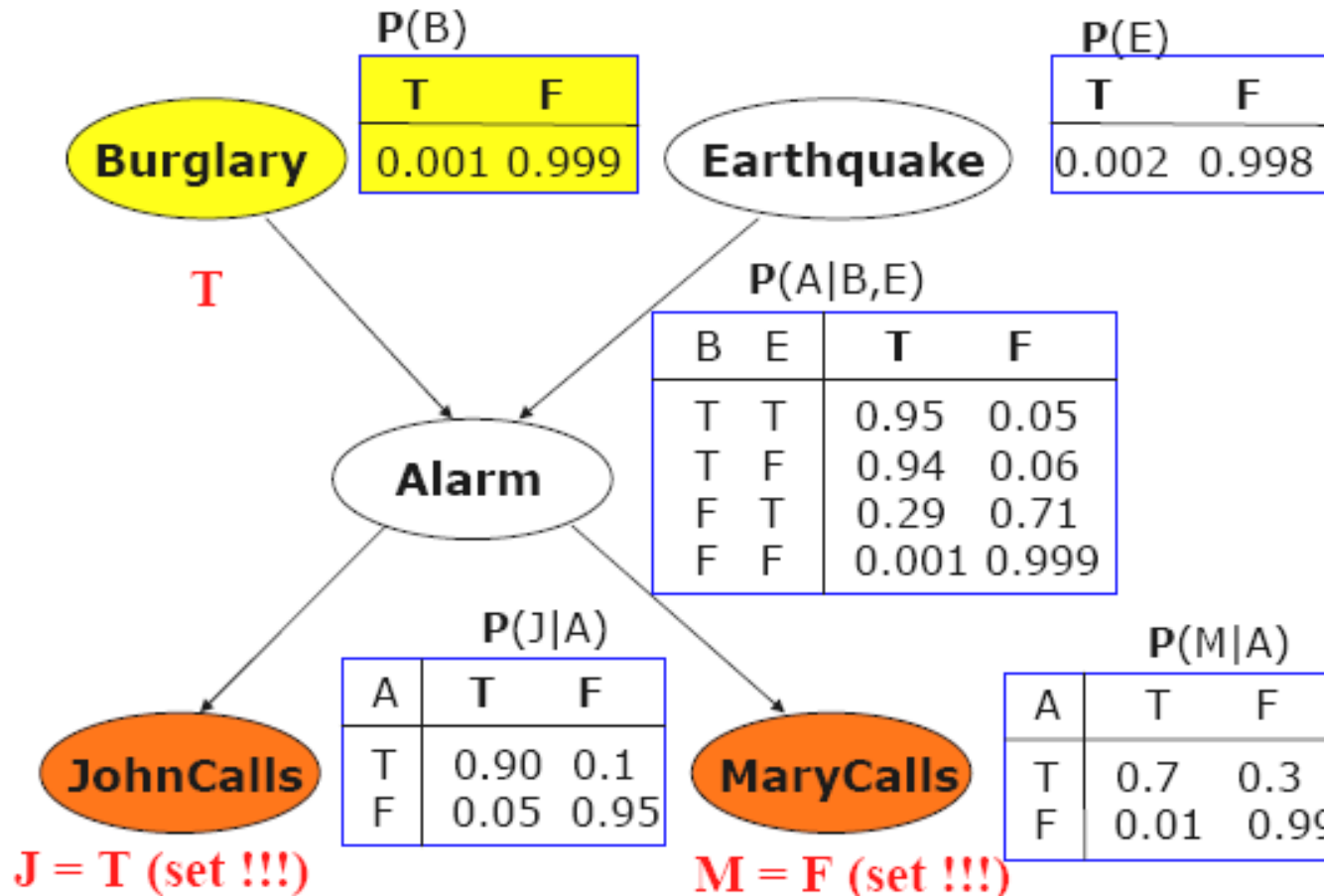
Rejection sampling:

- Generate sample for the full joint by sampling BN
- Use only samples that agree with the condition, the remaining samples are rejected
- Problem: many samples can be rejected

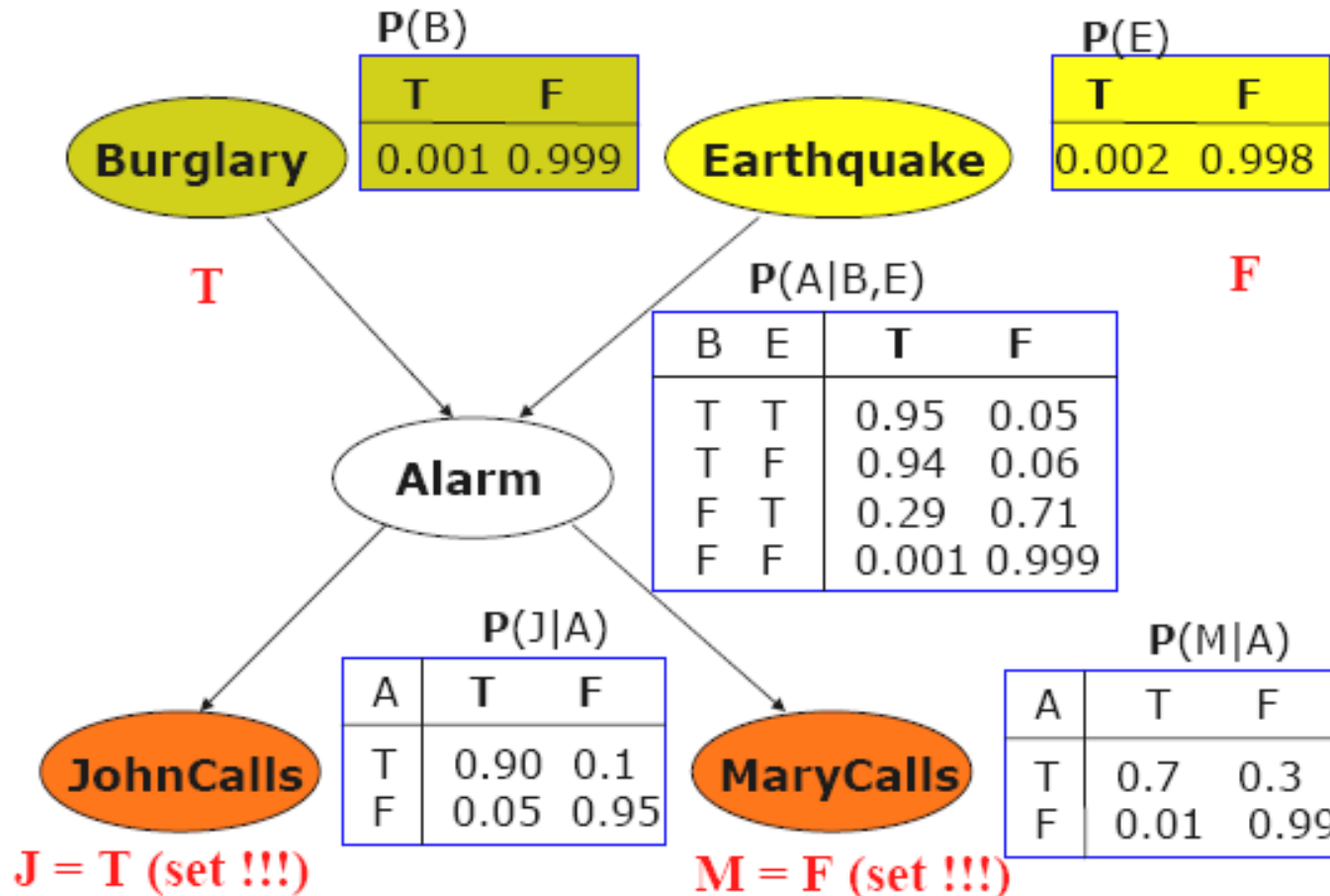
Likelihood weighting

- Avoids inefficiencies of rejection sampling
- Idea: generate only samples consistent with an evidence (or conditioning event)
- If the value is set by evidence, there is no sampling
- Problem: using simple counts is not enough since these may occur with different probabilities
- Likelihood weighting: with every sample keep a weight with which it should count towards the estimate

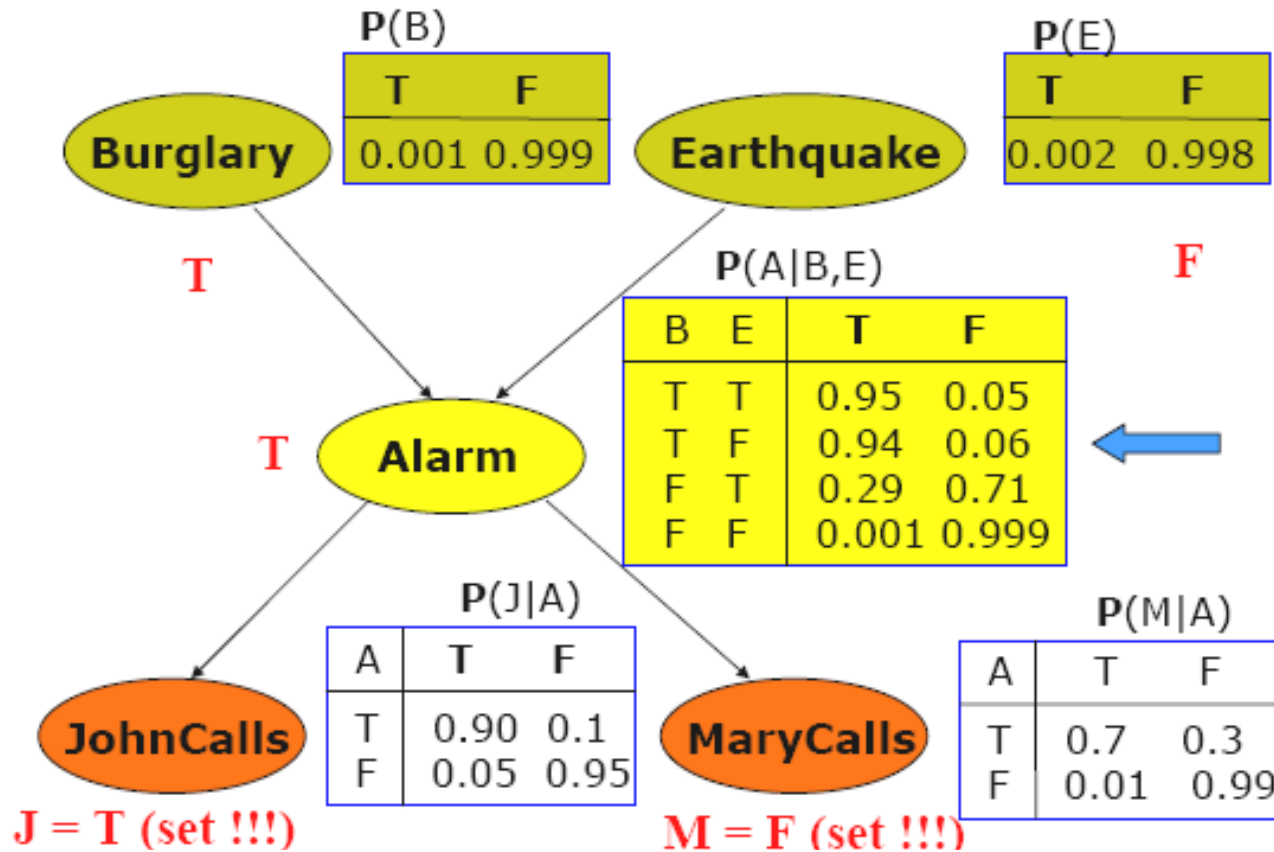
Likelihood weighting Example



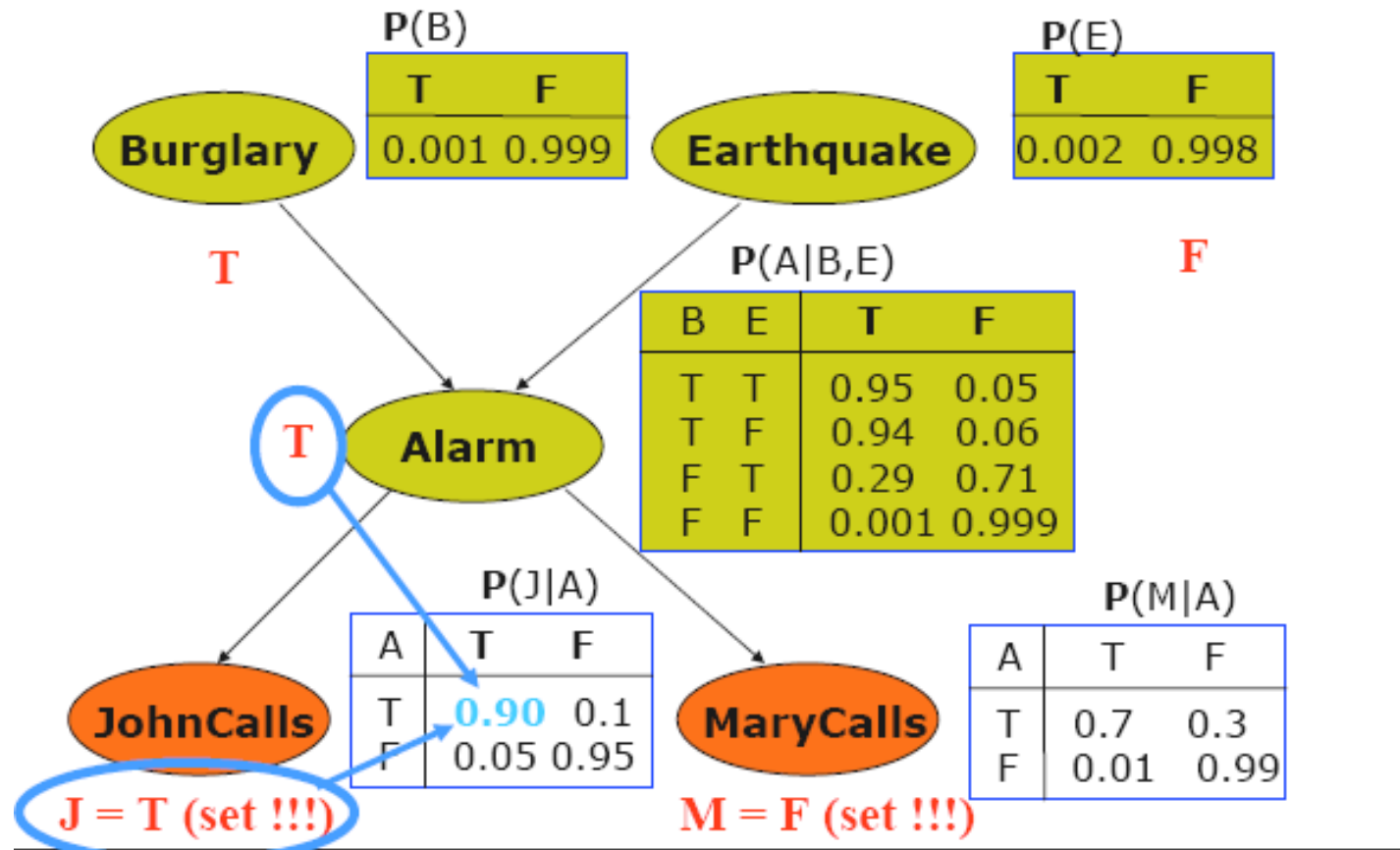
Likelihood weighting Example



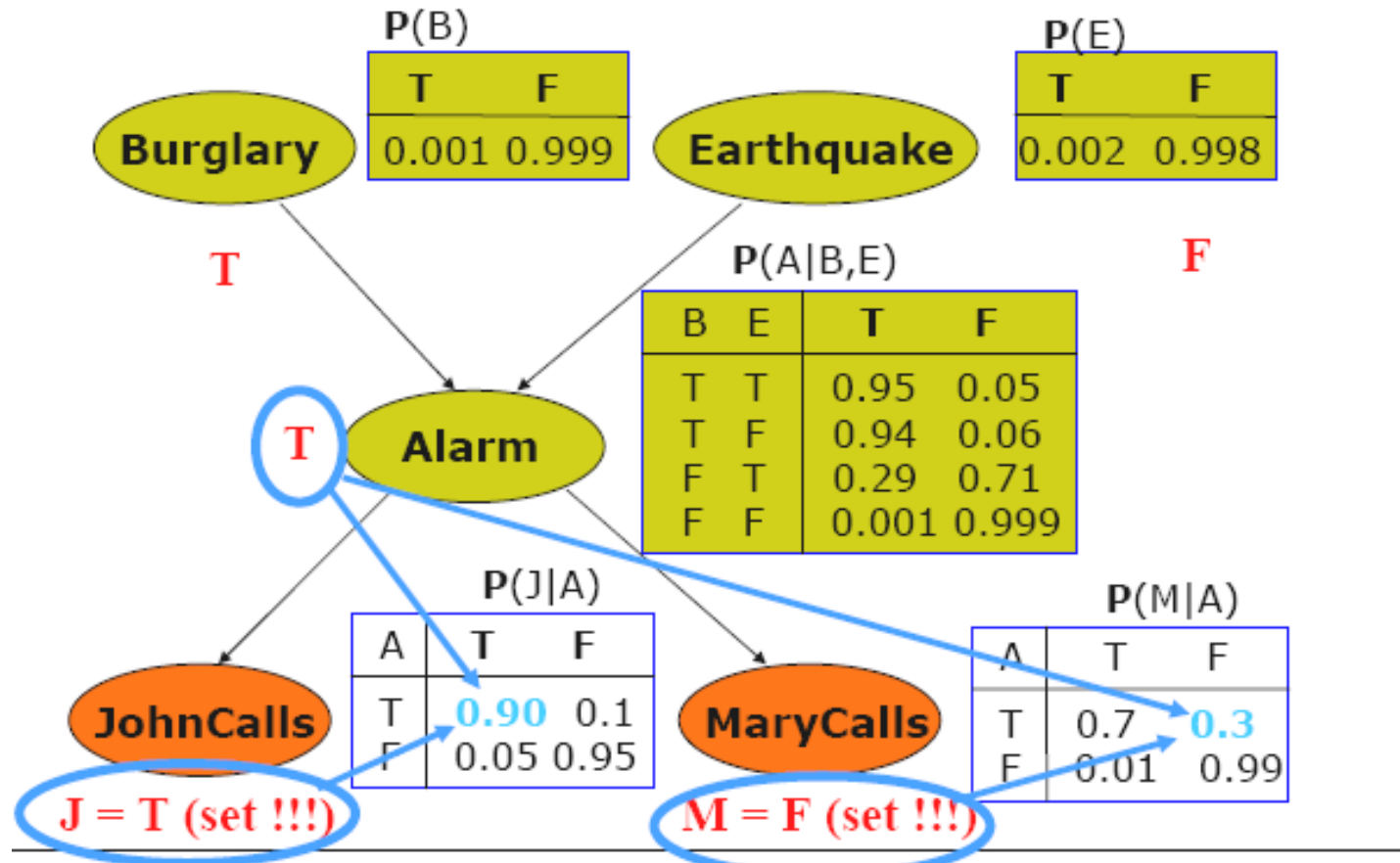
Likelihood weighting Example



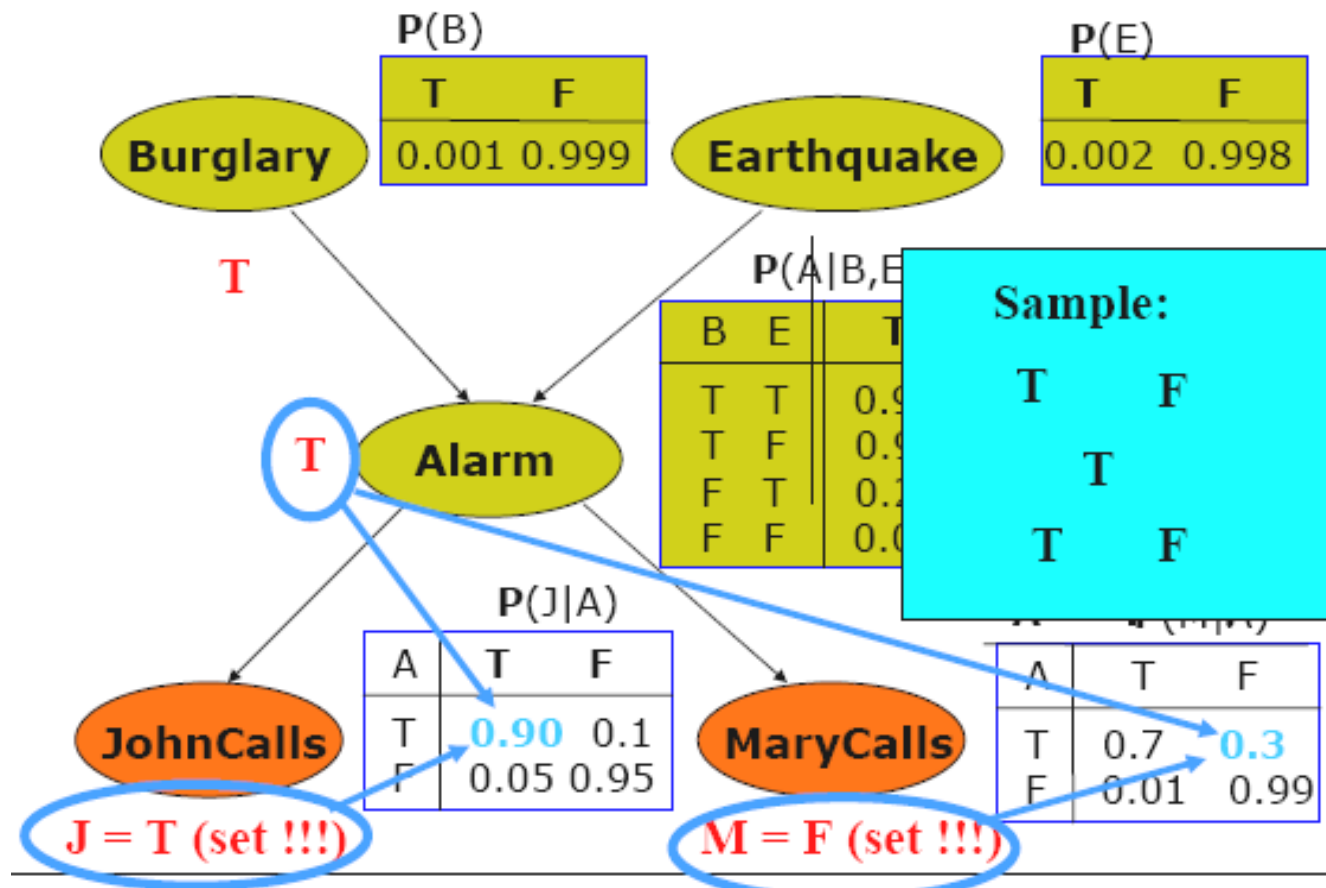
Likelihood weighting Example



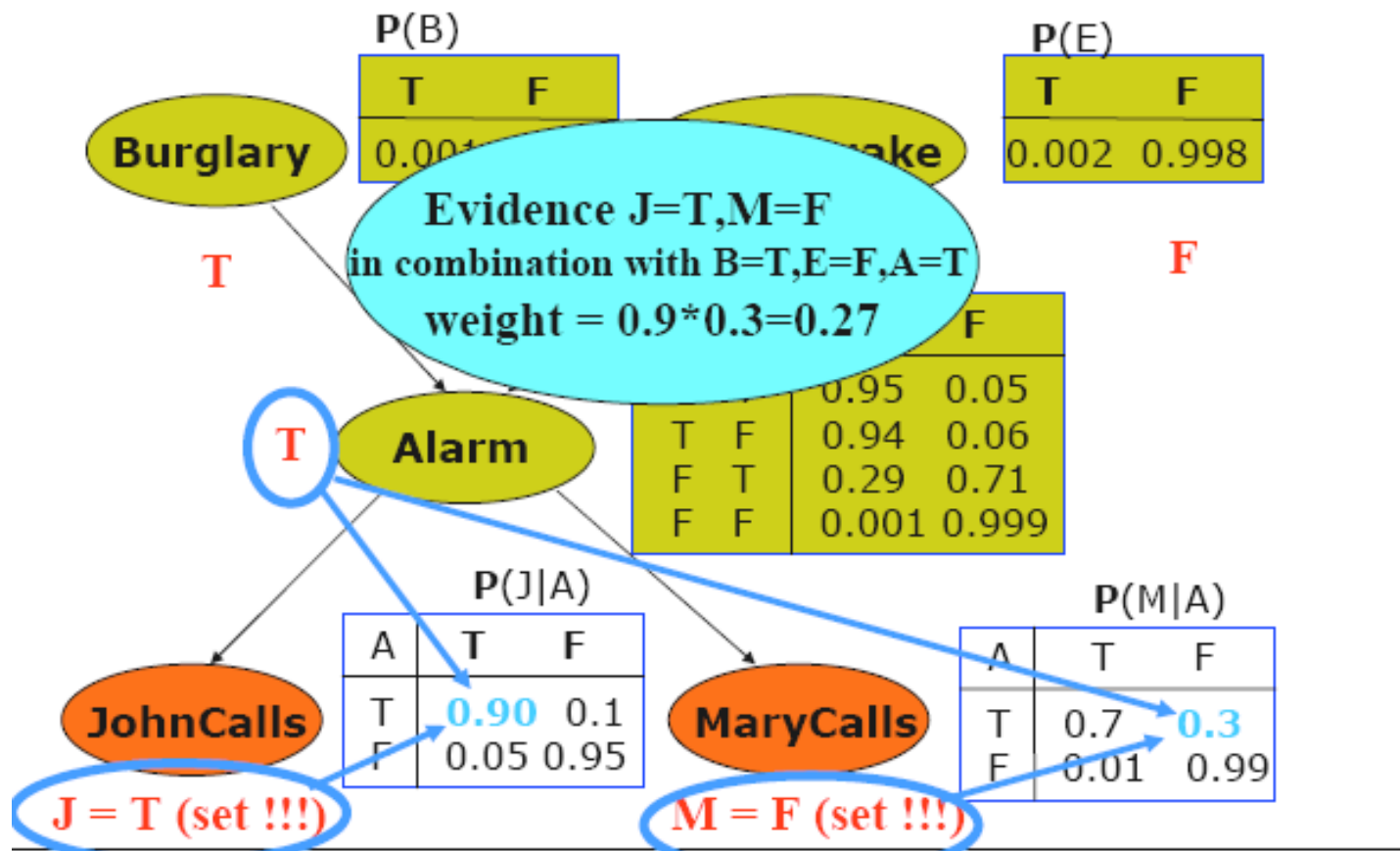
Likelihood weighting Example



Likelihood weighting Example

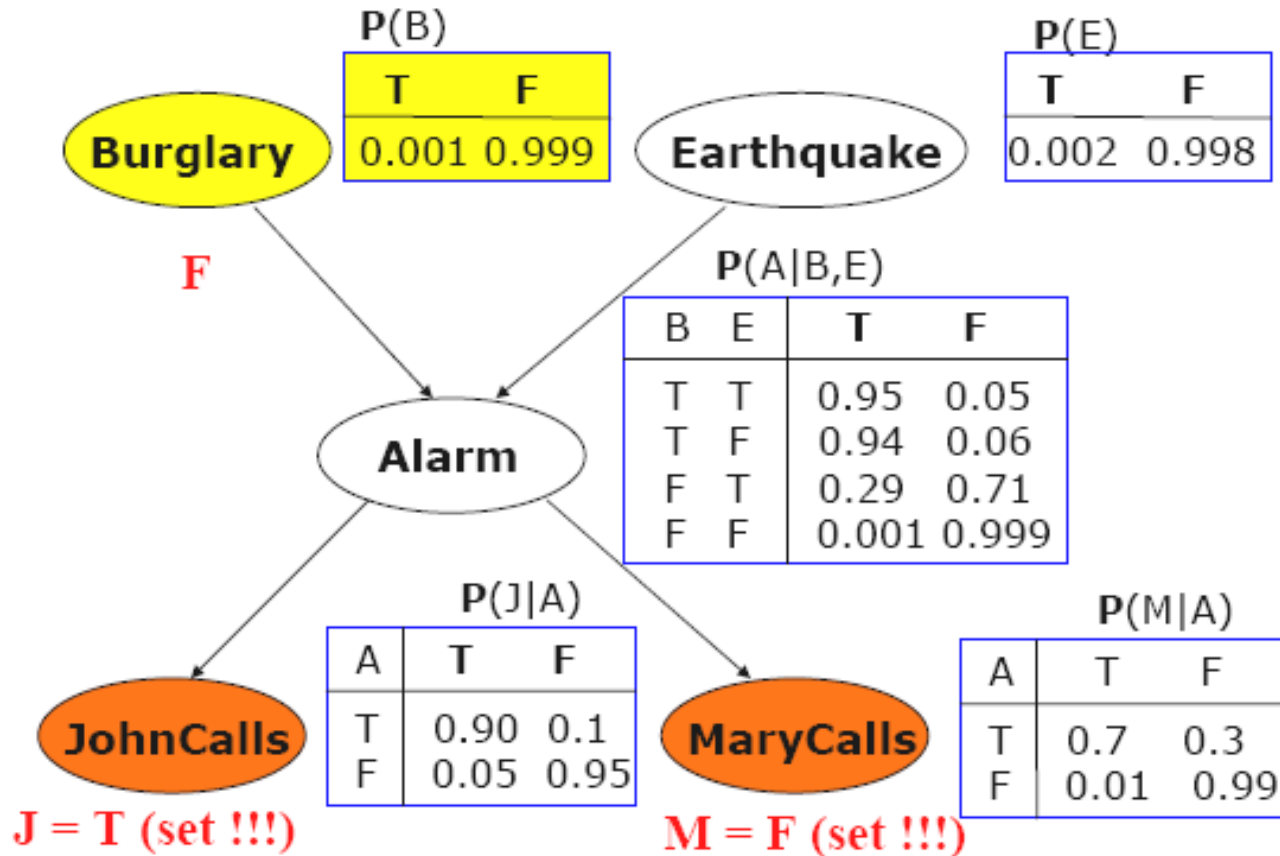


Likelihood weighting Example



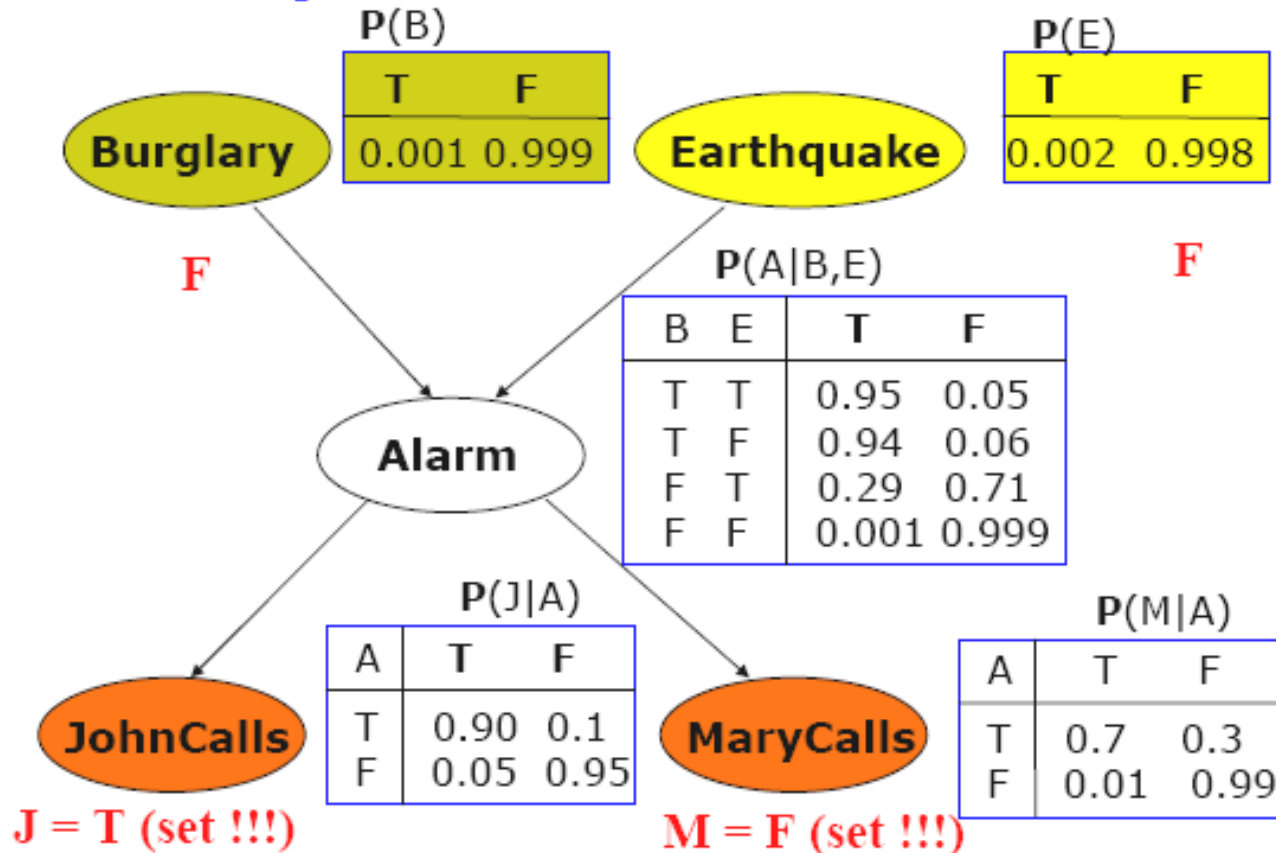
Likelihood weighting Example

Second sample



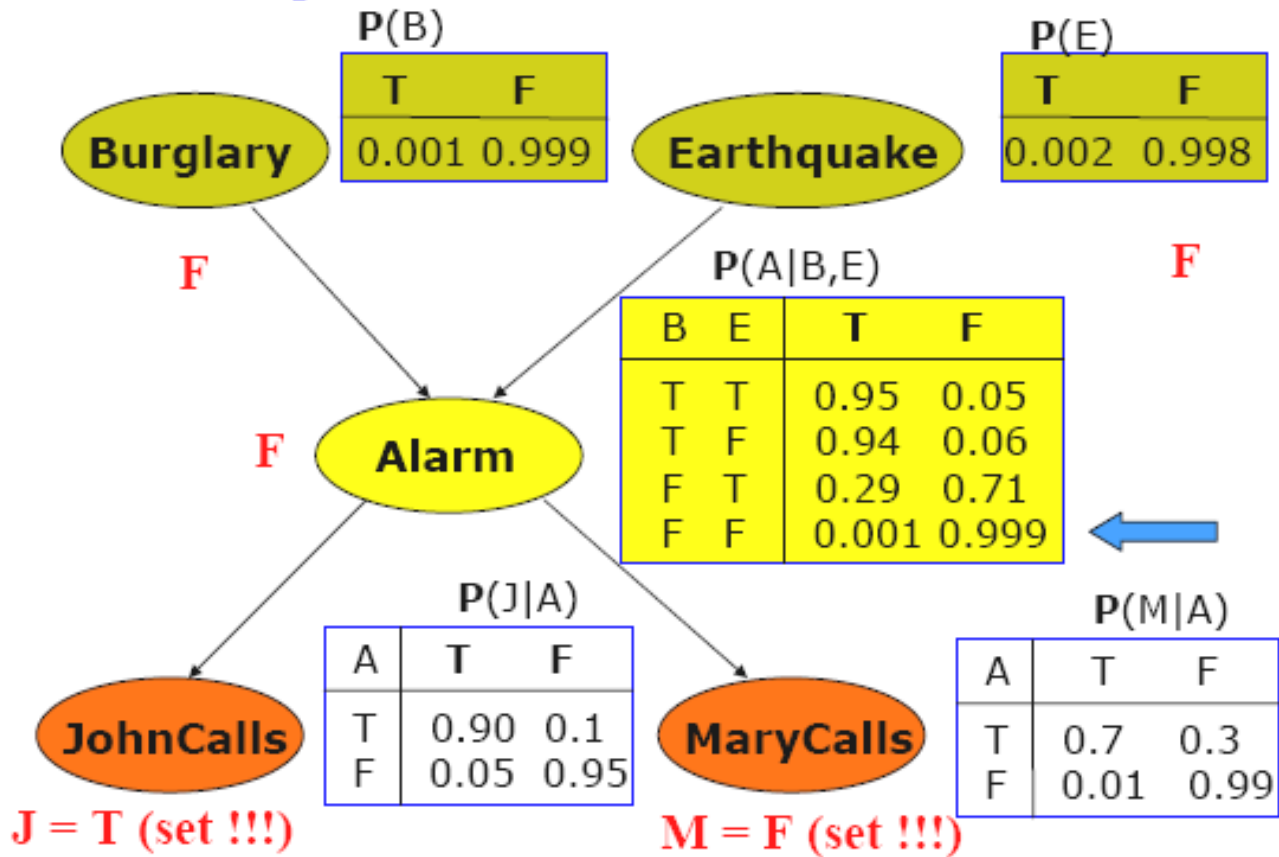
Likelihood weighting Example

Second sample



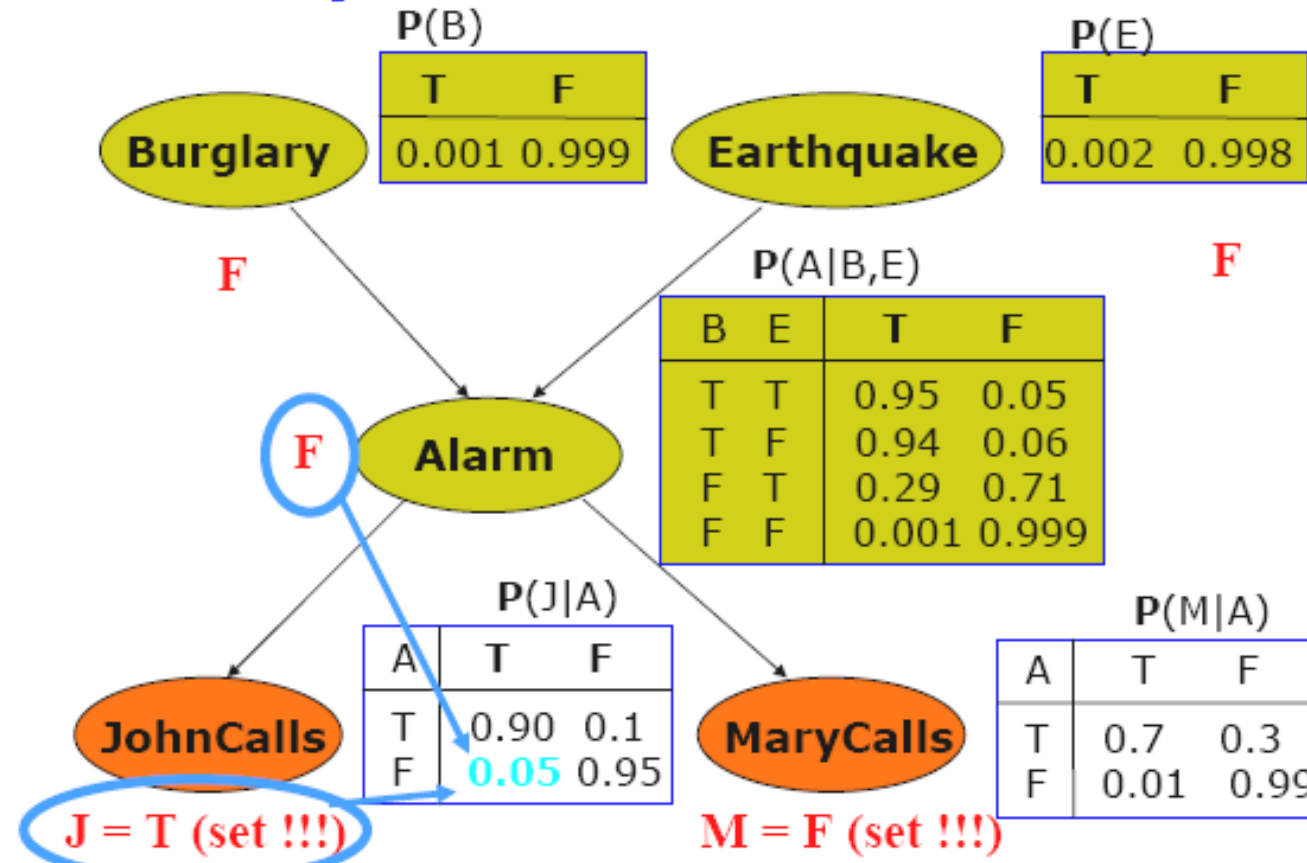
Likelihood weighting Example

Second sample



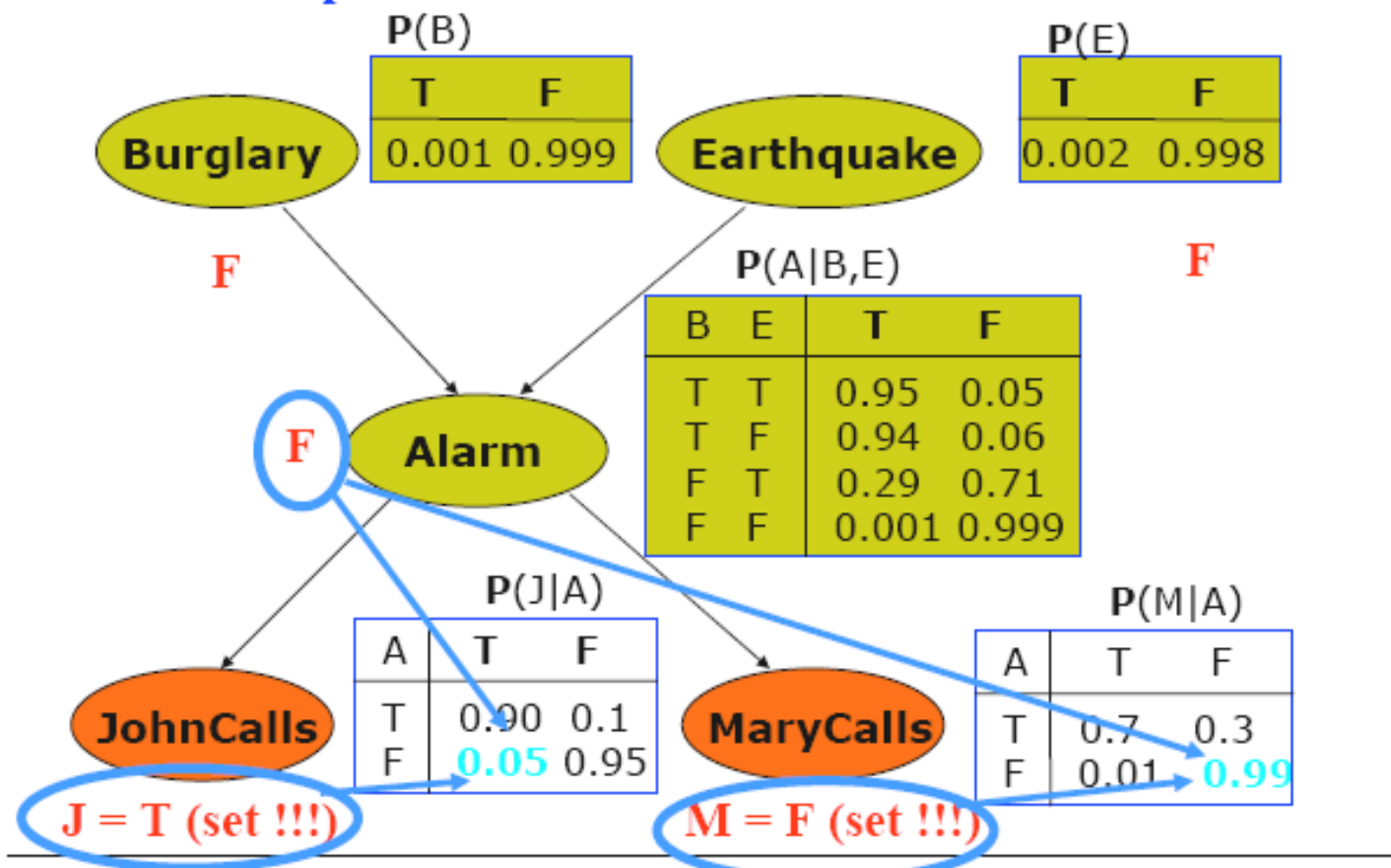
Likelihood weighting Example

Second sample



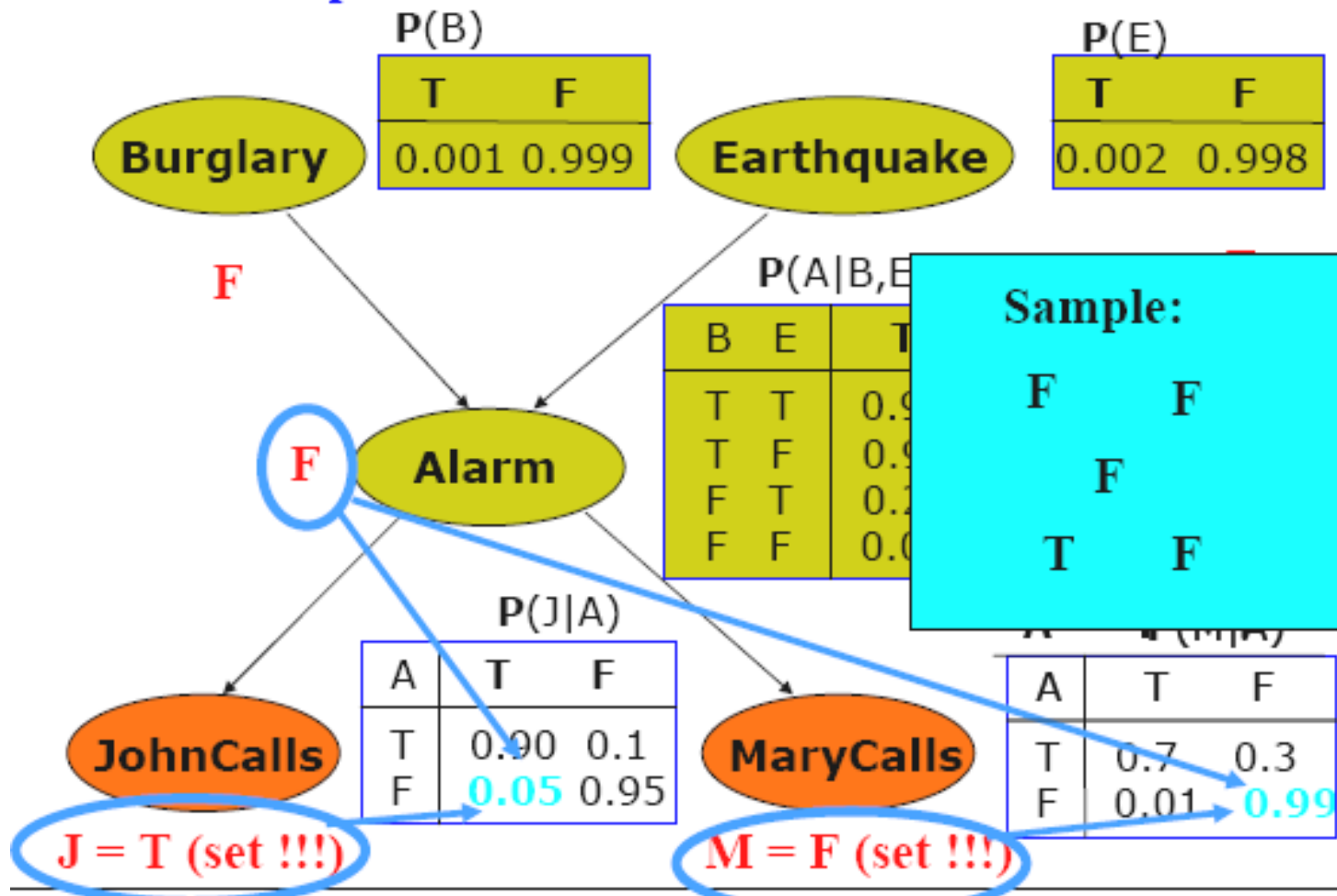
Likelihood weighting Example

Second sample



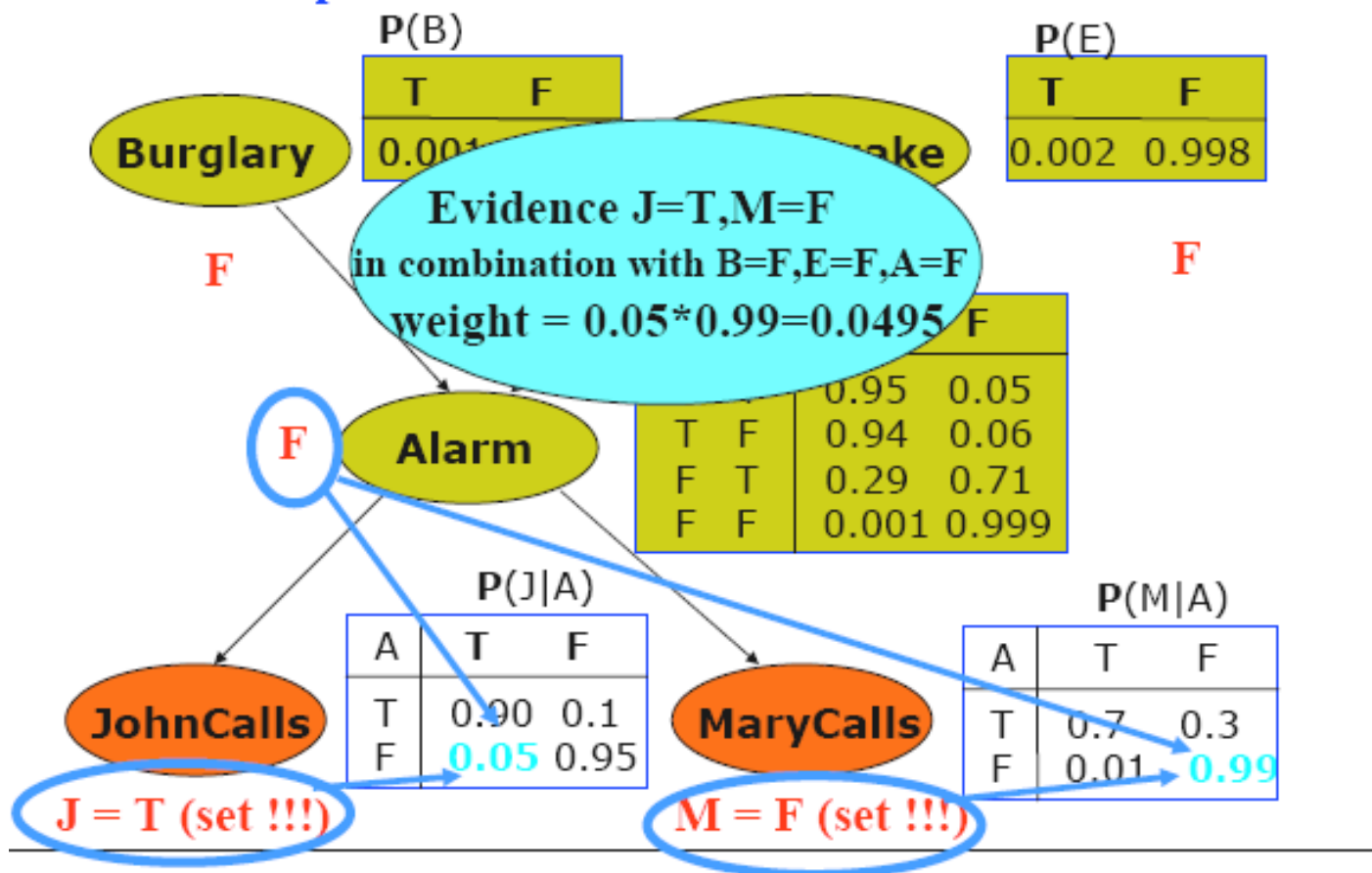
Likelihood weighting Example

Second sample



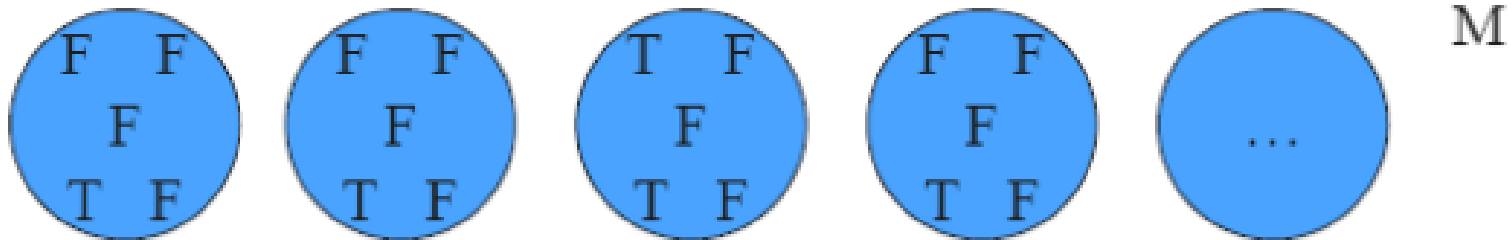
Likelihood weighting Example

Second sample



Likelihood Sampling

- Assume we have generated the following M samples:



- If we calculate the estimate:

$$P(B = T | J = T, M = F) = \frac{\#sample_with(B=T)}{\#total_sample}$$

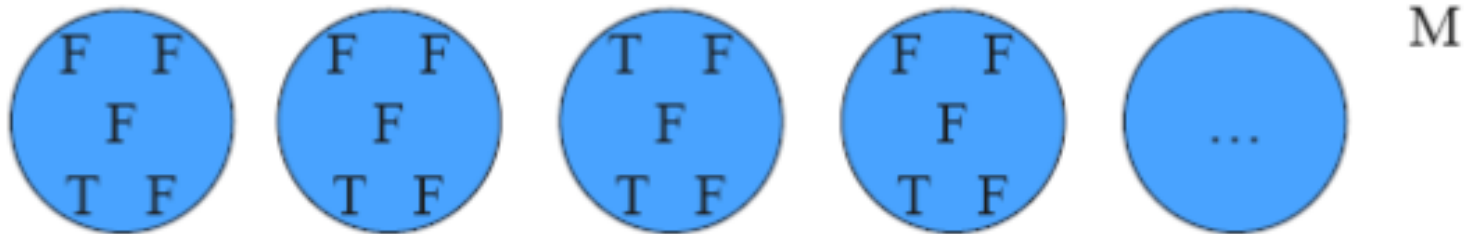
a less likely sample from $P(X)$ may be generated more often.

- For example, sample  is generated more often than in $P(X)$

- So the samples are not consistent with $P(X)$.

Likelihood Sampling

- Assume we have generated the following M samples:



How to make the samples consistent?

Weight each sample by probability with which it agrees with the conditioning evidence $P(e)$.



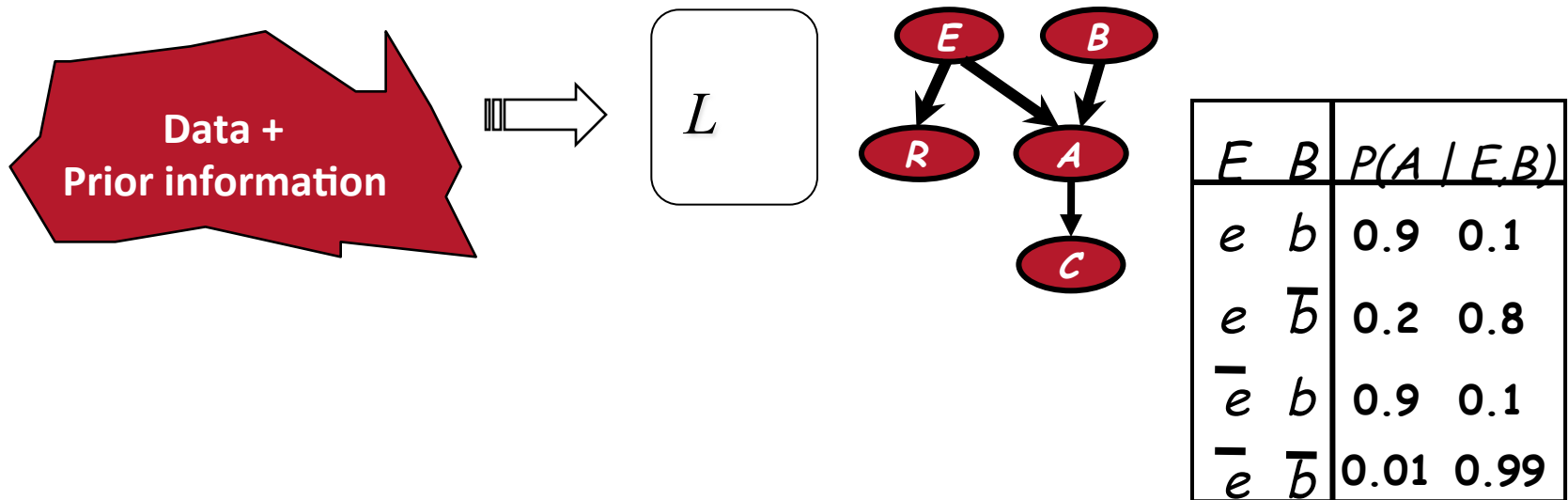
Likelihood Weighting

- How to compute weights for the sample?
- Assume the query $P(B = T \mid J = T, M = F)$
- Likelihood weighting:
 - **With every sample keep a weight with which it should count towards the estimate**

$$\tilde{P}(B = T \mid J = T, M = F) = \frac{\sum_{i=1}^M 1\{B^{(i)} = T\} w^{(i)}}{\sum_{i=1}^M w^{(i)}}$$

$$\tilde{P}(B = T \mid J = T, M = F) = \frac{\sum_{\text{samples with } B=T \text{ and } J=T, M=F} w_{B=T}}{\sum_{\text{samples with any value of } B \text{ and } J=T, M=F} w_{B=x}}$$

Learning Bayesian networks



The Learning Problem

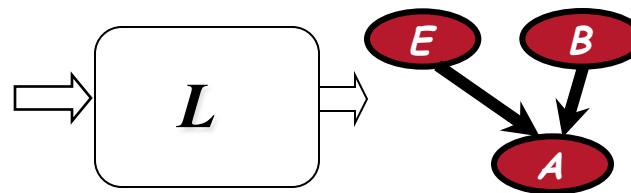
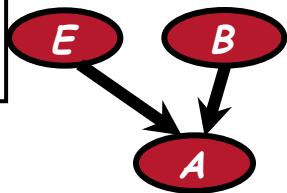
	Known Structure	Unknown Structure
Complete Data	<p>Statistical parameter estimation (closed-form eq.)</p>	<p>Discrete optimization over structures (discrete search)</p>
Incomplete Data	<p>Parametric optimization (EM, gradient descent...)</p>	<p>Combined (Structural EM, mixture models...)</p>

Learning Problem

	Known Structure	Unknown Structure
Complete Data	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete Data	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)

E	B	$P(A E, B)$	
e	b	?	?
e	\bar{b}	?	?
\bar{e}	b	?	?
\bar{e}	\bar{b}	?	?

- E, B, A
- $\langle Y, N, N \rangle$
- $\langle Y, Y, Y \rangle$
- $\langle N, N, Y \rangle$
- $\langle N, Y, Y \rangle$
- \vdots
- $\langle N, Y, Y \rangle$



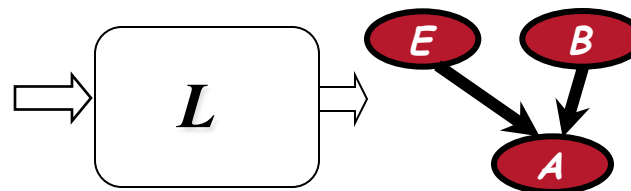
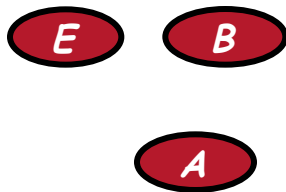
E	B	$P(A E, B)$	
e	b	0.9	0.1
e	\bar{b}	0.2	0.8
\bar{e}	b	0.9	0.1
\bar{e}	\bar{b}	0.01	0.99

Learning Problem

	Known Structure	Unknown Structure
Complete Data	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete Data	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)

E	B	$P(A E, B)$	
e	b	?	?
e	\bar{b}	?	?
\bar{e}	b	?	?
\bar{e}	\bar{b}	?	?

- E, B, A
- $\langle Y, N, N \rangle$
- $\langle Y, Y, Y \rangle$
- $\langle N, N, Y \rangle$
- $\langle N, Y, Y \rangle$
- \vdots
- $\langle N, Y, Y \rangle$



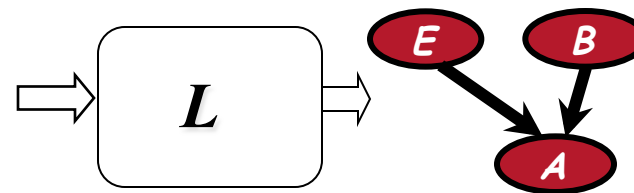
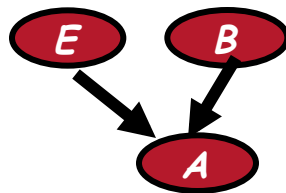
E	B	$P(A E, B)$	
e	b	0.9	0.1
e	\bar{b}	0.2	0.8
\bar{e}	b	0.9	0.1
\bar{e}	\bar{b}	0.01	0.99

Learning Problem

	Known Structure	Unknown Structure
Complete Data	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete Data	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)

E	B	$P(A E, B)$	
e	b	?	?
e	\bar{b}	?	?
\bar{e}	b	?	?
\bar{e}	\bar{b}	?	?

- E, B, A
- $\langle Y, N, ? \rangle$
- $\langle Y, ?, Y \rangle$
- $\langle N, N, Y \rangle$
- $\langle ?, Y, Y \rangle$
- \vdots
- $\langle N, ?, Y \rangle$



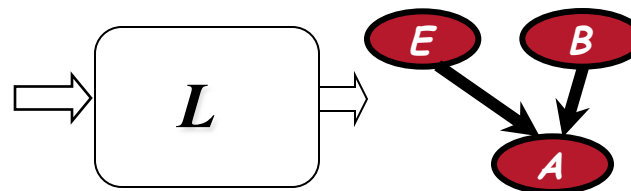
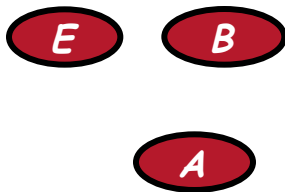
E	B	$P(A E, B)$	
e	b	0.9	0.1
e	\bar{b}	0.2	0.8
\bar{e}	b	0.9	0.1
\bar{e}	\bar{b}	0.01	0.99

Learning Problem

	Known Structure	Unknown Structure
Complete Data	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete Data	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)


E	B	$P(A E, B)$	
e	b	?	?
e	\bar{b}	?	?
\bar{e}	b	?	?
\bar{e}	\bar{b}	?	?

- E, B, A
- $\langle Y, N, ? \rangle$
- $\langle Y, ?, Y \rangle$
- $\langle N, N, Y \rangle$
- $\langle ?, Y, Y \rangle$
- \vdots
- $\langle N, ?, Y \rangle$



E	B	$P(A E, B)$	
e	b	0.9	0.1
e	\bar{b}	0.2	0.8
\bar{e}	b	0.9	0.1
\bar{e}	\bar{b}	0.01	0.99

Learning Bayesian Networks

	Known Structure	Unknown Structure
Complete data		
Incomplete data		

- » Parameter learning: Complete data (Review)
 - Statistical parametric fitting
 - Maximum likelihood estimation
 - Bayesian inference
- Parameter learning: Incomplete data
- Structure learning: Complete data
- Application: classification
- Structure learning: Incomplete data

Learning Parameters

- Estimation relies on sufficient statistics
 - For multinomial these are of the form $N(x_i, pa_i)$
 - Parameter estimation

$$\hat{\theta}_{x_i | pa_i} = \frac{N(x_i, pa_i)}{N(pa_i)}$$

MLE

$$\tilde{\theta}_{x_i | pa_i} = \frac{\alpha(x_i, pa_i) + N(x_i, pa_i)}{\alpha(pa_i) + N(pa_i)}$$

Bayesian (Dirichlet)

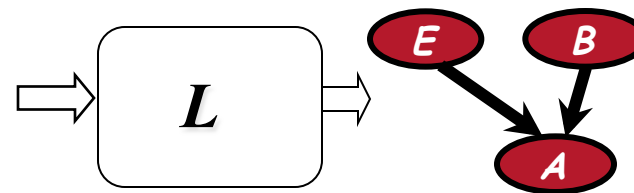
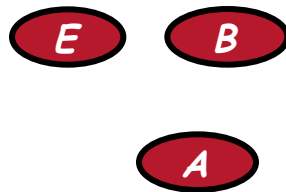
- Bayesian methods also require choice of priors
- Both MLE and Bayesian estimates are asymptotically equivalent and consistent but the latter work better with small samples
- Both can be implemented in an on-line manner by accumulating sufficient statistics

Learning Problem

	Known Structure	Unknown Structure
Complete Data	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete Data	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)

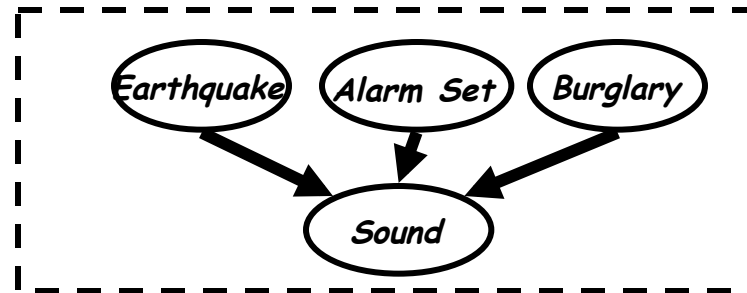
E	B	$P(A E, B)$	
e	b	?	?
e	\bar{b}	?	?
\bar{e}	b	?	?
\bar{e}	\bar{b}	?	?

- E, B, A
- $\langle Y, N, N \rangle$
- $\langle Y, Y, Y \rangle$
- $\langle N, N, Y \rangle$
- $\langle N, Y, Y \rangle$
- \vdots
- $\langle N, Y, Y \rangle$

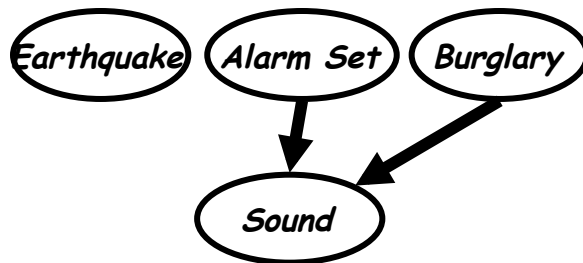


E	B	$P(A E, B)$	
e	b	0.9	0.1
e	\bar{b}	0.2	0.8
\bar{e}	b	0.9	0.1
\bar{e}	\bar{b}	0.01	0.99

Why do we need accurate structure?

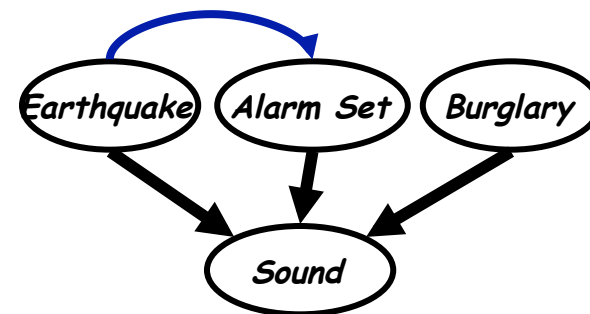


Missing an arc



- Cannot be compensated for by fitting parameters
- Incorrect independence assumptions

Extraneous arc



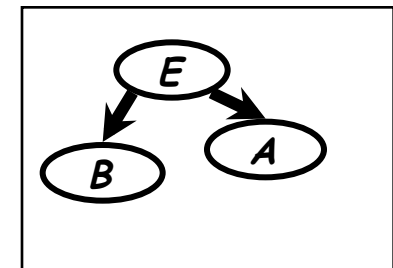
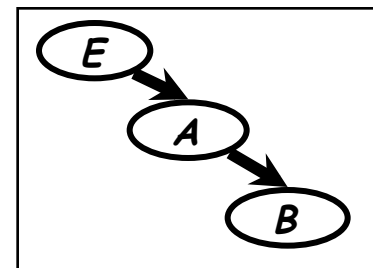
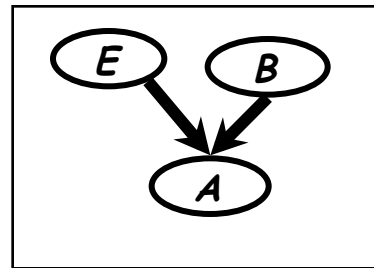
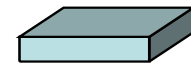
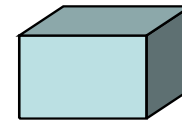
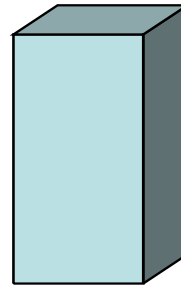
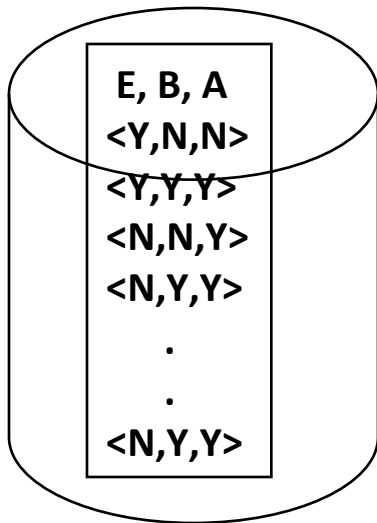
- Increases the number of parameters to be estimated
- Incorrect independence assumptions

Approaches to BN Structure Learning

- Score based methods
 - assign a score to each candidate BN structure using a suitable scoring function
 - Search the space of candidate network structures for a BN structure with the maximum score
- Independence testing based methods
 - Use independence tests to determine the structure of the network

Score-based BN Structure Learning

Define a scoring function that evaluates how well a structure matches the data



Search for a structure that maximizes the score

Basic idea: Minimum description length (MDL) principle

$$\begin{aligned}h_{MAP} &= \arg \max_{h \in H} P(h | D) \\ &= \arg \max_{h \in H} \frac{P(D | h)P(h)}{P(D)} \\ &= \arg \max_{h \in H} P(D | h)P(h) \\ h_{MAP} &= \arg \min_{h \in H} (-\log P(D | h) - \log P(h)) \\ h_{MDL} &= \arg \min_{h \in H} (C_{D|h}(D | h) + C_h(h))\end{aligned}$$

We need to design a scoring function that minimizes the sum of the description length of the hypothesis and the description length of the data given the hypothesis.

In this case, the hypothesis is a Bayesian network which represents a joint probability distribution

Scoring function

A BN scoring function consists of

- A term that corresponds to the number of bits needed to encode the BN structure and parameters
- A term that corresponds to the number of bits needed to encode the data given the BN

Encoding a Bayesian Network

It suffices to

- list the parents of each node
- record the conditional probabilities associated with each node

Consider a *BN* with n variables.

- Consider a node i with k_i parents.
- We need $k_i \log_2 n$ bits to list its parents.
- Suppose the node i (variable X_i) takes s_i distinct values.
- Suppose the j th parent takes s_j distinct values.
- Suppose we use d bits to store each conditional probability.

Encoding a Bayesian Network

- Under the encoding scheme described, the description length of a particular Bayesian network is given by

-

$$\sum_{i=1}^n \left(k_i \log_2(n) + d(s_i - 1) \prod_{X_j \in \text{Parents}(X_i)} s_j \right)$$

Encoding the Data

- Suppose we have M independent observations (instantiations) of the random variables X_1, \dots, X_n

Let V_i be the domain of random variable X_i

Each observation corresponds to an atomic event

$$e_k \in V_1 \times V_2 \times \dots \times V_n$$

Let p_k be the probability of e_k

When M is large, we expect $M p_k$ occurrences of e_k among the M observations. Under optimal encoding, the number of bits needed to encode the data is

$$-M \sum_{e_k \in V_1 \times \dots \times V_n} p_k \log_2(p_k)$$

Encoding the Data ..using a Bayesian network

- But.. We do not know p_k - the probability of e_k !
- What we have instead is a Bayesian network which assigns a probability q_k to e_k
- When we use the learned network to encode the data, the number of bits needed to encode the network (and hence the data using the network) is

$$-M \sum_{e_k \in V_1 \times \dots \times V_n} p_k \log_2(q_k)$$

Encoding the Data ..using a Bayesian network

Theorem (Gibbs):

$$-M \sum_{e_k \in V_1 \times \dots \times V_n} p_k \log_2(p_k) \leq -M \sum_{e_k \in V_1 \times \dots \times V_n} p_k \log_2(q_k)$$

- with equality holding if and only if $\forall i \ p_i = q_i$
- Number of bits needed to encode the data if true probabilities of each atomic event are known is less than or equal to the number of bits needed to encode the data using a code based on the estimated probabilities.

Putting the two together

- MDL principle recommends minimizing the sum of the encoding lengths of the model (Bayes network) and the encoding length of the data using the model

$$\sum_{i=1}^N \left(k_i \log_2(n) + d(s_i - 1) \prod_{X_j \in \text{Parents}(X_i)} s_j \right) - M \sum_{e_k \in V_1 \times \dots \times V_n} p_k \log_2(q_k)$$

- Problems with evaluating the second term:
 - We do not know the probabilities p_k
 - The second term requires summation over all atomic events (all instantiations of the n random variables)

[Lam and Bacchus, 1994]

Kullback-Leibler Divergence to the rescue!

- Let P and Q be two probability distributions over the same event space such that an event e_i is assigned probability p_i by P and q_i by Q

$$KL(P \parallel Q) = \sum_k p_k \log \left(\frac{p_k}{q_k} \right)$$

$$= \sum_k p_k (\log p_k - \log q_k)$$

$$KL(P \parallel Q) \geq 0$$

$$KL(P \parallel Q) = 0 \text{ iff } P = Q$$

Kullback-Leibler Divergence to the rescue!

- **Theorem:** The encoding length of the data (distributed according to P) given the model (distribution Q) is a monotonically increasing function of $KL(P \parallel Q)$
- **Proof:** From Gibbs Theorem and the definition of KL divergence
- Hence, we can use the estimated KL divergence as a proxy for the encoding length of the data (given a model) to score a model.
- We can use local computations over a Bayes network to evaluate

$$KL(P \parallel Q)$$

Applying the MDL Principle

- Exhaustive search over the space of all networks infeasible!
- Evaluating KL-divergence directly is infeasible!
- Hence we need to
 - Resort to a heuristic search to find a network with a near minimal description length
 - Develop a more efficient method of evaluating KL divergence of a candidate network

Heuristic search

A possible search strategy

- There can be between 0 and $n(n-1)/2$ arcs in a DAG with n nodes
- For each possible number of arcs, we search heuristically for networks with low KL divergence
- We then examine the resulting networks and pick one that has minimum description length

Learning Tree-Structured Bayes Networks

- If we measured a distribution P , what is the tree-dependent distribution P_t that best approximates P ?
 - Search Space: All possible spanning trees
 - Goal: From all possible spanning trees find the one closest to P
 - Closeness Measure: Kullback–Leibler divergence
 - Operators/Procedure

Kullback-Leibler divergence

- For probability distributions P and Q of a discrete random variable the K–L divergence of Q from P is defined to be

$$D_{KL}(P, Q) = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

- It can be seen from the definition of the Kullback-Leibler divergence that

$$\begin{aligned} D_{KL}(P, Q) &= - \sum_x P(x) \log(Q(x)) + \sum_x P(x) \log(P(x)) \\ &= H(P, Q) - H(P) \end{aligned}$$

- where $H(P, Q)$ is called the cross entropy of P and Q , and $H(P)$ is the entropy of P .
- Non negative measure (by Gibb's inequality)

Evaluating KL divergence for a network

- Theorem (Chou and Liu, 1969). Suppose we define mutual information between any two nodes X_i and X_j as

$$W(X_i, X_j) = \sum_{(X_i, X_j)} P(X_i, X_j) \log_2 \frac{P(X_i, X_j)}{P(X_i)P(X_j)}$$

- Then the cross entropy $KL(P||Q)$ over all tree-structured distributions is minimized when the graph representing $Q(X_1 .. X_n)$ is a maximum weight spanning tree of the graph where the edge between nodes X_i and X_j is assigned the weight equal to $W(X_i, X_j)$.

Mutual information

- The **mutual information** of 2 random variables is a quantity that measures the mutual dependence of the two variables
- Intuitively, mutual information measures the information that X and Y share.

The algorithm

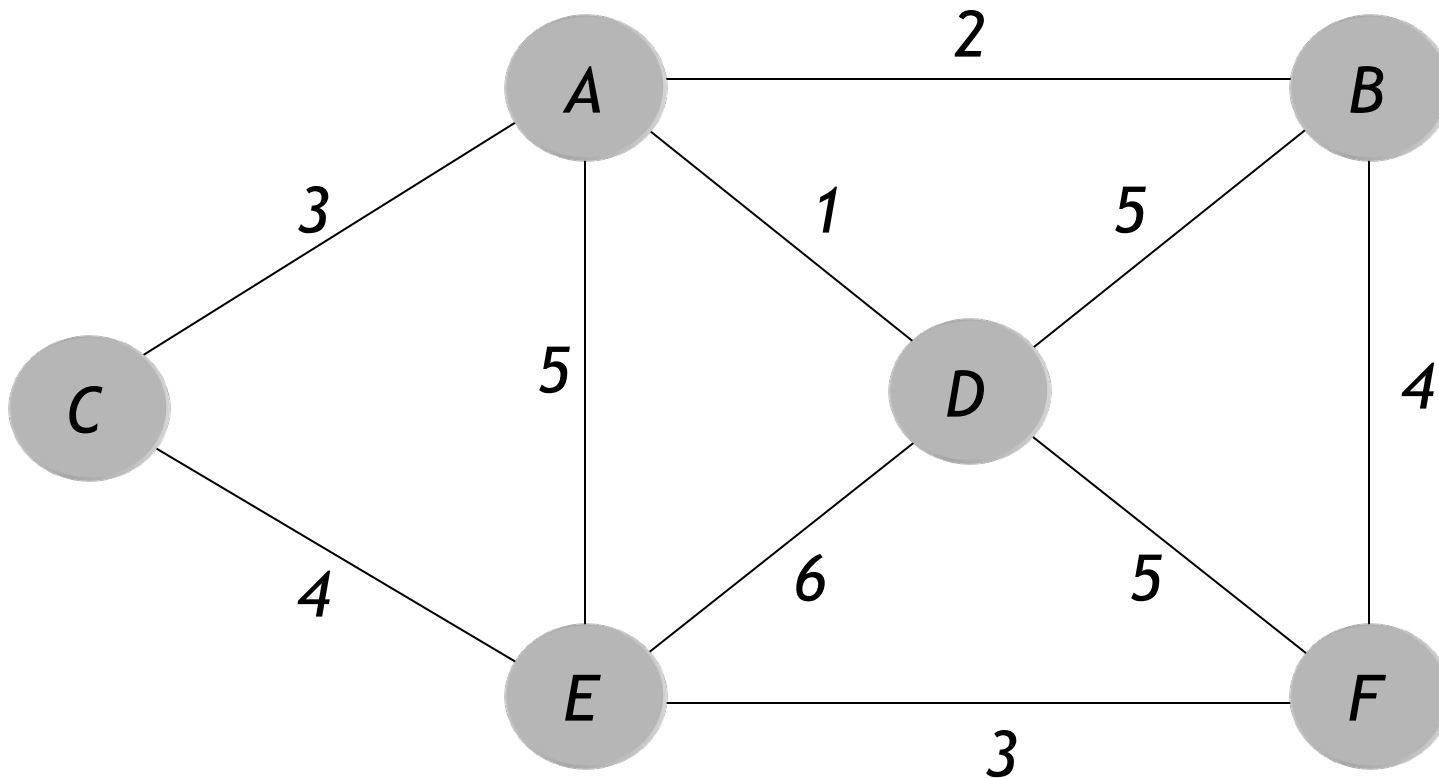
- Use Kruskal to find Maximum spanning tree with weights given by :

$$W(X_i, X_j) = \sum_{(X_i, X_j)} P(X_i, X_j) \log_2 \frac{P(X_i, X_j)}{P(X_i)P(X_j)}$$

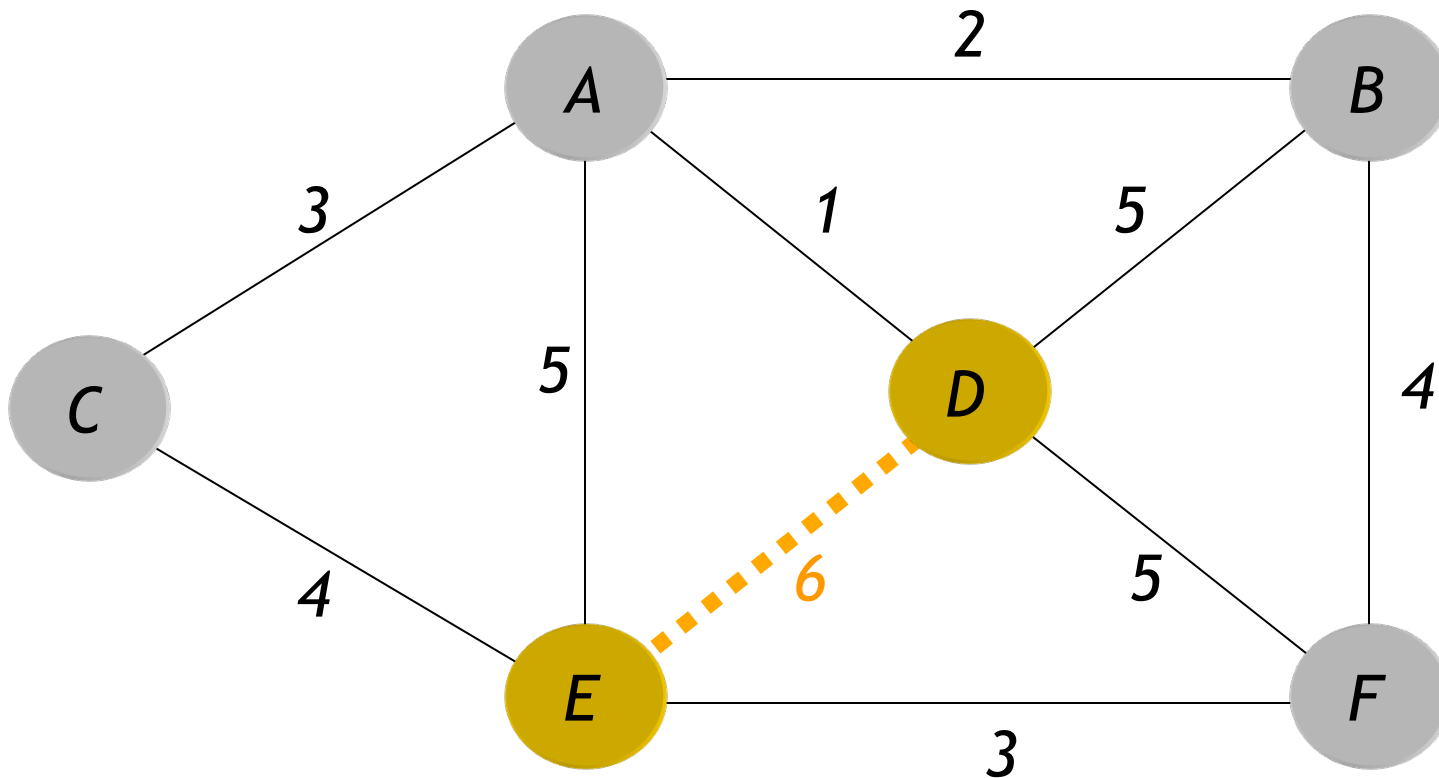
- Compute P_t
 - Select an arbitrary root node and compute

$$P_t = P(X_i | Parent_t(X_i))$$

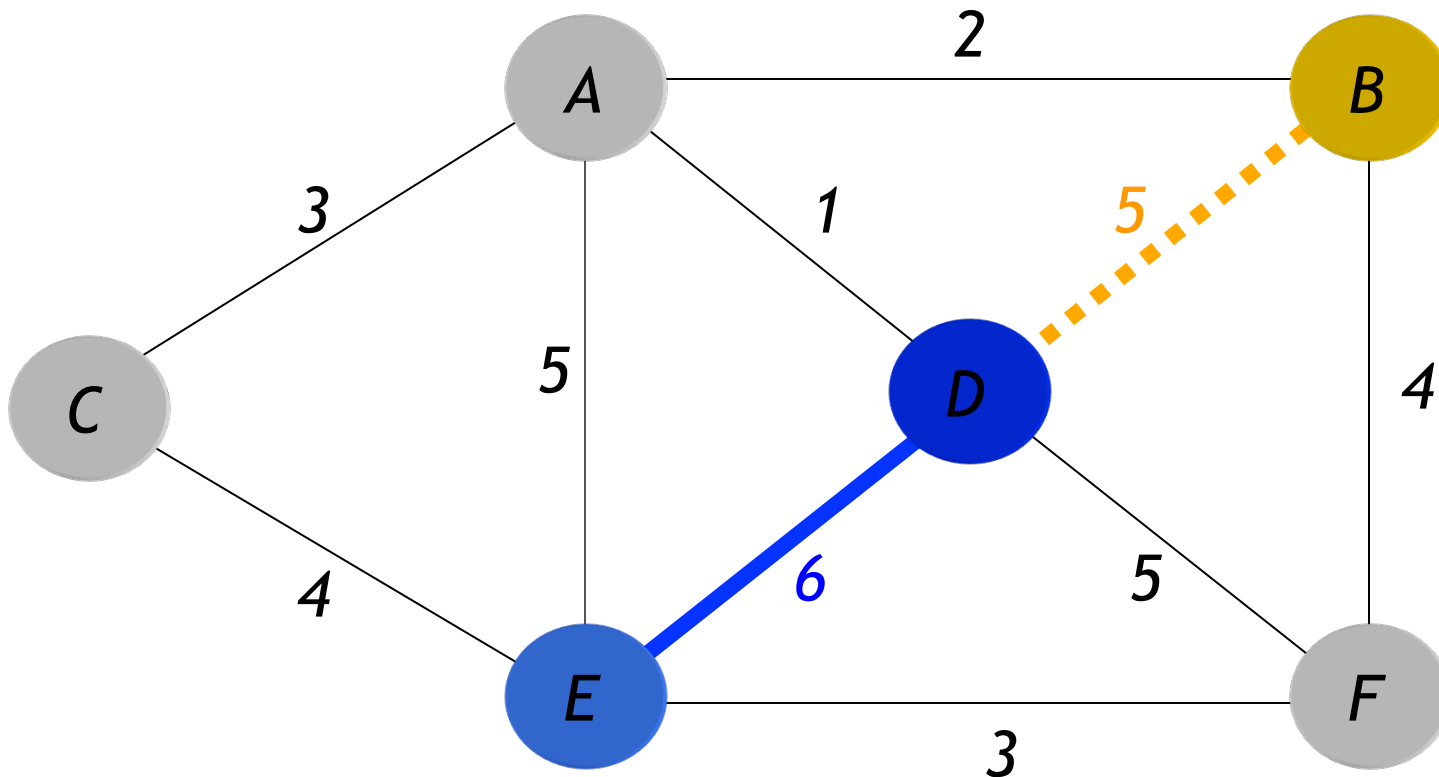
Kruskal Algorithm



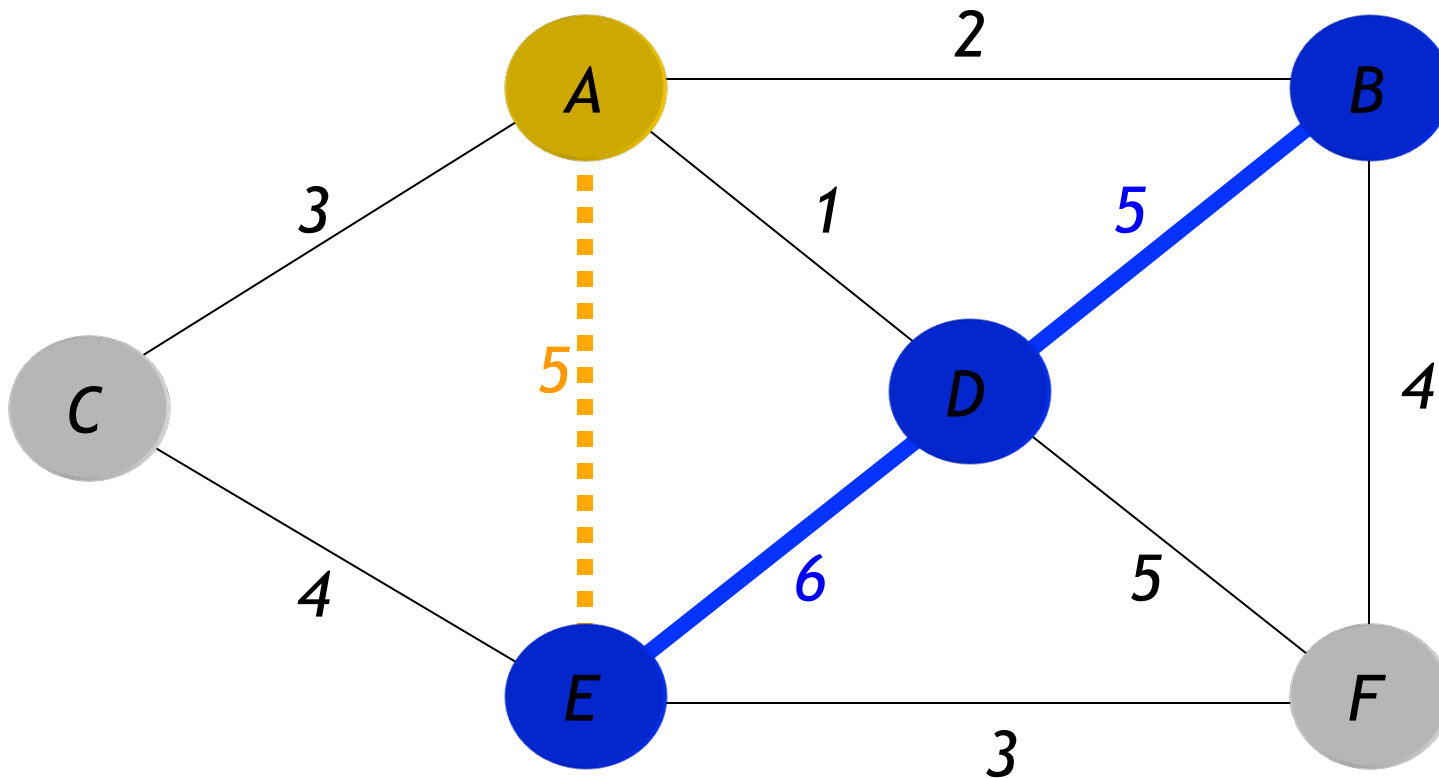
Kruskal Algorithm



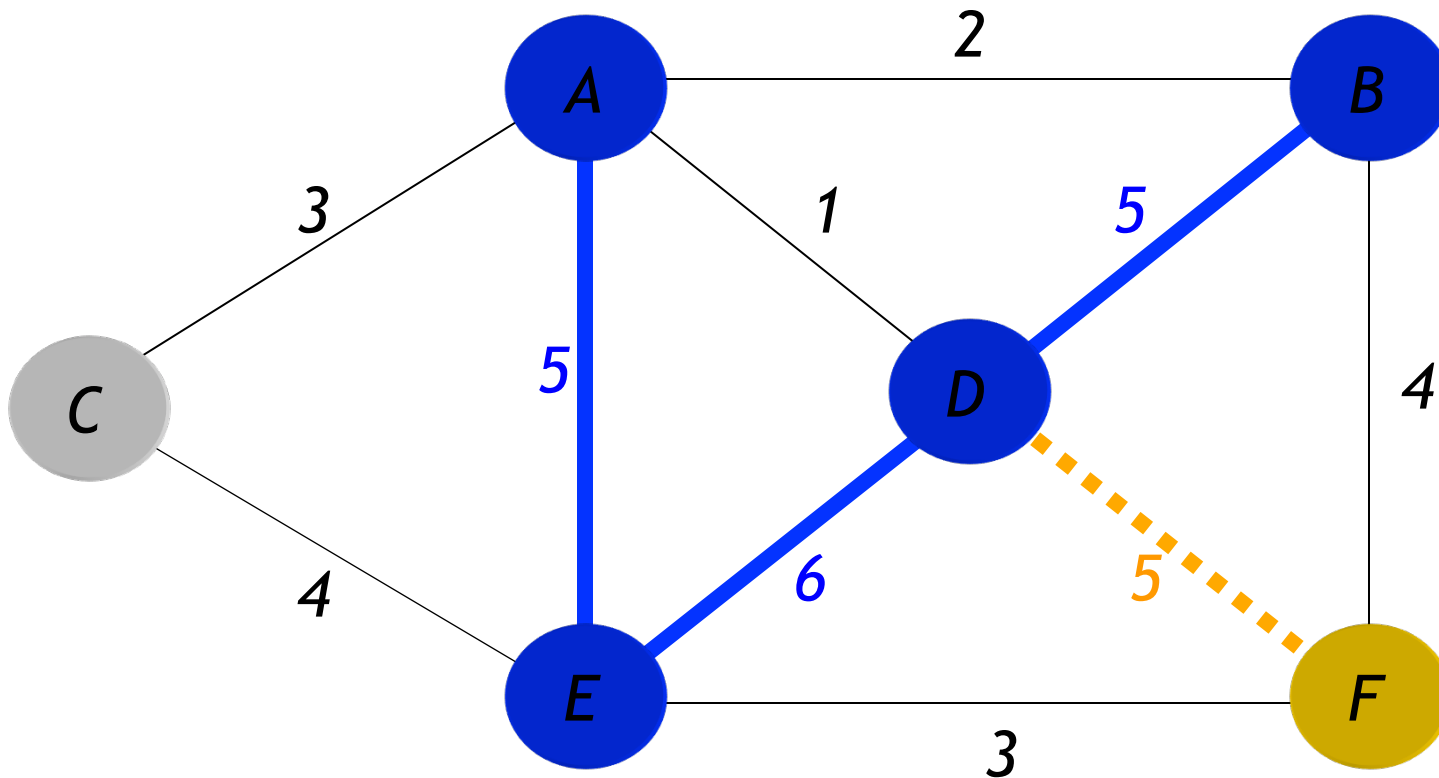
Kruskal Algorithm



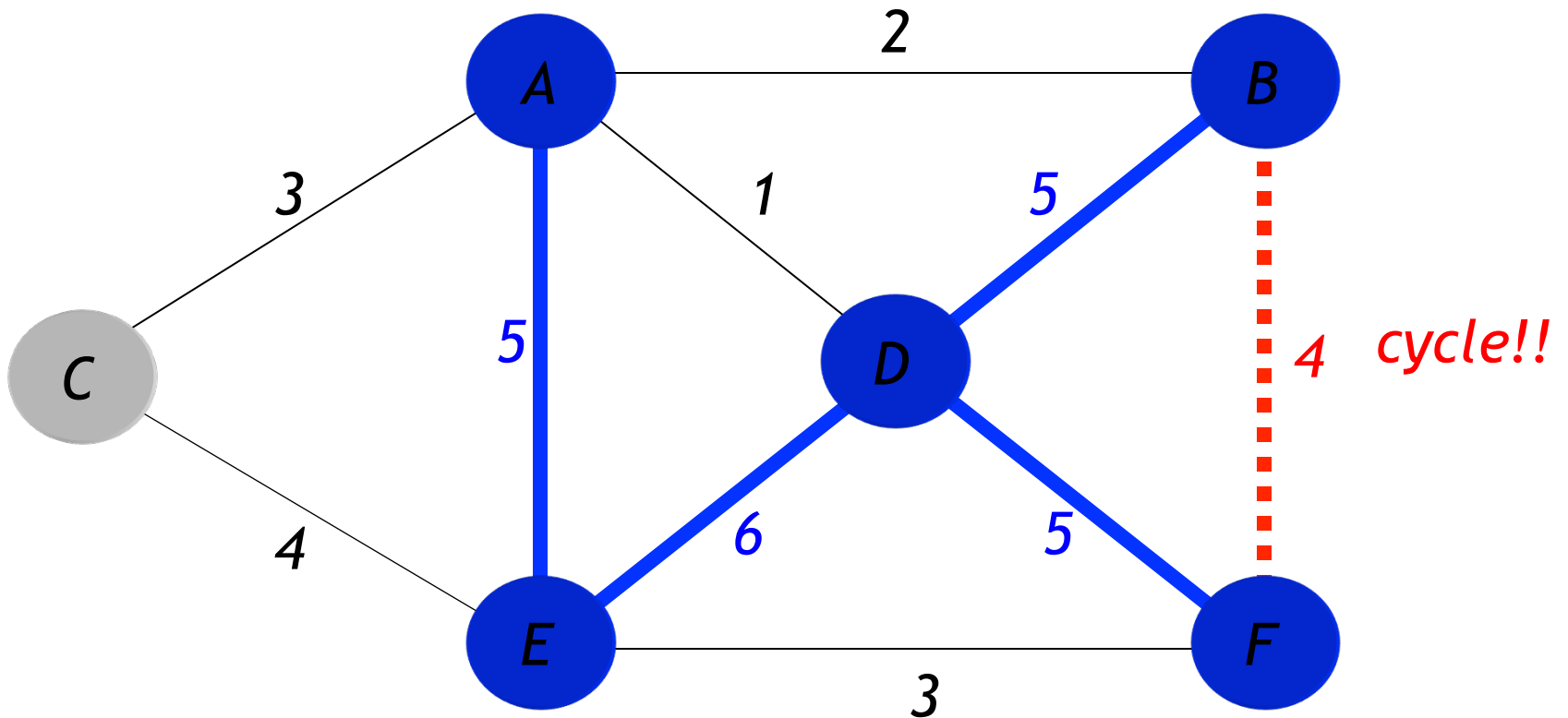
Kruskal Algorithm



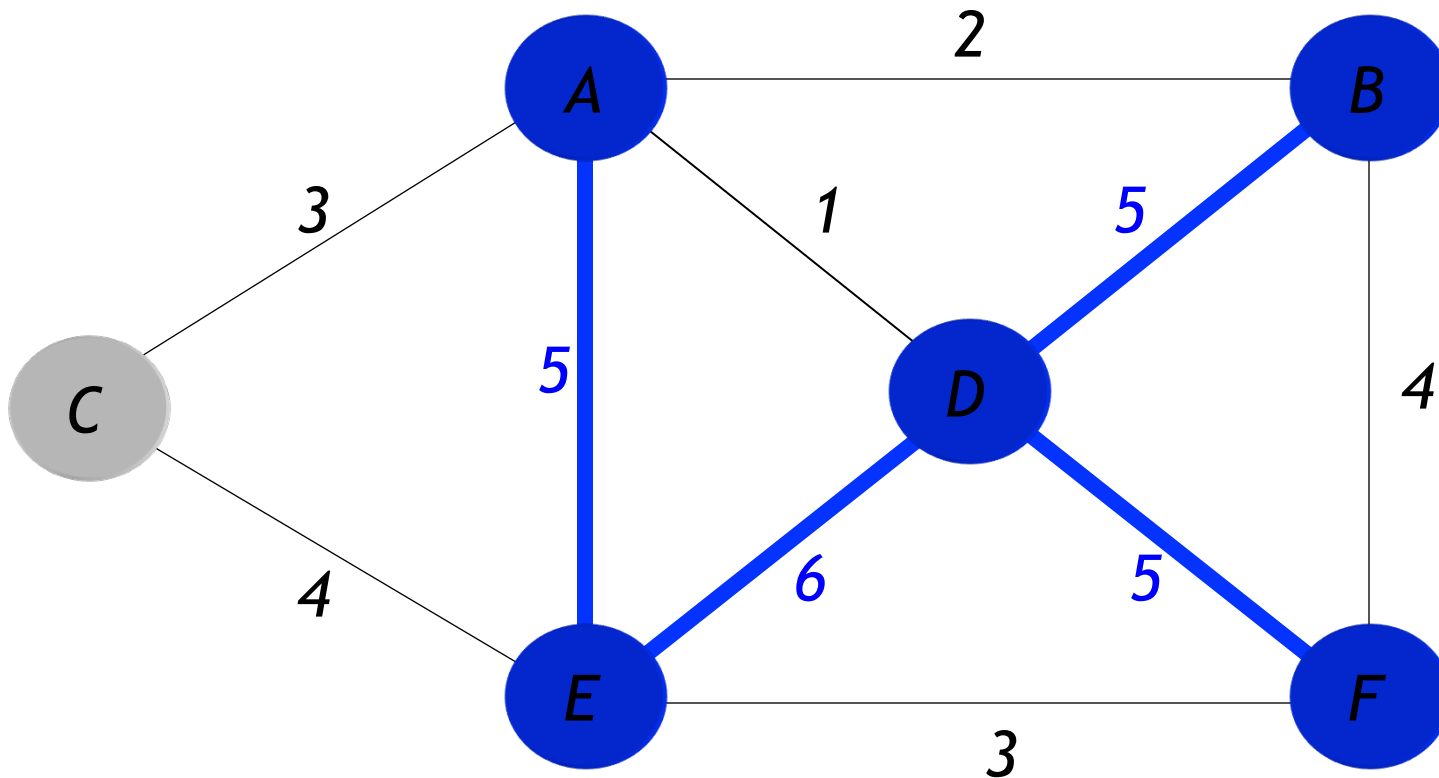
Kruskal Algorithm



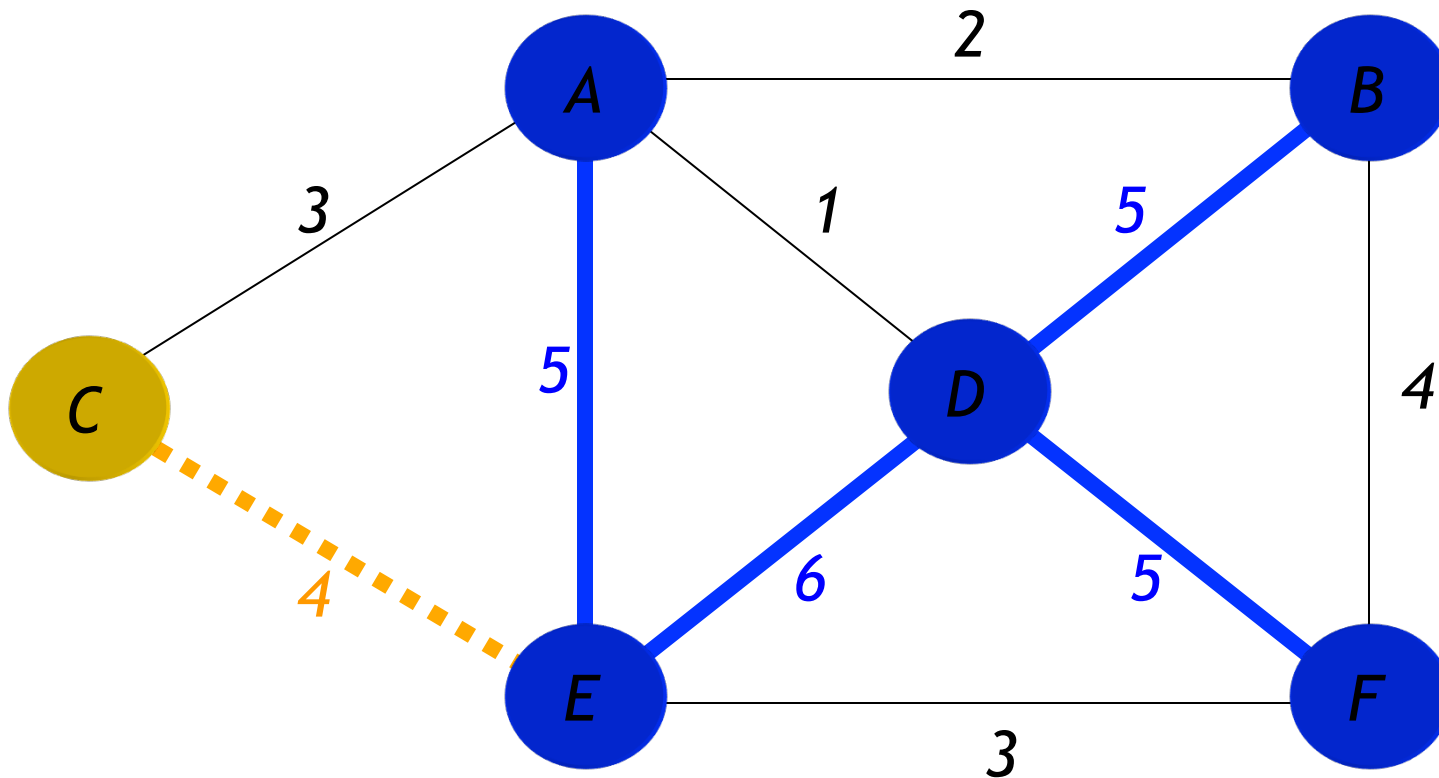
Kruskal Algorithm



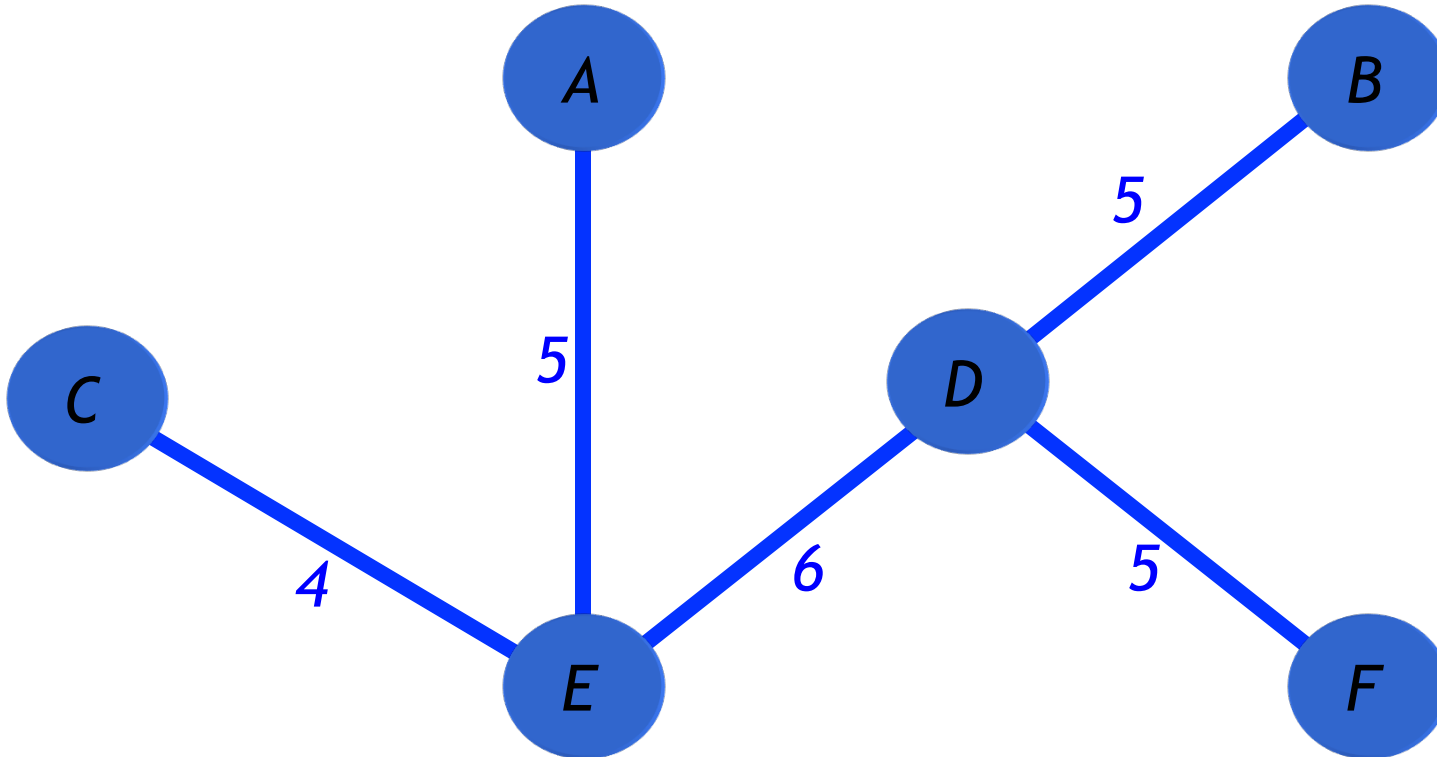
Kruskal Algorithm



Kruskal Algorithm



Kruskal Algorithm



Theorem 1 : If we Force the probabilities along the branches of the tree t to coincide with those computed from P , we get the best t -dependent approximation of P

$$\begin{aligned}
 D_{KL}(P, P_t) &= \sum_X P(X) \sum_{i=1}^n \log(P_t(x_i | parent(x_i))) + \sum_X P(X) \log P(X) \\
 &= \sum_X P(X) \sum_{i=1}^n \log(P_t(x_i | parent(x_i))) - H(X) \\
 &= \sum_{i=1}^n \sum_{x_i, x_j = parent(x_i)} P(x_j) P(x_i | x_j) \log(P_t(x_i | x_j)) - H(X)
 \end{aligned}$$

The proof follows from Gibbs' inequality

Gibbs' inequality

$$\log x \leq x - 1$$

$$D_{KL}(P, Q) = - \sum_{x \in X} p(x) \log \frac{q(x)}{p(x)} \geq - \sum_{x \in X} p(x) \left(\frac{q(x)}{p(x)} - 1 \right)$$

$$= - \sum_{x \in X} q(x) + \sum_{x \in X} p(x)$$

$$= - \sum_{x \in X} q(x) + 1 \geq 0$$

Gibbs' inequality

$$-\sum_{x \in X} p(x) \log \frac{q(x)}{p(x)} \geq 0$$

$$-\sum_{x \in X} p(x) \log q(x) \geq -\sum_{x \in X} p(x) \log p(x)$$

$$\sum_{x \in X} p(x) \log q(x) \leq \sum_{x \in X} p(x) \log p(x)$$

From theorem 1, we have:

$$D_{KL}(P, P_t) = - \sum_{i=1}^n \sum_{x_i, x_j = \text{parents}(x_i)} P(x_i, x_j) \log(P_t(x_i|x_j) - H(X))$$

$$P_t(x_i|x_j) = P(x_i|x_j)$$

maximizes D_{KL}

After assignment:

$$D_{KL}(P, P_t) = - \sum_{i=1}^n \sum_{x_i, x_j} P(x_i, x_j) \left[\log \left(\frac{P(x_i, x_j)}{P(x_i)P(x_j)} \right) + \log(P(x_i)) \right] - H(X)$$

$$W(X_i, X_j) = \sum_{(X_i, X_j)} P(X_i, X_j) \log_2 \frac{P(X_i, X_j)}{P(X_i)P(X_j)}$$

Evaluating KL divergence for a network

- Theorem (Chou and Liu, 1969). Suppose we define mutual information between any two nodes X_i and X_j as

$$W(X_i, X_j) = \sum_{(X_i, X_j)} P(X_i, X_j) \log_2 \frac{P(X_i, X_j)}{P(X_i)P(X_j)}$$

- Then the cross entropy $KL(P||Q)$ over all tree-structured distributions is minimized when the graph representing $Q(X_1 .. X_n)$ is a maximum weight spanning tree of the graph where the edge between nodes X_i and X_j is assigned the weight equal to $W(X_i, X_j)$.

Chow-Liu (CL) Results

- If distribution P is tree-structured, CL finds CORRECT one
- If distribution P is NOT tree-structured, CL finds tree structured Q that has minimal KL-divergence – $\operatorname{argmin}_Q \text{KL}(P; Q)$
- Even though $2^{\theta(n \log n)}$ trees, CL finds BEST one in poly time $O(n^2 [m + \log n])$

Evaluating KL divergence for a network

- Theorem (Lam and Bacchus, 1994). Suppose we define the weight between a nodes X_i and a set of arbitrary parents $Parents(X_i)$

$$W(X_i, Parents(X_i)) = \sum_{(X_i, Parents(X_i))} P(X_i, Parents(X_i)) \log_2 \frac{P(X_i, Parents(X_i))}{P(X_i)P(Parents(X_i))}$$

- Then the cross entropy $KL(P||Q)$ for a Bayesian network representing $Q(X_1 .. X_n)$ is a monotonically decreasing function of

$$\sum_{i=1, Parents(X_i) \neq \emptyset}^n W(X_i, Parents(X_i))$$

- Hence, $KL(P||Q)$ is minimized if and only if this sum of weights is maximized

Learning BN Structure

- If we find a Bayes network that maximizes

$$\sum_{i=1, Parents(X_i) \neq \emptyset}^n W(X_i, Parents(X_i))$$

- Then the probability distribution Q modeled by network will be closest to the underlying distribution P from which the data have been sampled with respect to $KL(P || Q)$
- It is possible to decrease $KL(P || Q)$ by adding arcs to the network – not a good idea
- Hence the need for MDL!

Score-based Bayesian Network Learning

- We need to find a Bayes network that maximizes

$$\sum_{i=1, Parents(X_i) \neq \emptyset}^n W(X_i, Parents(X_i))$$

- While minimizing

$$\sum_{i=1}^N \left(k_i \log_2(n) + d(s_i - 1) \prod_{X_j \in Parents(X_i)} s_j \right)$$

Alternative Scoring Functions - Notation

Each X_i takes r_i distinct values

$$s_i = \prod_{X_j \in Parents(X_i)} r_j$$

θ_{ijk} = probability that X_i takes the j th value in its domain given the k th instantiation of its parent set $Parents(X_i)$

η_{ijk} are the pseudocounts (from the Dirichlet prior)

N_{ijk} are the observed counts for the corresponding instantiation

$$N_{ik} = \sum_j N_{ijk}; \quad \eta_{ik} = \sum_j \eta_{ijk};$$

Bayesian scoring function

- Let $B = (G, \theta)$ be a Bayesian network with graph structure G and probability distribution parameterized by θ over a set of n random variables X_1, \dots, X_n
- Prior probability distribution $p(B)$ over the networks = $p(G, \theta)$
- Posterior probability given data D is given by

$$\begin{aligned}
 p(G, \theta | D) &= \frac{p(G, \theta, D)}{p(D)} = \frac{p(G, \theta, D)}{\sum_{G, \theta} p(G, \theta, D)} = \frac{p(D, G, \theta)}{\sum_{G, \theta} p(D, G, \theta)} \\
 &= \frac{p(G) p(\theta | D, G) p(D | G, \theta)}{\sum_{G, \theta} p(G, \theta) p(D | G, \theta)} \\
 &\propto p(G) p(\theta | D, G) p(D | G, \theta)
 \end{aligned}$$

Bayesian scoring function

$$p(D|G,\theta) \propto \prod_{i=1}^n \prod_{j=1}^{r_i} \prod_{k=1}^{s_i} \theta_{ijk}^{N_{ijk}}$$

where n is the number of random variables

r_i is the number of parents of node i

s_i is the number of instantiations of the parents of node i

θ_{ijk} = the probability of the j th value of the i th RV
given the k th instantiation of its parents

N_{ijk} = the corresponding counts estimated from D

Bayesian scoring function

$$p(D|G, \theta) \propto \prod_{i=1}^n \prod_{j=1}^{r_i} \prod_{k=1}^{s_i} \theta_{ijk}^{N_{ijk}}$$

N_{ijk} = the corresponding counts estimated from D

$$p(\theta|G) \propto \prod_{i=1}^n \prod_{j=1}^{r_i} \prod_{k=1}^{s_i} \theta_{ijk}^{\eta_{ijk} - 1}$$

$$p(\theta|G, D) \propto \prod_{i=1}^n \prod_{j=1}^{r_i} \prod_{k=1}^{s_i} \theta_{ijk}^{N_{ijk} + \eta_{ijk} - 1}$$

η_{ijk} = the corresponding pseudocounts

Standard Bayesian Measure

- Standard Bayesian Measure for a BN with graph G and parameters Θ

$$Q_{Bayes}(G, D) = \log p(G) + \sum_{i=1}^n \sum_{j=1}^{r_i} \sum_{k=1}^{s_i} (N_{ijk} + \eta_{ijk} - 1) \log \frac{(N_{ijk} + \eta_{ijk} - 1)}{(N_{ik} + \eta_{ik} - (r_i - 1))} - \frac{1}{2} Dim(G) \log N$$

where $Dim(G)$ is the number of parameters in the BN and N is the sample size

$\frac{1}{2} \log N$ is the average number of bits needed to store a number between 1 and N

Geiger Heckerman Scoring Function

- Geiger-Heckerman Measure for a BN with graph G and parameters Θ

$$\begin{aligned}
 Q_{GH}(G, D) &= \log p(G) + \log \int p(D | G, \Theta) p(\Theta | G) d\Theta \\
 &= \log p(G) + \sum_{i=1}^n \left\{ \sum_{k=1}^{s_i} \left\{ \log \frac{\Gamma(\eta_{ik})}{\Gamma(\eta_{ik} + N_{ik})} + \sum_{j=1}^{r_i} \log \frac{\Gamma(\eta_{ijk} + N_{ijk})}{\Gamma(\eta_{ijk})} \right\} \right\}
 \end{aligned}$$

- Can choose $p(G)$ to penalize complex networks

Cooper-Herskovits Scoring Function

- Cooper-Herskovits Measure for a BN with graph G and parameters Θ

$$Q_{CH}(G, D) = \log p(G) + \sum_{i=1}^n \left\{ \sum_{k=1}^{s_i} \left\{ \log \frac{\Gamma(r_i)}{\Gamma(r_i + N_{ik})} + \sum_{j=1}^{r_i} \log \Gamma(1 + N_{ijk}) \right\} \right\}$$

- Can choose $p(G)$ to penalize complex networks

Standard Bayesian Measure – Asymptotic version

- Asymptotic version of the standard Bayesian Measure for a BN with graph G and parameters Θ

$$\begin{aligned}
 Q_{AsymBayes}(G, D) &= Q_{MDL}(G, D) \\
 &= \log p(G) + \sum_{i=1}^n \sum_{j=1}^{r_i} \sum_{k=1}^{s_i} N_{ijk} \log \left(\frac{N_{ijk}}{N_{ik}} \right) - \left(\frac{1}{2} \right) Dim(G) \log N
 \end{aligned}$$

Asymptotic Information Measures

$$Q_I(B, D) = \log p(G)$$

$$+ \sum_i \sum_j \sum_k N_{ijk} \log \left(\frac{N_{ijk}}{N_{ik}} \right)$$

$$- \dim(G) f(|D|)$$

where $f(|D|)$ is a non - negative penalty function

$f(|D|) = 0$ for maximum likelihood information criterion

$f(|D|) = 1$ for Akaike information criterion

$f(|D|) = \frac{1}{2}(\log N)$ for Schwartz information criterion

Note : MDL is a special case of this measure

Structure Search as Optimization

- Input:
 - Training data
 - Scoring function
 - Set of possible structures
- Output:
 - A network that maximizes the score
- Key Computational Property: Decomposability:
$$\text{score}(G) = \sum \text{score} (\text{“family” of } X \text{ in } G)$$

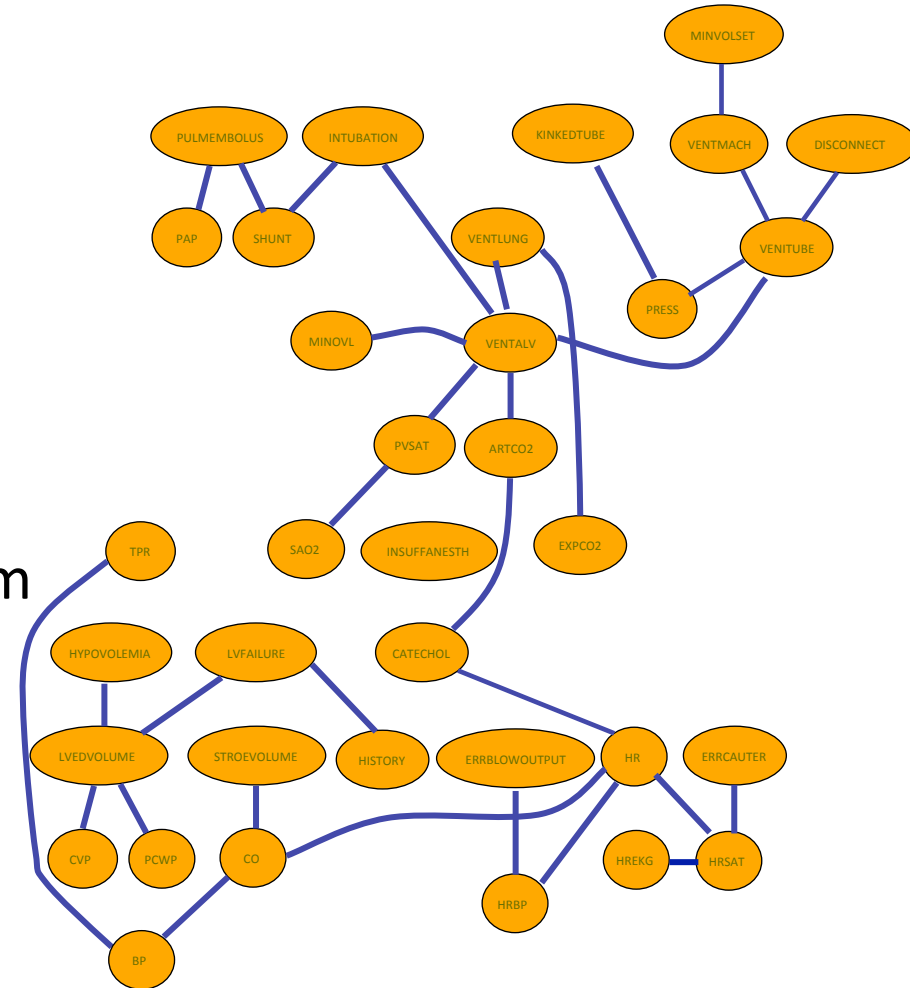
Tree-Structured Networks

Trees:

- At most one parent per variable

Why trees?

- Elegant mathematics
 - We can exactly and efficiently solve the optimization problem
- Sparse parameterization
 - Avoid overfitting



Learning Trees

- Let Pa_i denote parent of X_i
- We can write the Bayesian score as

$$Score(G : D) = \sum_i Score(X_i : Pa_i)$$

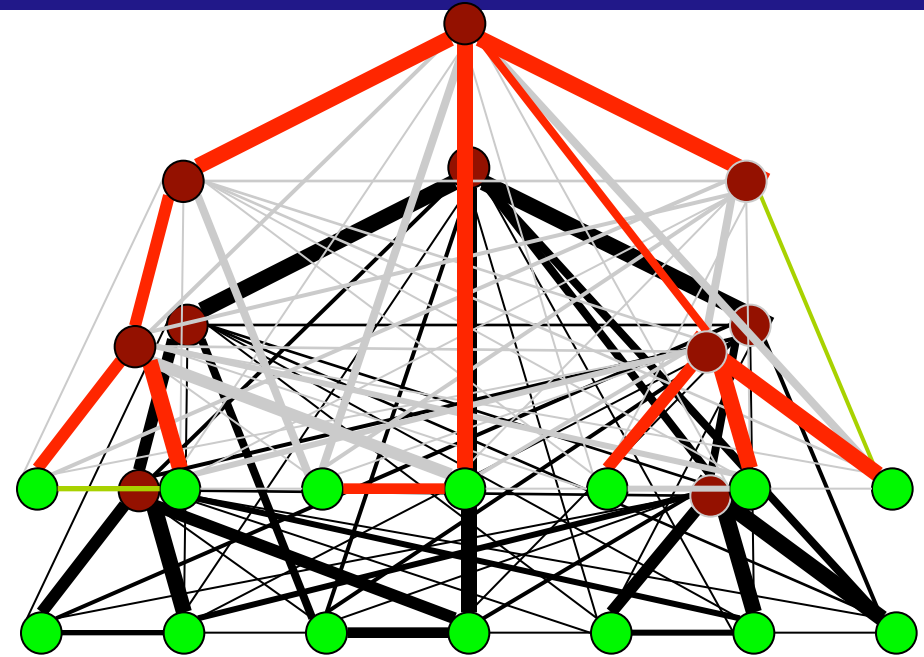
$$= \sum_i (Score(X_i : Pa_i) - Score(X_i)) + \sum_i Score(X_i)$$

Improvement over
 “empty” network

Score of “empty”
 network

- Score = sum of edge scores + constant

Learning Trees



- Set $w(j \rightarrow i) = \text{Score}(X_j \rightarrow X_i) - \text{Score}(X_i)$
- Find tree (or forest) with maximal weight --Standard max spanning tree algorithm — $O(n^2 \log n)$
- Theorem: This procedure finds tree with max score

Beyond Trees

- When we consider more complex network, the problem is not as easy
- Suppose we allow at most two parents per node
- A greedy algorithm is no longer guaranteed to find the optimal network
- **Theorem:** Finding maximal scoring structure with at most k parents per node is NP-hard for $k > 1$

Heuristic Search

- Define a search space:
 - search states are possible structures
 - operators make small changes to structure
- Traverse space looking for high-scoring structures
- Search techniques:
 - Greedy hill-climbing
 - Best first search
 - Simulated Annealing
 - ...

K2 Algorithm (Cooper and Herskovits)

- Start with an **ordered list** of random variables
- For each variable X_i add to its parent set, a node that is **lower numbered than X_i** and **yields the maximum improvement in score**
- Repeat until score does not improve or a complete network is obtained
- **Disadvantage:** Requires an **ordered list** of nodes

B Algorithm (Buntine)

- Start with the parent set for each random variables initialized to an empty set
- At each step, add a link (a node to the parent set of some node), that **does not introduce a cycle** and **yields the maximum improvement in score**
- Repeat until score does not improve or a complete network is obtained

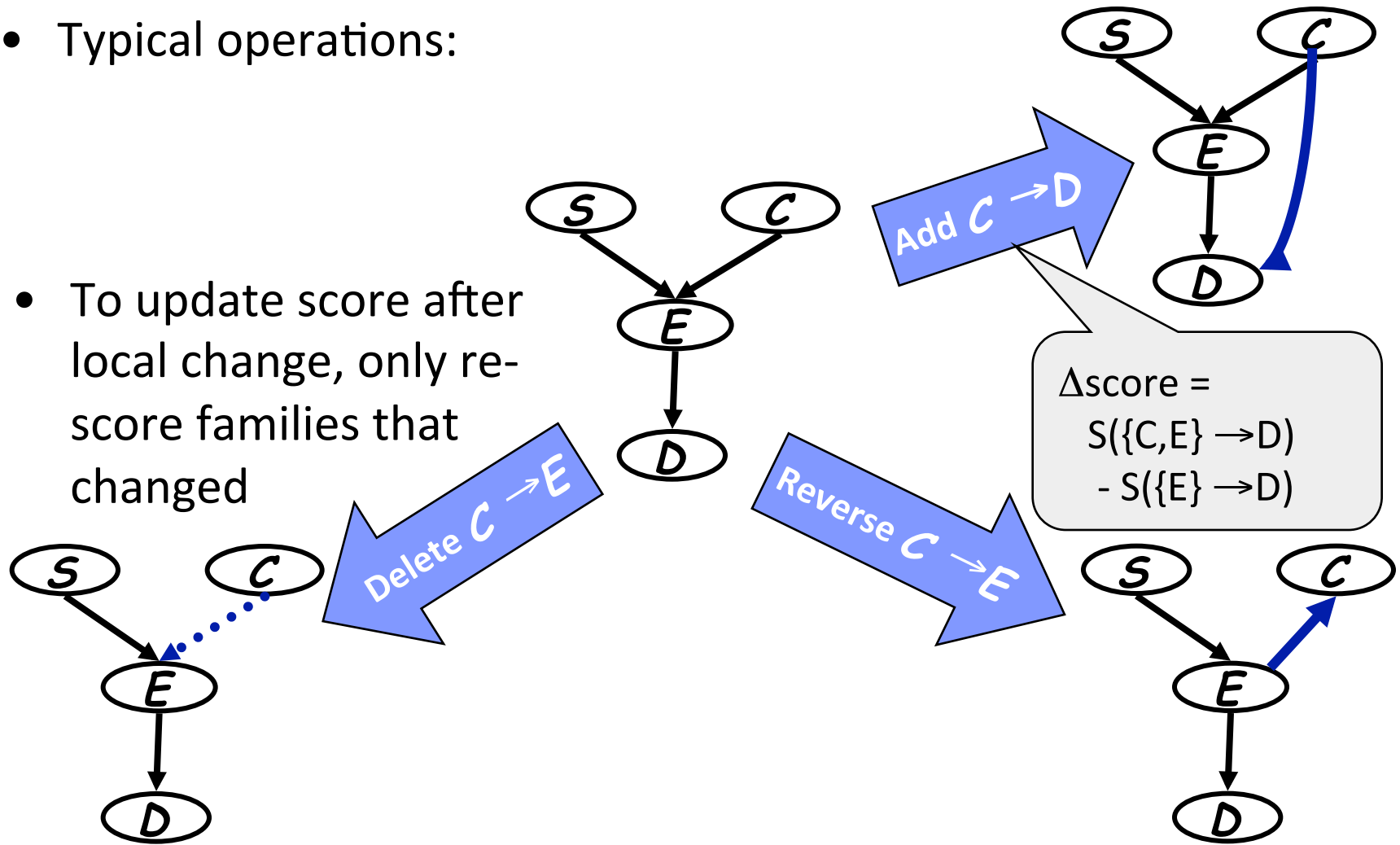
Local Search

- Start with a given network
 - empty network
 - best tree
 - a random network
- At each iteration
 - Evaluate all possible changes
 - Apply change based on score
- Stop when no modification improves score

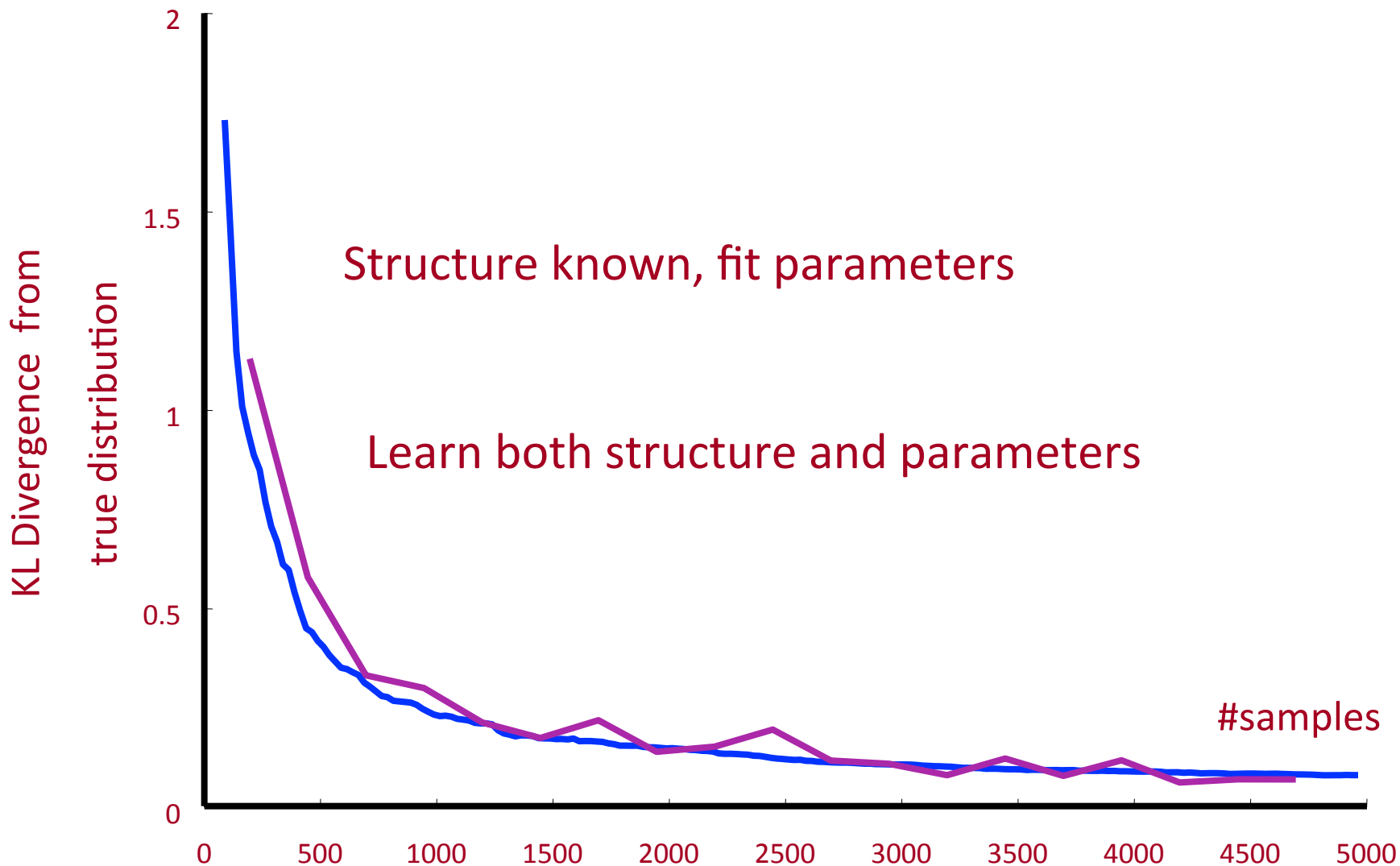
Heuristic Search

- Typical operations:

- To update score after local change, only re-score families that changed



Learning in Practice: Alarm network



Local Search: Possible Pitfalls

- Local search can get stuck in:
 - Local Maxima – All one-edge changes reduce the score
 - Plateau – Some one-edge changes leave the score unchanged
- Standard techniques can be used to cope with both
 - Random restarts
 - TABU search
 - Simulated annealing
 - ...

Independence Based Methods

- Rely on independence tests to decide whether to add links between nodes in the structure search phase
- Need to penalize for complex structures – Hard to beat a fully connected network!
- In the most general setting, there are too many independence tests to consider
- Sometimes it is possible to infer additional independences based on known (or inferred) independences (See Bromberg et al., 2009 and references cited therein)

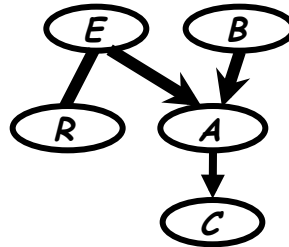
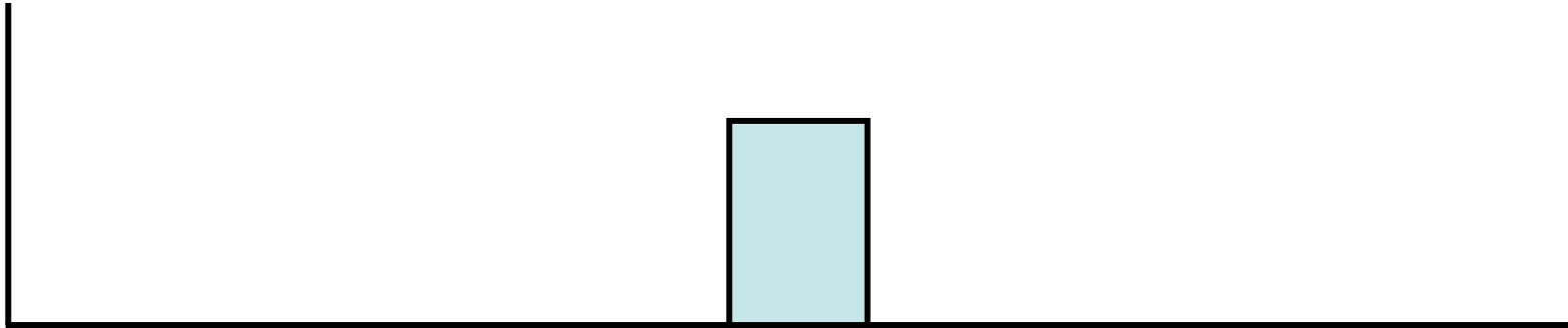
Structure Search: Summary

- Discrete optimization problem
- In some cases, optimization problem is easy
 - Example: learning trees
- In general, NP-Hard
 - Need to resort to heuristic search
 - Or restrict connectivity – each node assumed to have no more than l parents where l is much smaller than n
 - Stochastic search – e.g., simulated annealing, genetic algorithms

Structure Discovery

- Task: Discover structural properties
 - Is there a direct connection between X & Y
 - Does X separate between two “subsystems”
 - Does X causally effect Y
- Example: scientific data mining
 - Disease properties and symptoms
 - Interactions between the expression of genes

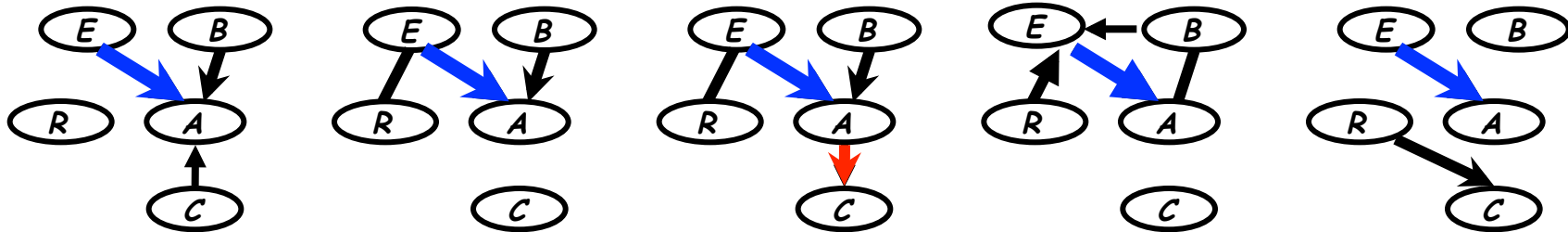
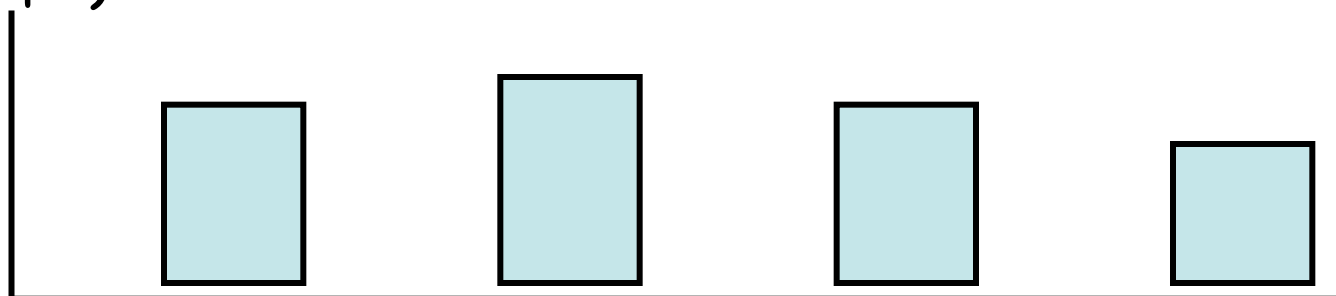
$P(G|D)$ Score based Structure Discovery



- Model selection
 - Pick a single high-scoring model
 - Use that model to infer domain structure

Bayesian Structure Discovery

$P(G|D)$



- Problem

- Small sample size \Rightarrow many high scoring models
- Individual models often unreliable
- Look for features shared across many models

Bayesian Approach

- Posterior distribution over structures
- Estimate probability of features
 - Edge $X \rightarrow Y$
 - Path $X \rightarrow \dots \rightarrow Y$
 - ...

Bayesian score
for G

$$P(f \mid D) = \sum_G f(G) P(G \mid D)$$

Feature of G ,
e.g., $X \rightarrow Y$

Indicator function
for feature f

MCMC over Networks

- Cannot enumerate structures, so sample structures
- MCMC Sampling
 - Define Markov chain over BNs
 - Run chain to get samples from posterior $P(G | D)$

- Possible pitfalls

$$P(f(G) | D) \approx \frac{1}{n} \sum_{i=1}^n f(G_i)$$

- Huge (super-exponential) number of networks
- Time for chain to converge to posterior is unknown
- Islands of high posterior, connected by low bridges

Fixed Ordering

- Suppose that
- We know the ordering of variables
 - say, $X_1 > X_2 > X_3 > X_4 > \dots > X_n$
 - parents for X_i must be in X_1, \dots, X_{i-1}

} $2^{k \cdot n \cdot \log n}$
 networks

- Limit number of parents per nodes to k

Intuition: Order decouples choice of parents

- Choice of $Pa(X_7)$ does not restrict choice of $Pa(X_{12})$

Upshot: Can compute efficiently in closed form

- Likelihood $P(D | \prec)$
- Feature probability $P(f | D, \prec)$

Sample Orderings

- We can write

$$P(f | D) = \sum_{\prec} P(f | \prec, D) P(\prec | D)$$

- Sample orderings and approximate

$$P(f | D) \approx \sum_{i=1}^n P(f | \prec_i, D)$$

- MCMC Sampling

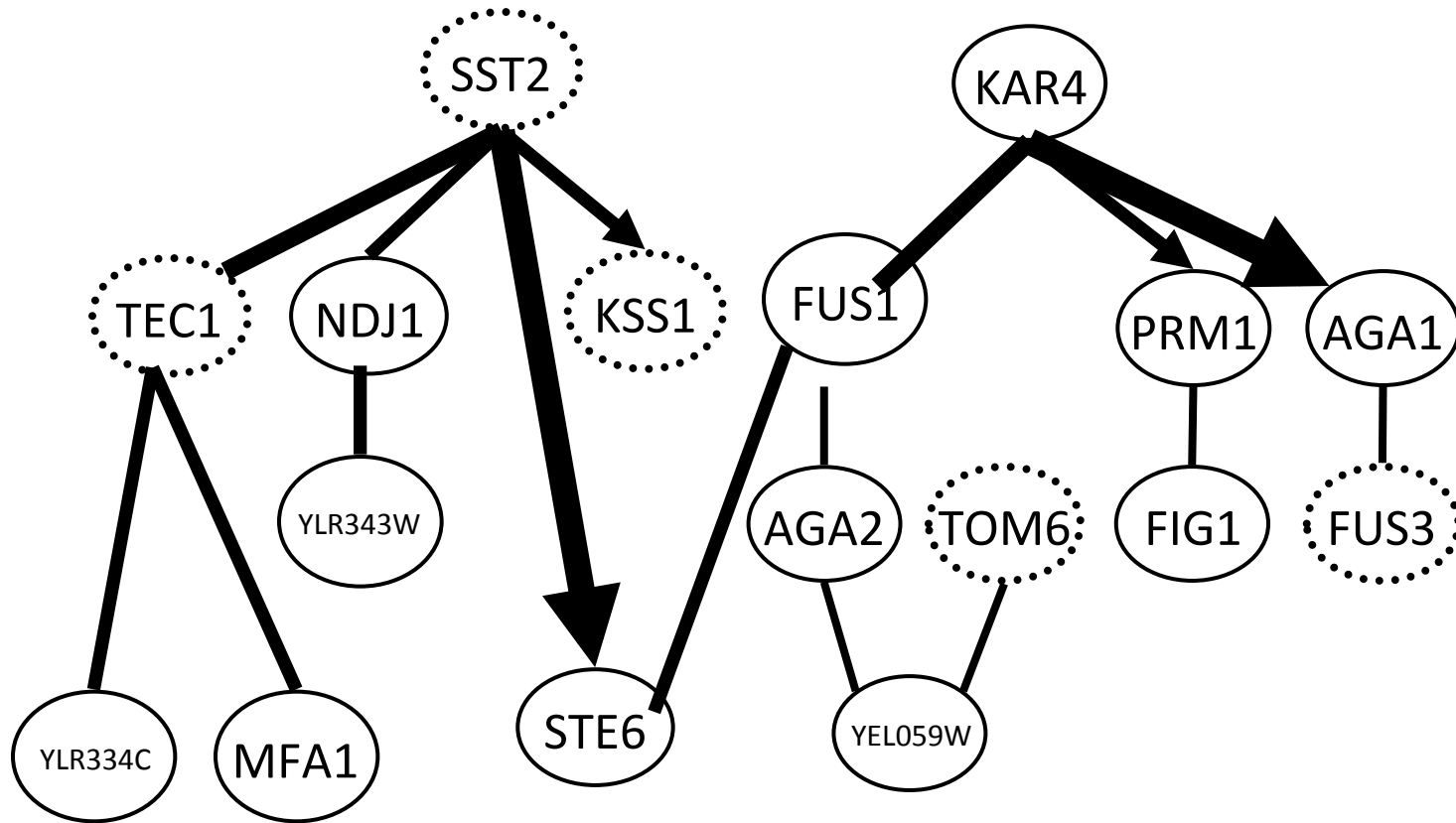
- Define Markov chain over orderings
- Run chain to get samples from posterior $P(\prec | D)$

Application: Gene expression Data Analysis

Friedman et al., 2001

- Input: Measurement of gene expression under different conditions
 - Thousands of genes
 - Hundreds of experiments
- Output: Model of gene interaction
 - Uncover pathways

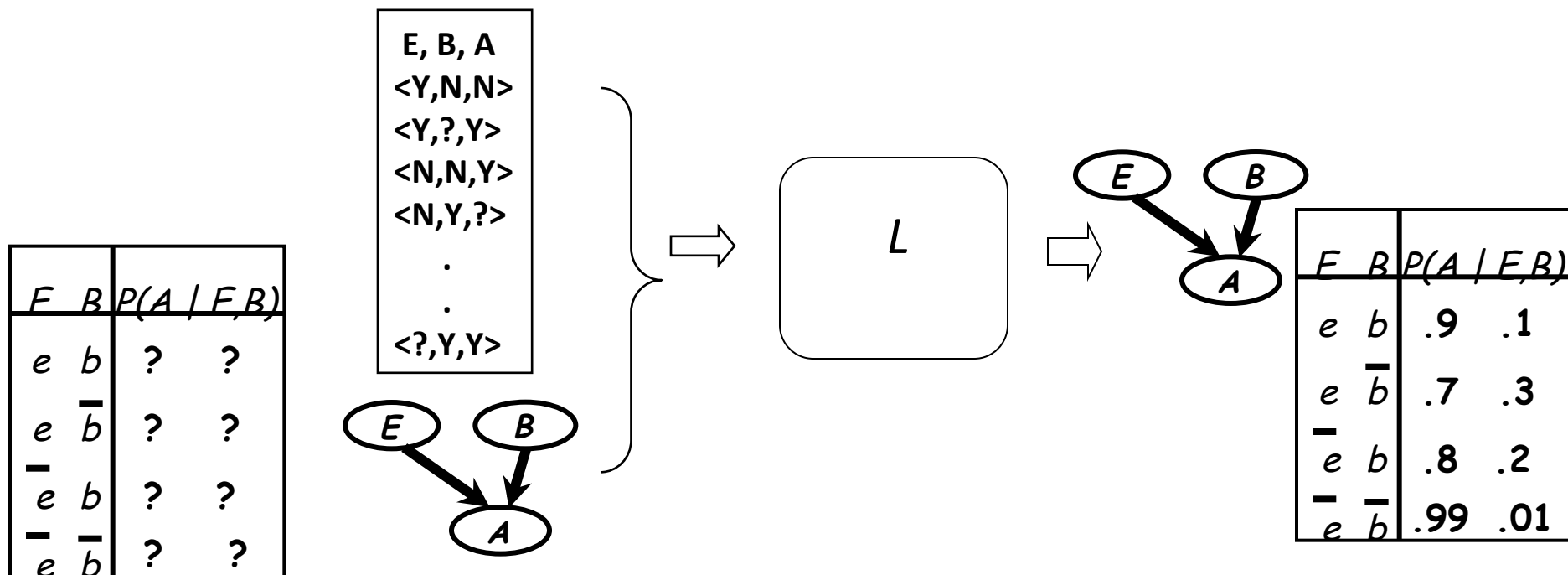
“Mating response” Substructure



- Automatically constructed sub-network of high-confidence edges
- Almost exact reconstruction of yeast mating pathway

Learning Problem

	Known Structure	Unknown Structure
Complete	Statistical parametric estimation (closed-form eq.)	Discrete optimization over structures (discrete search)
Incomplete	Parametric optimization (EM, gradient descent...)	Combined (Structural EM, mixture models...)



Incomplete Data

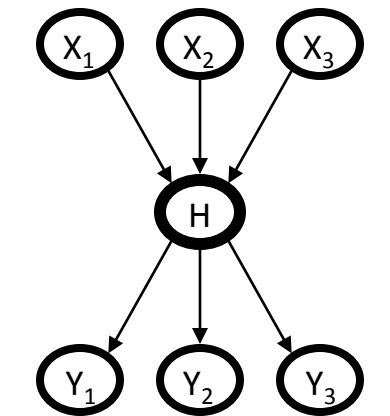
- Data are often **incomplete**
- Some variables of interest are not assigned values

This phenomenon occurs when we have

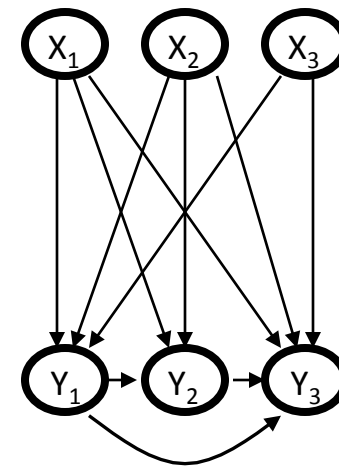
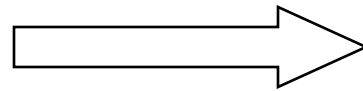
- **Missing values**
 - Some variables unobserved in some instances
- **Hidden variables**
 - Some variables are never observed
 - We might not even know they exist

Hidden (Latent) Variables

- Why should we care about hidden variables?



17 parameters



59 parameters

Incomplete Data

- In the presence of incomplete data, the likelihood can have multiple maxima



- Example:
- If H has two values, likelihood has two maxima
- In practice, many local maxima

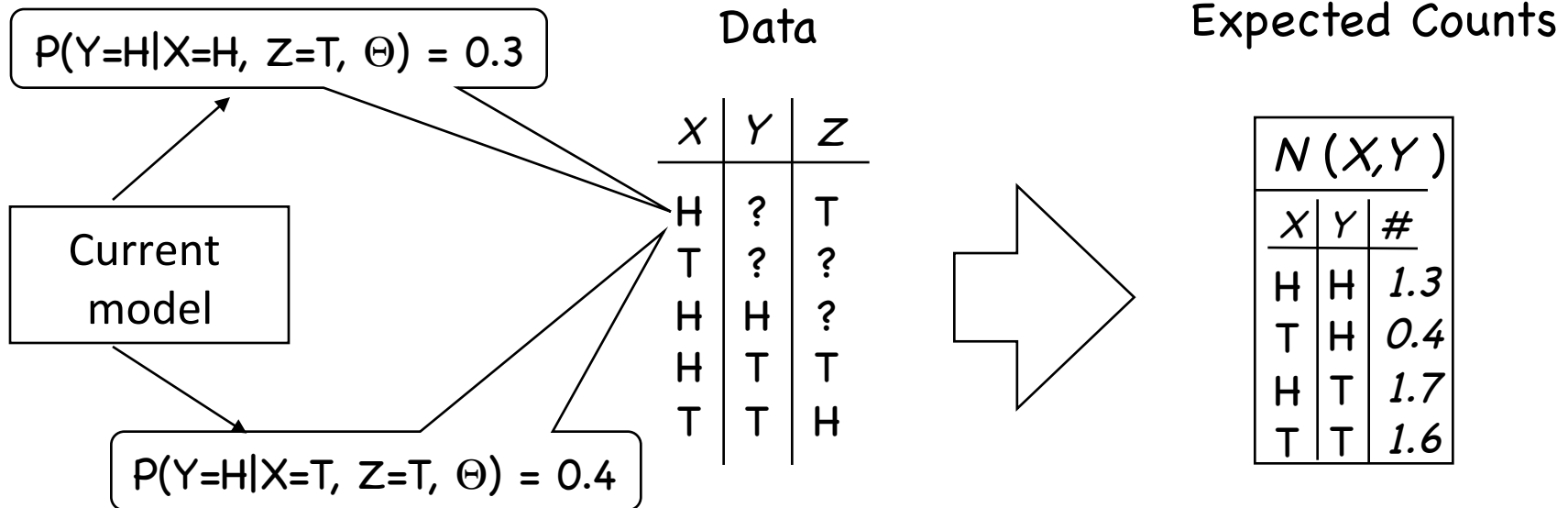
Expectation Maximization (EM)

- A general purpose method for learning from incomplete data

Intuition:

- If we had true counts, we could estimate parameters
- But with missing values, counts are unknown
- We “complete” counts using probabilistic inference based on current parameter assignment
- We use completed counts (as if they were actual counts) to re-estimate parameters

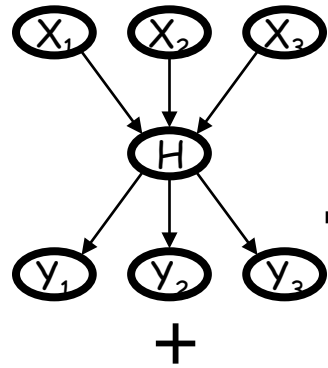
Expectation Maximization (EM)



Expectation Maximization (EM)

Iterate

Initial network (G, Θ_0)

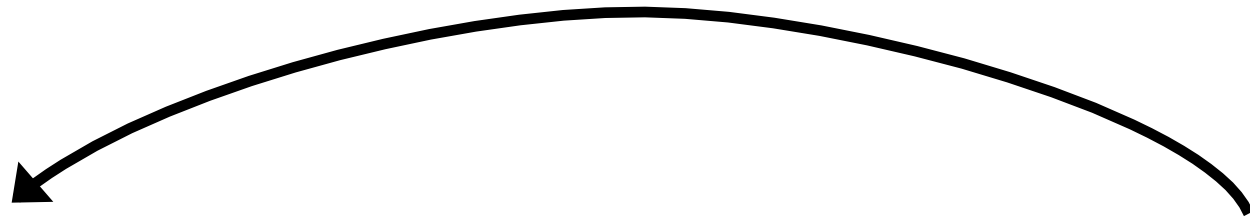
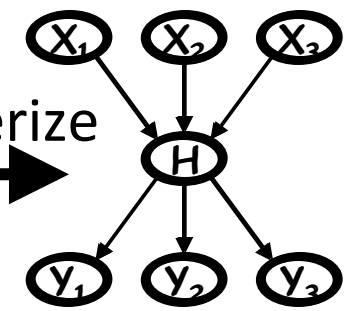


Computation
 (E-Step)

Expected Counts
$N(X_1)$
$N(X_2)$
$N(X_3)$
$N(H, X_1, X_2, X_3)$
$N(Y_1, H)$
$N(Y_2, H)$
$N(Y_3, H)$

Updated network (G, Θ_1)

Reparameterize
 (M-Step)



Expectation Maximization (EM)

- Formal Guarantees
- $L(\Theta_1:\mathcal{D}) \geq L(\Theta_0:\mathcal{D})$
 - Each iteration improves the likelihood
- If $\Theta_1 = \Theta_0$, then Θ_0 is a **stationary point** of $L(\Theta:\mathcal{D})$
 - Usually, this means a local maximum

Expectation Maximization (EM)

- Computational bottleneck:
- Computation of expected counts in E-Step
 - Need to compute posterior for each unobserved variable in each instance of training set
 - All posteriors for an instance can be derived from one pass of standard BN inference

Summary of Parameter Learning from Incomplete Data

- Incomplete data makes parameter estimation hard
- Likelihood function
 - Does not have closed form
 - Is multimodal
- Finding max likelihood parameters:
 - EM
 - Gradient ascent
- Both exploit inference procedures for Bayesian networks to compute expected sufficient statistics

Learning Problem Known Structure

Unknown Structure

Complete

Statistical parametric estimation
(closed-form eq.)

Discrete optimization over structures
(discrete search)

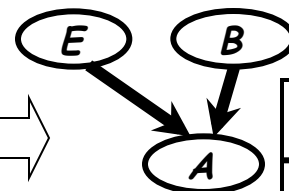
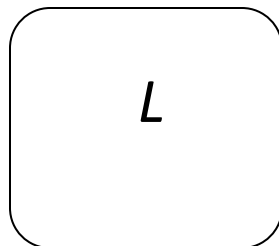
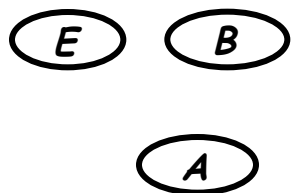
Incomplete

Parametric optimization
(EM, gradient descent...)

Combined
(Structural EM, mixture models...)

- E, B, A
- <Y,N,N>
- <Y,?,Y>
- <N,N,Y>
- <?,Y,Y>
- .
- .
- <N,Y, ?>

<i>F</i>	<i>B</i>	$P(A F, B)$	
<i>e</i>	<i>b</i>	?	?
<i>e</i>	\bar{b}	?	?
\bar{e}	<i>b</i>	?	?
\bar{e}	\bar{b}	?	?



<i>F</i>	<i>B</i>	$P(A F, B)$	
<i>e</i>	<i>b</i>	.9	.1
<i>e</i>	\bar{b}	.7	.3
\bar{e}	<i>b</i>	.8	.2
\bar{e}	\bar{b}	.99	.01

Incomplete Data: Structure Scores

- Recall, Bayesian score:

$$\begin{aligned} P(G | D) &\propto P(G)P(D | G) \\ &= P(G) \int P(D | G, \Theta) P(\Theta | G) d\theta \end{aligned}$$

- With incomplete data:
- Cannot evaluate **marginal likelihood** in closed form
- We have to resort to approximations:
 - Evaluate score around MAP parameters
 - Need to find MAP parameters (e.g., EM)

Structural EM

- Recall, in the case of complete data we had
Decomposition → efficient search

Idea:

- Instead of optimizing the real score...
 - Find decomposable alternative score
 - Such that maximizing the alternative score yields improvement in real score

Structural EM

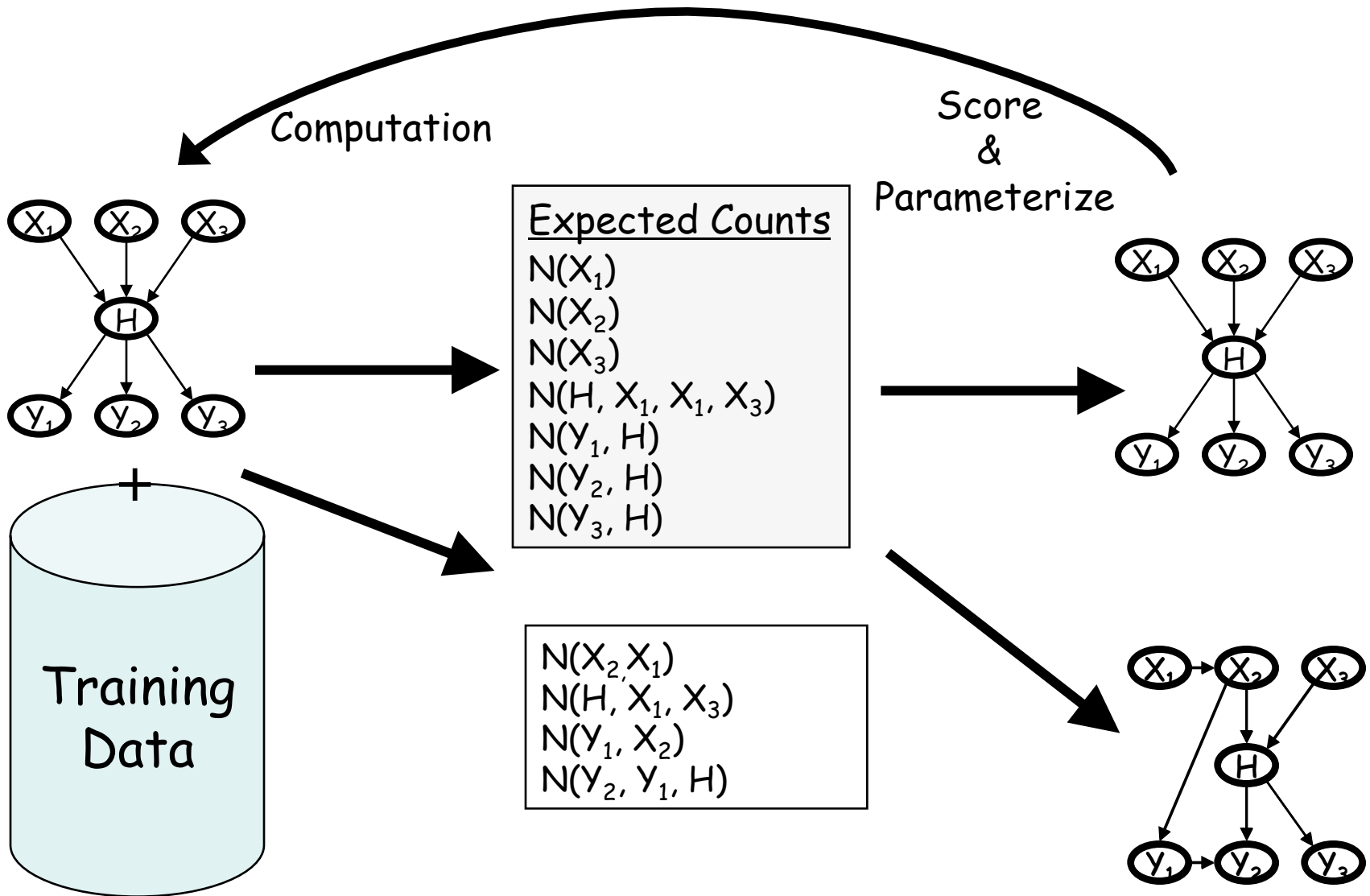
Idea:

- Use current model to help evaluate new structures

Outline:

- Perform search in (Structure, Parameters) space
- At each iteration, use current model for finding either:
 - Better scoring parameters: “parametric” EM step
 - or
 - Better scoring structure: “structural” EM step

Iterate



Probabilistic Relational Models

Probabilistic models of sequence data – A special case of Bayesian Networks

Outline

- Applications of sequence classification
- Bag of words, n-grams, and related models
- Markov models
- Hidden Markov models
- Higher order Markov models
- Variations on Hidden Markov Models
- Applications

Applications of Sequence Models

- Speech recognition
- Natural language processing
- Text processing
- Gesture recognition
- Biological sequence analysis
 - gene identification
 - protein classification

Bag of words, n -grams and related models

Map arbitrary length sequences to fixed length feature representations

Bag of words – represent sequences by feature vectors with as many components as there are words in the vocabulary

n -grams – short subsequences of n letters

Ignore relative ordering of words or n -grams along the sequence

“**cat chased the mouse**” and “**mouse chased the cat**” have identical bag of words representations

Bag of words, n -grams and related models

Fixed length feature representations make it possible to apply machine learning methods that work with feature-based representations

Features

- Given (as in the case of words English vocabulary)
- Discovered from data –statistics of occurrence of n -grams in data
 - If variable length n -grams are allowed, need to take into account possible overlaps
 - Computation of n -gram frequencies can be made efficient using dynamic programming – if a string appears k times in a piece of text, any substring of the string appears at least k times in the text

Markov models (Markov Chains)

A **Markov model** is a probabilistic model of symbol sequences in which the probability of the current event depends only on the immediately preceding event.

Consider a *sequence* of random variables X_1, X_2, \dots, X_N . Think of the subscripts as indicating word position in a sentence or a letter position in a sequence

Recall that a random variable is a *function*

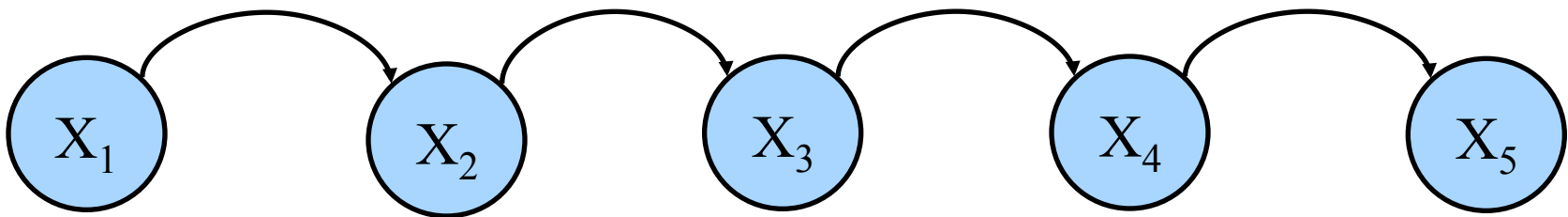
- In the case of sentences made of words, the range of the random variables is the vocabulary of the language.
- In the case of DNA sequences, the random variables take on values from a 4-letter alphabet $\{A, C, G, T\}$

Simple Model - Markov Chains

Markov Property: The state of the system at time $t+1$ only depends on the state of the system at time t

$$P[X_{t+1} = x_{t+1} / X_t = x_t, X_{t-1} = x_{t-1}, \dots, X_1 = x_1, X_0 = x_0]$$

$$= P[X_{t+1} = x_{t+1} / X_t = x_t]$$



Markov chains

The fact that subscript “1” appears on both the X and the x in “ $X_1 = x_1$ ” is a bit abusive of notation. It might be better to write:

$$P(X_1 = x_{s_1}, X_2 = x_{s_2}, \dots, X_t = x_{s_t})$$

where $\forall j \quad x_{s_j} \in \{v_1, \dots, v_L\} = \text{Range}(X_j)$

In what follows, we will abuse notation

Markov Chains

Stationarity -- Probabilities are independent of t when the process is stationary.

$$P[X_{t+1} = x_j \mid X_t = x_i] = a_{ij}$$

This means that if system is in state i , the probability that the system will transition to state j is p_{ij} regardless of the value of t

Describing a Markov Chain

A Markov chain can be described by the transition matrix A and initial state probabilities Q :

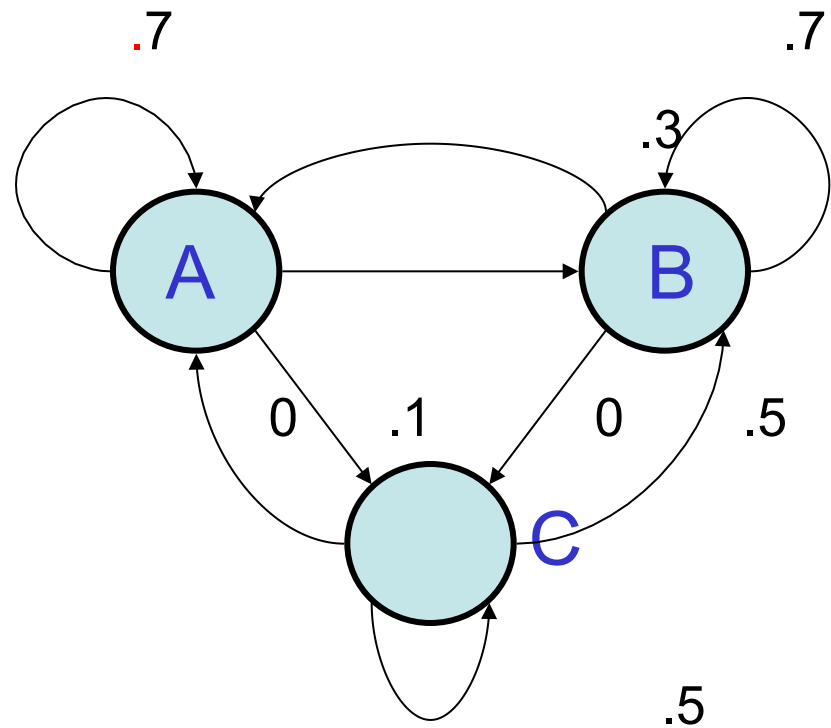
$$a_{ij} = P(X_{t+1} = j \mid X_t = i)$$

$$q_i = P(X_1 = i)$$

$$P(X_1, \dots, X_T) = P(X_1)P(X_2 \mid X_1) \dots P(X_T \mid X_{T-1}) = q_{X_1} \prod_{t=1}^{T-1} A(X_t, X_{t+1})$$

Markov chains

		Current symbol		
		A	B	C
Next Symbol	A	.7	.3	0
	B	.2	.7	.5
	C	.1	0	.5



Sample string: CCBBAAAAABAABACBABAAA

The probability of generating a string

Product of probabilities,
 one for each term in the
 sequence



$$p(\{X_t\}_1^T) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1})$$

↑
 This means a
 sequence of
 symbols from
 time 1 to time T

↑
 This comes from
 the table of initial
 probabilities

↑
 This is a
 transition
 probability

The fundamental questions

- **Likelihood** – Given a model $\mu = (A, Q)$, how can we efficiently compute the likelihood of an observation $P(X | \mu)$?





For any state sequence (X_1, \dots, X_T) :

$$P(X_1, \dots, X_T) = q_{x_1} a_{x_1 x_2} a_{x_2 x_3} \cdots a_{x_{T-1} x_T}$$

- **Learning** – Given a set of observation sequences X , and a generic model, how can we estimate the parameters that define the best model to describe the data?
- Use standard estimation methods – ML, MAP or Bayesian estimates discussed earlier in the course

Simple Example of a Markov model

Weather

raining today		rain tomorrow	$a_{rr} = 0.4$
raining today		no rain tomorrow	$a_{rn} = 0.6$
no raining today		rain tomorrow	$a_{nr} = 0.2$
no raining today		no rain tomorrow	$a_{nn} = 0.8$

Simple Example of a Markov model

$$A = \begin{pmatrix} 0.4 & 0.6 \\ 0.2 & 0.8 \end{pmatrix} \quad Q = \begin{pmatrix} 0.3 \\ 0.7 \end{pmatrix}$$

- Note that both the transition matrix and the initial state matrix are Stochastic Matrices (rows sum to 1)
- Note that in general, the transition probabilities between two states need not be symmetric ($a_{ij} \neq a_{ji}$) and the probability of transition from a state to itself (a_{ii}) need not be zero

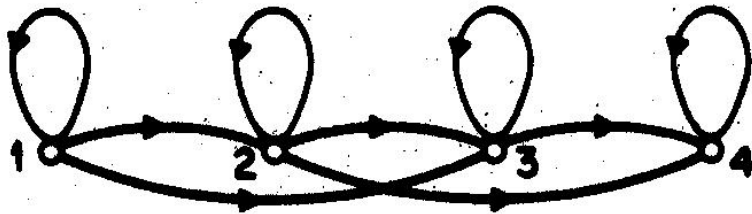
Types of Markov models – Ergodic models

Ergodic model - Strongly connected – directed path with **positive probabilities** from **each** state i to each state j (but not necessarily a complete directed graph). That is, for all i, j $a_{ij} > 0$; $a_{ii} > 0$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Types of Models – LR models

Left-to-Right (LR) model -- Index of state **non-decreasing** with time



$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}$$

$$a_{ij} = 0, \quad j > i + \Delta i$$

$$a_{ij} = 0, \quad j < i$$

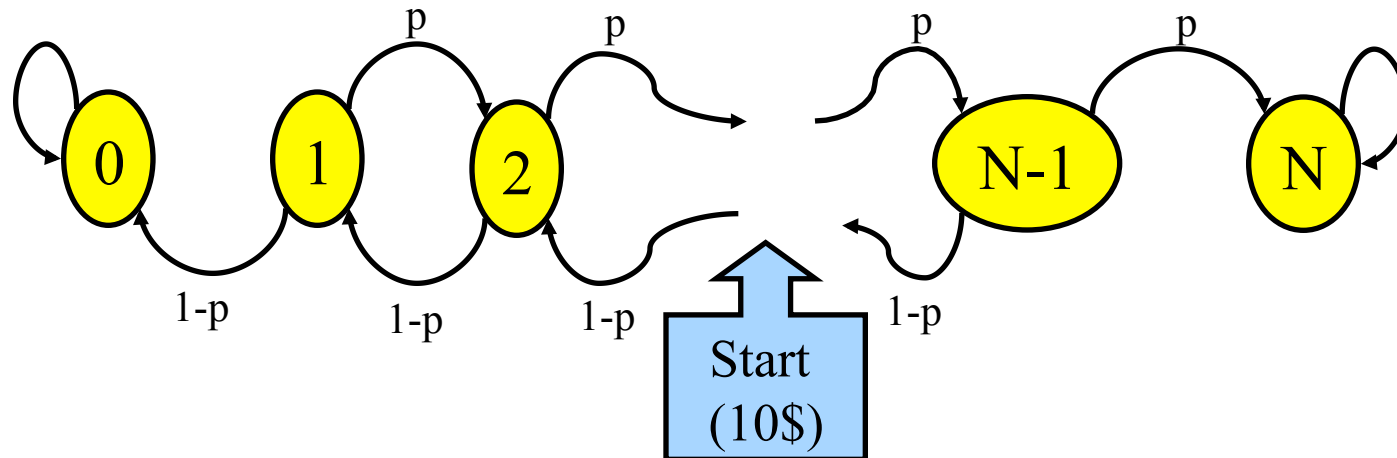
$$\pi_i = \begin{cases} 0, & i \neq 1 \\ 1, & i = 1 \end{cases}$$

Markov models with absorbing states

At each play

- Gambler wins \$1 with probability p or
- Gambler loses \$1 with probability $1-p$

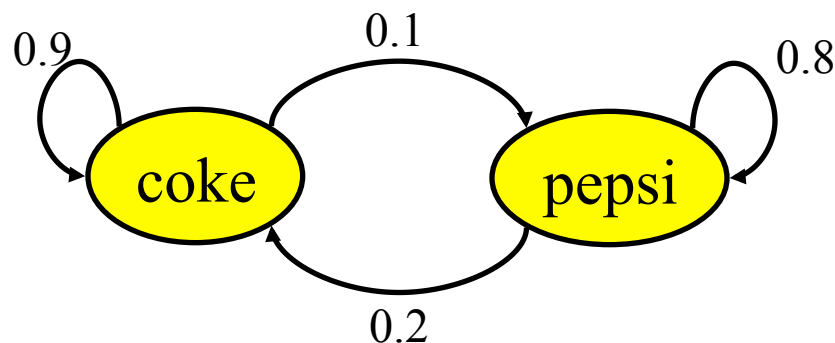
Game ends when gambler goes broke, or gains a fortune of \$100 -- Both \$0 and \$100 are absorbing states



Coke vs. Pepsi

Given that a person's last cola purchase was Coke, there is a 90% chance that her next cola purchase will also be Coke.

If a person's last cola purchase was Pepsi, there is an 80% chance that her next cola purchase will also be Pepsi.



Coke vs. Pepsi

Given that a person is currently a Pepsi purchaser, what is the probability that she will purchase Coke two purchases from now?

The transition matrix is:

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

(Corresponding to one purchase ahead)

$$A^2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} = \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix}$$

Coke vs. Pepsi

Given that a person is currently a Coke drinker, what is the probability that she will purchase Pepsi **three** purchases from now?

$$A^3 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix} = \begin{bmatrix} 0.781 & 0.219 \\ 0.438 & 0.562 \end{bmatrix}$$

Coke vs. Pepsi

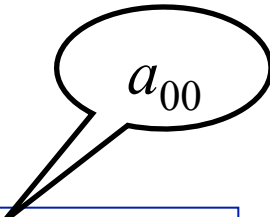
Assume each person makes one cola purchase per week. Suppose 60% of all people now drink Coke, and 40% drink Pepsi.

What fraction of people will be drinking Coke three weeks from now?

Let $(q_0, q_1) = (0.6, 0.4)$ be the initial probabilities.

We will denote Coke by 0 and Pepsi by 1

We want to find $P(X_3=0)$

$$A = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$


$$P(X_3 = 0) = \sum_{i=0}^1 q_i a_{i0}^{(3)} = q_0 a_{00}^{(3)} + q_1 a_{10}^{(3)} = (0.6)(0.781) + (0.4)(0.438) = 0.6438$$

Equilibrium distribution

$$p(x_t = i) = \sum_j p(x_t = i \mid x_{t-1} = j) p(x_{t-1} = j)$$

$$p_\infty(i) = \lim_{t \rightarrow \infty} p(x_t = i)$$

If $p_\infty(i) = \lim_{t \rightarrow \infty} p(x_t = i)$ is independent of the initial distribution, we call the resulting distribution the equilibrium distribution of the MC

Example: PageRank

- Define a matrix $A_{ij} = \begin{cases} 1 & \text{if website } j \text{ links to web site } i \\ 0 & \text{otherwise} \end{cases}$
- Construct a Markov Transition Matrix $M_{ij} = \frac{A_{ij}}{\sum_k A_{kj}}$
- The equilibrium distribution of the resulting Markov chain $p_\infty(i)$ corresponds to the probability of visiting website i
- A crude search engine works as follows:
 - For each website, collect a list of words that appear on it
 - For each word, make a list of websites that contain the word
 - Respond to a query with a list of websites that contain the query word, ordered by their $p_\infty(i)$ values

Learning the conditional probability table

Naïve: Just observe a lot of strings and set the conditional probabilities equal to observed probabilities

$$p(B | A) = \frac{\sum_{strings} \text{occurrences of } AB}{\sum_{strings} \text{occurrences of } A}$$

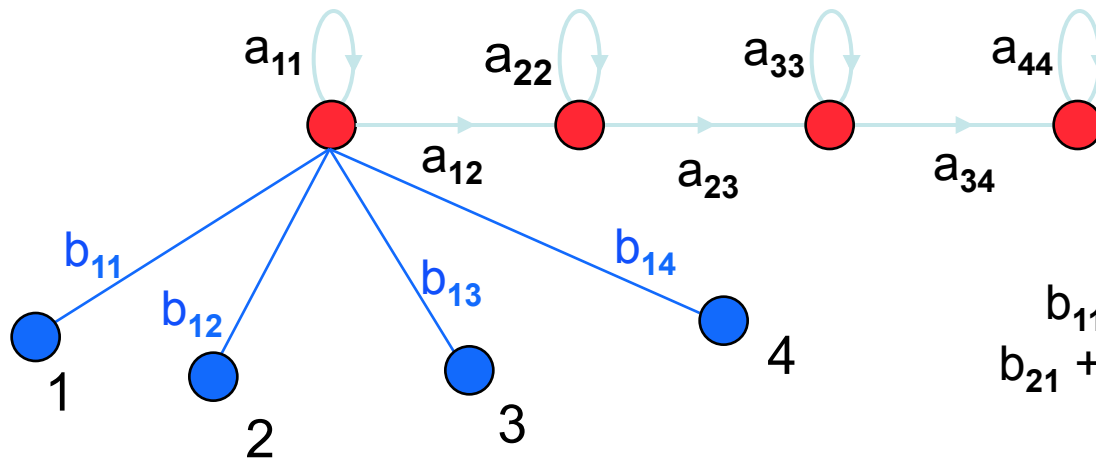
Better: add 1 to top and number of symbols to bottom - a weak uniform prior over the transition probabilities.

$$p(B | A) = \frac{1 + \sum_{strings} \# AB}{N_{symbols} + \sum_{strings} \# A}$$

Hidden Markov Models

Reading: Poritz, 1988

In many scenarios states cannot be **directly** observed.
 We need an extension -- **Hidden Markov Models**



$$b_{11} + b_{12} + b_{13} + b_{14} = 1,$$

$$b_{21} + b_{22} + b_{23} + b_{24} = 1, \text{ etc.}$$

Observations

a_{ij} are **state transition** probabilities.

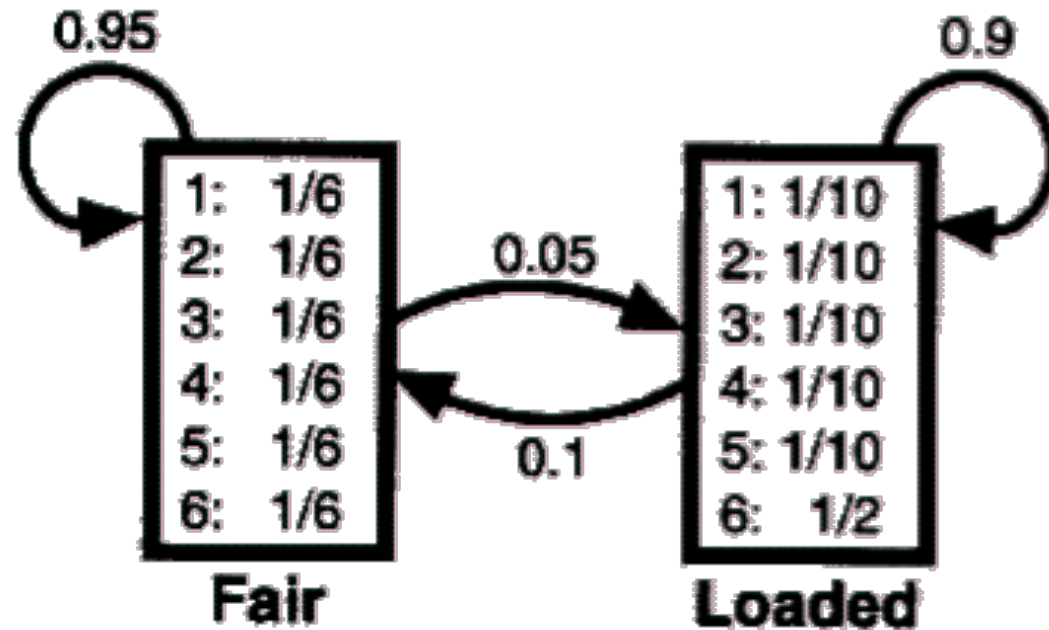
b_{ik} are **observation (output)** probabilities.

Hidden Markov Models

We introduce hidden states to get a hidden Markov model:

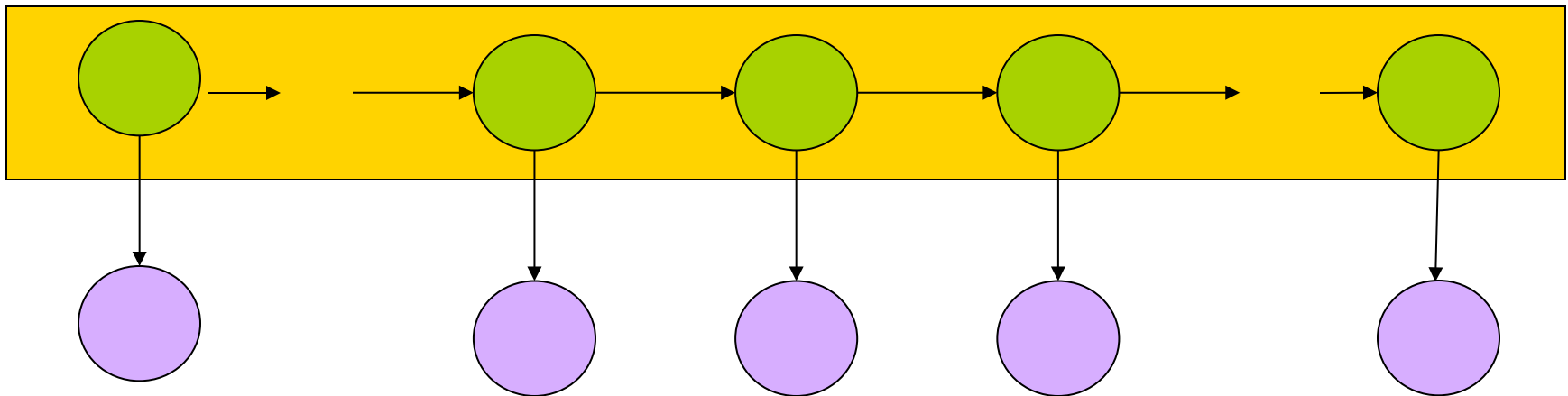
- The next hidden state depends only on the current hidden state
- The current symbol depends **only** on the current hidden state.

Example: Dishonest Casino



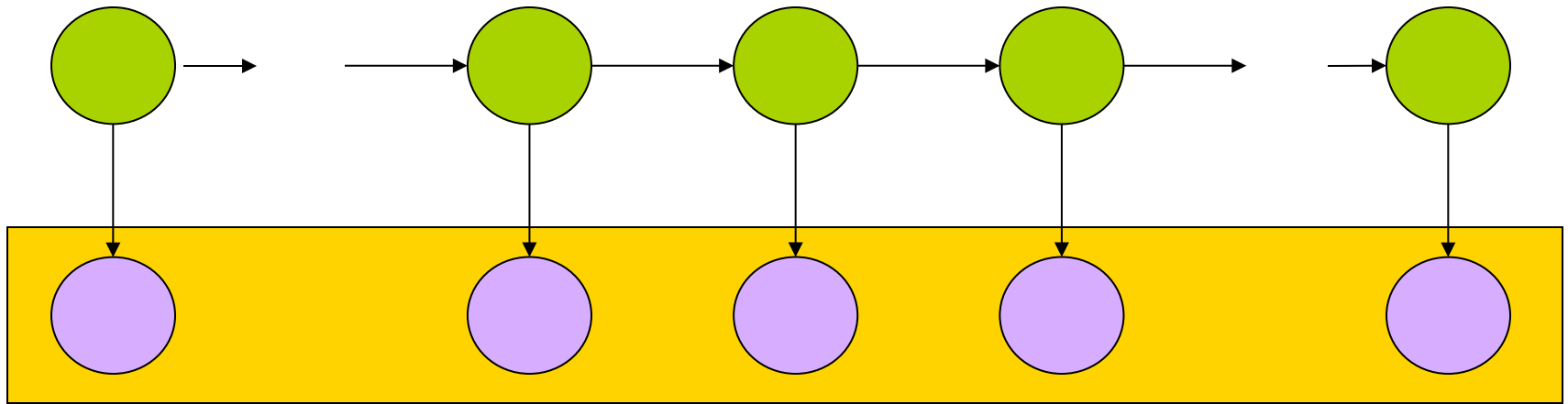
- What is **hidden** in this model? – State sequences
- You are allowed to see the outcome of a die roll
- You do not know which outcomes were obtained by a fair die and which outcomes were obtained by a loaded die

What is an HMM?



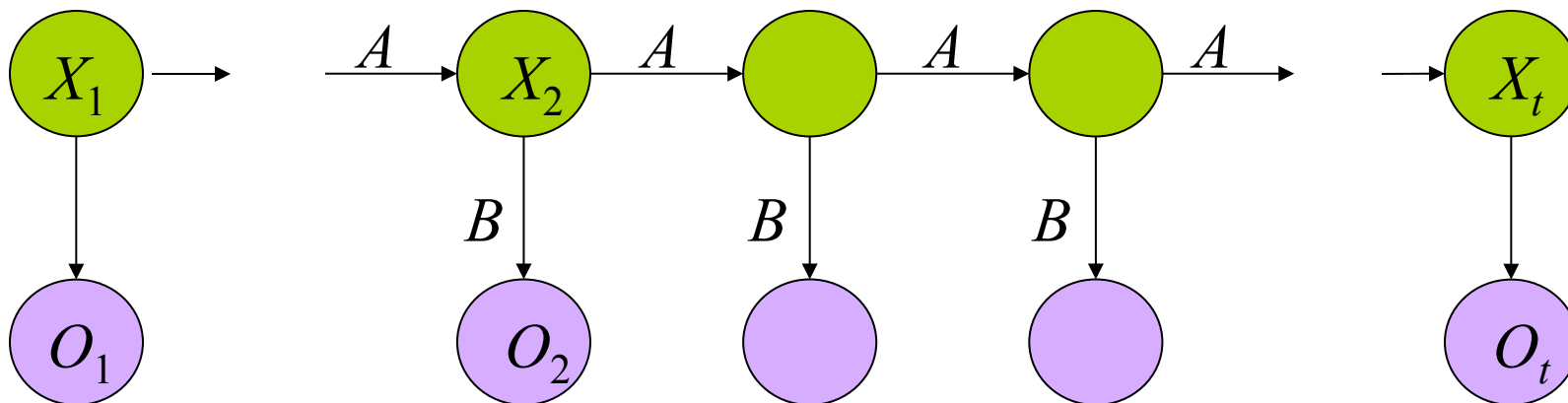
- Green circles are hidden states
- Each hidden state is dependent only on the previous state: Markov process
- “The past is independent of the future given the present.”

What is an HMM?



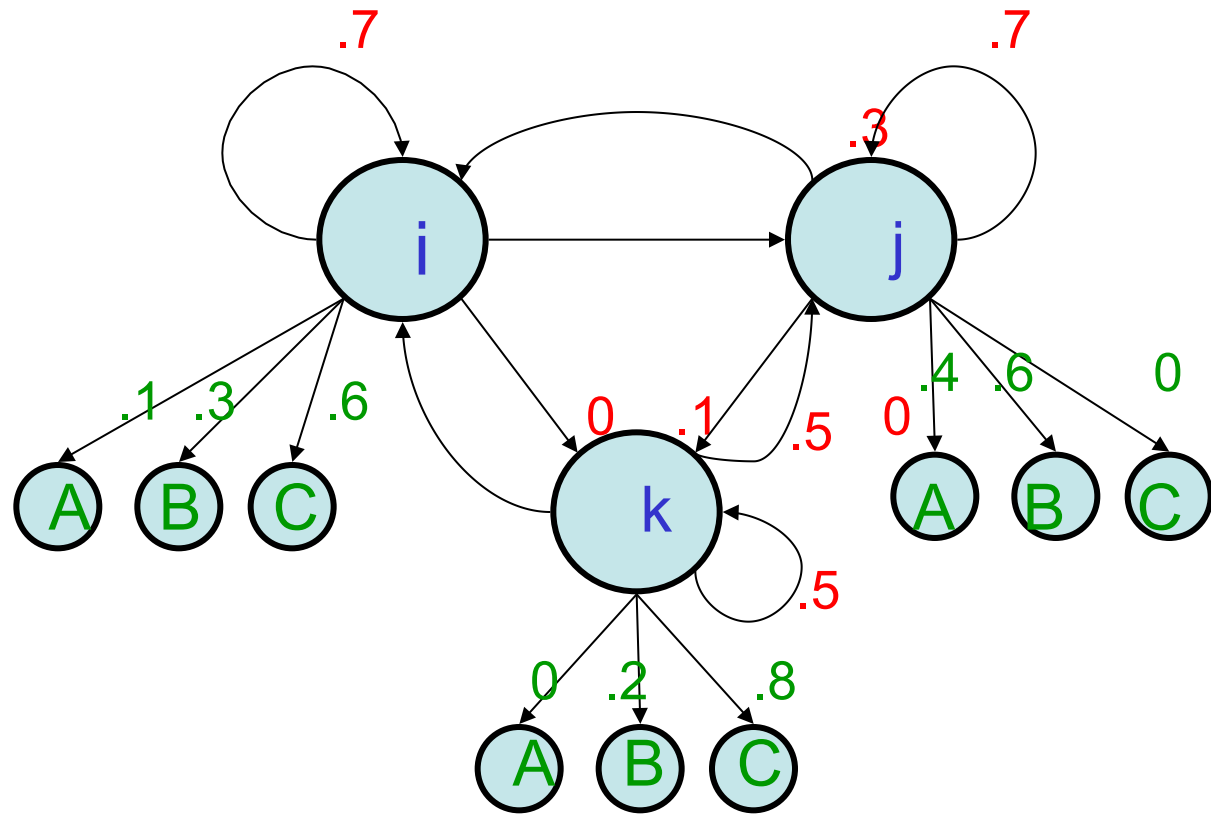
- Purple nodes are observed states
- Each observed state is dependent only on the corresponding hidden state

Specifying HMM



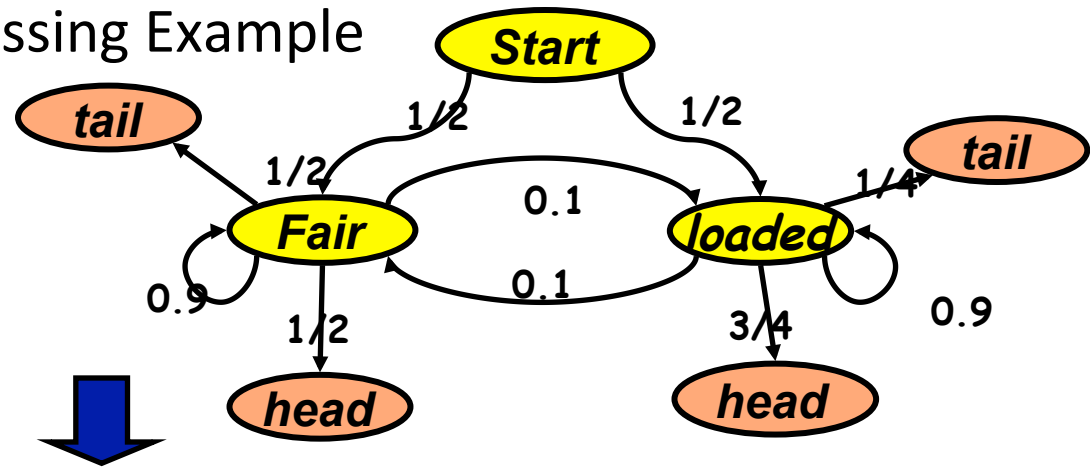
- $\{X, O, \Pi, A, B\}$
- $\Pi = \{\pi_i\}$ are the initial state probabilities
- $A = \{a_{ij}\}$ are the state transition probabilities
- $B = \{b_{ik}\}$ are the observation state probabilities

A hidden Markov model

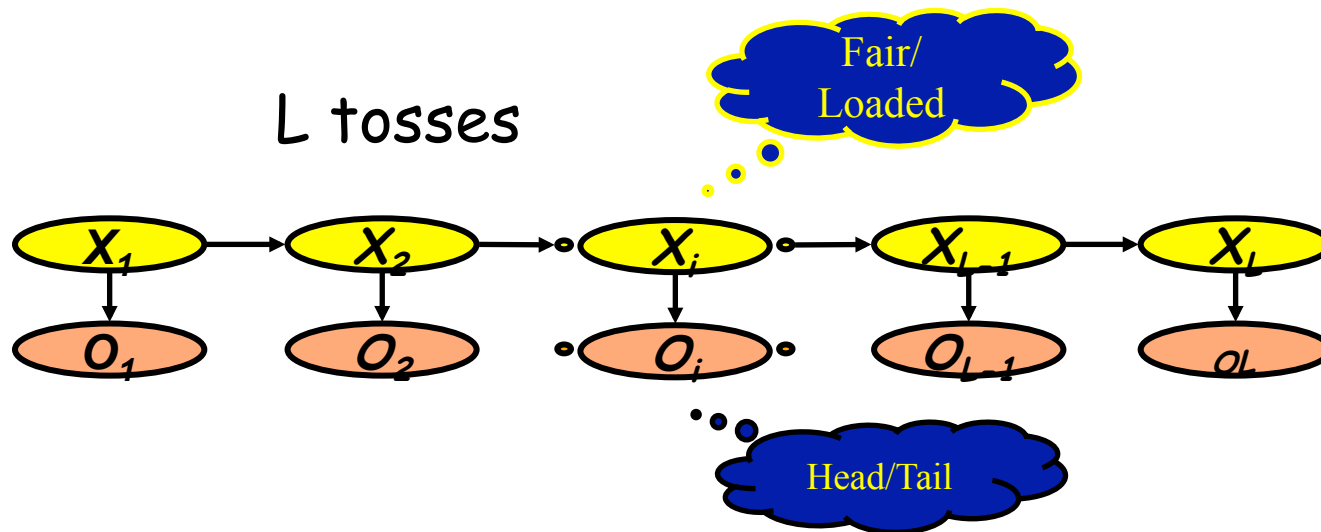


Each hidden node has a vector of **transition probabilities** and a vector of **output probabilities**.

Coin-Tossing Example

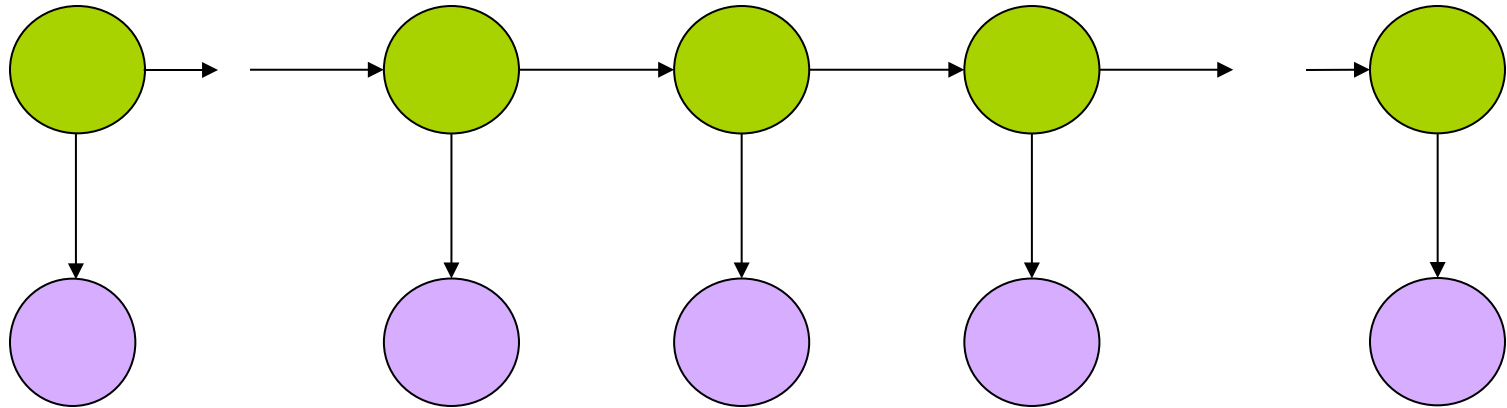


L tosses



Query: what are the most likely values in the X -nodes to generate the given data?

Fundamental problems



- **Filtering: Inferring the present** Compute the probability of a hidden state given the observations $p(x_t | o_{1:t})$
- **Prediction: Inferring the future** $p(x_t | o_{1:s}) \quad t > s$
- **Smoothing: Inferring the past** $p(x_t | o_{1:u}) \quad t < u$
- **Likelihood** – Compute the probability of an observation sequence given a model (HMM) $p(o_{1:T})$
- **Decoding** – Given an observation sequence, and a model, compute the most likely hidden state sequence

Generating a string from an HMM

It is easy to generate strings if we know the parameters of the model.

At each time step, make two random choices:

- Use the transition probabilities from the current hidden node to pick the next hidden node.
- Use the output probabilities from the current hidden node to pick the current symbol to output.

Generating a string from an HMM

It is easy to generate strings if we know the parameters of the model.

- First produce a hidden sequence
- From each hidden state in the sequence, produce an output symbol.
 - Hidden nodes only depend on previous hidden nodes
 - The probability of generating a hidden sequence does not depend on the output sequence that it generates.

The probability of generating a hidden sequence

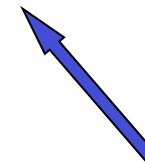
Product of probabilities, one for each term in the sequence



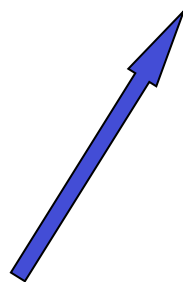
$$p(X_{1:T}) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1})$$



From the table of initial probabilities of hidden states



This is a transition probability between hidden states

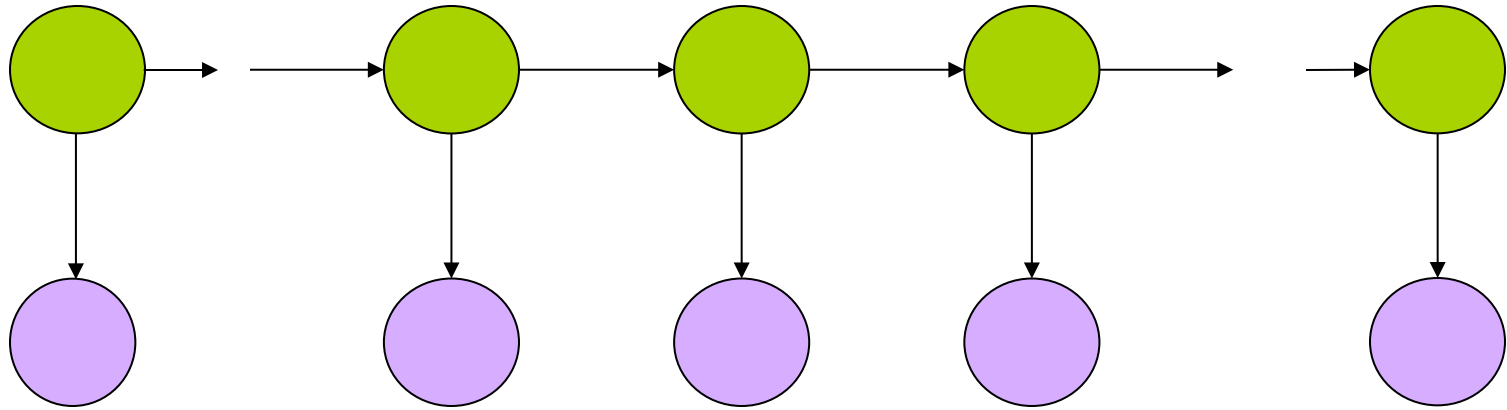


A sequence of hidden nodes from time 1 to time T



$$A_{ij} = p(X_t=j | X_{t-1}=i)$$

Fundamental problems



- **Filtering: Inferring the present** Compute the probability of a hidden state given the observations $p(x_t | o_{1:t})$
- **Prediction: Inferring the future** $p(x_t | o_{1:s}) \quad t > s$
- **Smoothing: Inferring the past** $p(x_t | o_{1:u}) \quad t < u$
- **Likelihood** – Compute the probability of an observation sequence given a model (HMM) $p(o_{1:T})$
- **Decoding** – Given an observation sequence, and a model, compute the most likely hidden state sequence

Filtering

$$\begin{aligned} p(x_t, o_{1:t}) &= \sum_{x_{t-1}} p(x_t, x_{t-1}, o_{1:t-1}, o_t) \\ &= \sum_{x_{t-1}} p(o_t | o_{1:t-1}, x_t, x_{t-1}) p(x_t | o_{1:t-1}, x_{t-1}) p(x_{t-1}, o_{1:t-1}) \\ &= \sum_{x_{t-1}} p(o_t | x_t) p(x_t | x_{t-1}) p(x_{t-1}, o_{1:t-1}) \end{aligned}$$

$$\alpha(x_t) = p(o_t | x_t) \sum_{x_{t-1}} p(x_t | x_{t-1}) \alpha(x_{t-1}) \quad t > 1$$

Smoothing

$$\begin{aligned}
 p(x_t, o_{1:T}) &= p(x_t, o_{1:t}, o_{t+1:T}) \\
 &= p(x_t, o_{1:t})p(o_{t+1:T} \mid x_t, o_{1:t}) \\
 &= \textit{past} \quad \textit{future} \\
 &= \alpha(x_t)\beta(x_t)
 \end{aligned}$$

$$\beta(x_t) = p(o_{t+1:T} \mid x_t)$$

$$\begin{aligned}
 p(o_{1:T} \mid x_{t-1}) &= \sum_{x_t} p(o_t, o_{t+1:T}, x_t \mid x_{t-1}) \\
 &= \sum_{x_t} p(o_t \mid x_t)p(o_{t+1:T} \mid x_t)p(x_t \mid x_{t-1})
 \end{aligned}$$

$$\beta(x_{t-1}) = \sum_{x_t} p(o_t \mid x_t)p(x_t \mid x_{t-1})\beta(x_t) \quad 2 \leq t \leq T$$

$$\beta(x_T) = 1$$

$$p(x_t \mid o_{1:T}) = \gamma(x_t) = \frac{\alpha(x_t)\beta(x_t)}{\sum_{x_t} \alpha(x_t)\beta(x_t)}$$

Computing the pair-wise marginal (needed for learning)

$$\begin{aligned}
 p(x_t, x_{t-1} \mid o_{1:T}) &\propto p(o_{1:T}, x_t, x_{t+1}) \\
 &= p(o_{1:t}, o_{t+1}, o_{t+2:T}, x_t, x_{t+1}) \\
 &= p(o_{t+2:T} \mid o_{1:t}, o_{t+1}, x_t, x_{t+1}) p(o_{1:t}, o_{t+1}, x_t, x_{t+1}) \\
 &= p(o_{t+2:T} \mid x_{t+1}) p(o_{t+1} \mid o_{1:t}, x_t, x_{t+1}) p(o_{1:t}, x_t, x_{t+1}) \\
 &= p(o_{t+2:T} \mid x_{t+1}) p(o_{t+1} \mid x_{t+1}) p(o_{1:t}, x_t, x_{t+1}) \\
 &= p(o_{t+2:T} \mid x_{t+1}) p(o_{t+1} \mid x_{t+1}) p(x_{t+1} \mid o_{1:t}, x_t) p(o_{1:t}, x_t) \\
 &= p(o_{t+2:T} \mid x_{t+1}) p(o_{t+1} \mid x_{t+1}) p(x_{t+1} \mid x_t) p(x_t, o_{1:t}) \\
 &= \beta(x_{t+1}) p(o_{t+1} \mid x_{t+1}) p(x_{t+1} \mid x_t) \alpha(x_t)
 \end{aligned}$$

Computing the likelihood

$$p(o_{1:T}) = \sum_{x_t} p(x_t, o_{1:T}) = \sum_{x_t} \alpha(x_t) \beta(x_t) \quad \textit{from smoothing}$$

Most likely path given the observation sequence (Viterbi algorithm)

$$\max_{x_T} \prod_{t=1}^T p(o_t | x_t) p(x_t | x_{t-1}) = \left\{ \prod_{t=1}^T p(o_t | x_t) p(x_t | x_{t-1}) \right\} \underbrace{\max_{x_T} p(o_T | x_T) p(x_T | x_{T-1})}_{\mu(x_{T-1})}$$

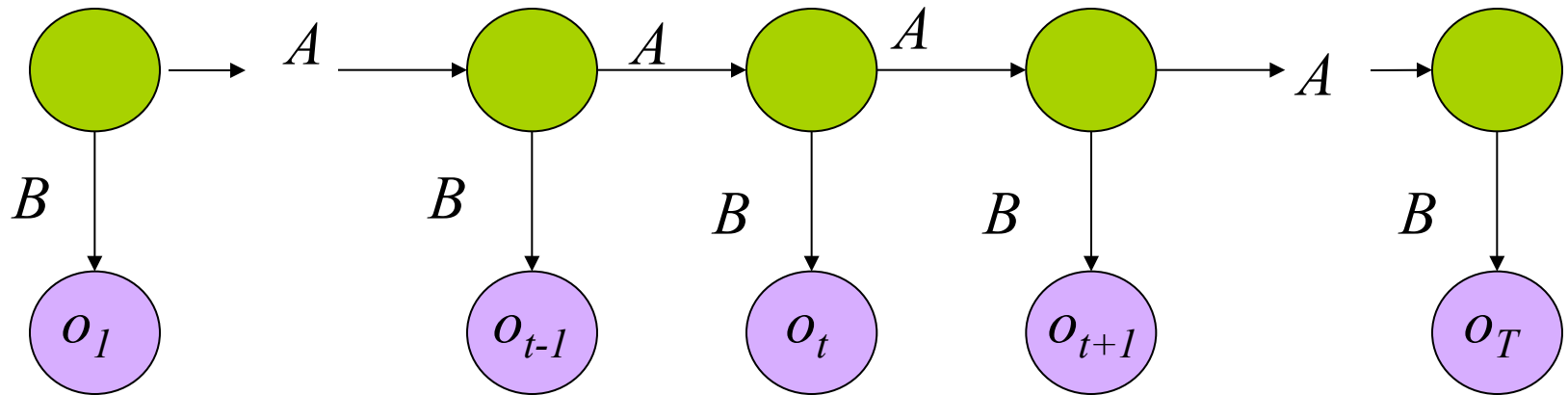
$$\mu(x_{t-1}) = \max_{x_t} p(o_t | x_t) p(x_t | x_{t-1}) \mu(x_t) \quad 2 \leq t \leq T$$

$$\mu(x_T) = 1$$

$$x_1^* = \arg \max_{x_1} p(o_1 | x_1) p(x_1) \mu(x_1)$$

$$x_t^* = \arg \max_{x_t} p(o_t | x_t) p(x_t | x_{t-1}^*) \mu(x_t)$$

Learning HMM – Parameter Estimation



- Given an observation sequence, find the model that is most likely to produce that sequence.
- Given a model and observation sequence, update the model parameters to better fit the observations.

Learning HMM

- Given a data set of i.i.d observed sequences

$$\Omega = \{ \mathbf{o}^1 \cdots \mathbf{o}^N \}$$

$\mathbf{o}^n = o_{1:T_n}^n$ is the n^{th} observation sequence of length T_n

- We seek the HMM transition matrix \mathbf{A} , emission matrix \mathbf{B} , and initial vector \mathbf{a} that are most likely to have generated the data
- We assume for simplicity that we know the number of hidden states and the number of observed states
- Approaches:
 - Expectation maximization
 - Gradient ascent
- Beware of multiple optima

Learning HMM

- Given a data set of i.i.d observed sequences

$$\Omega = \{ \mathbf{o}^1 \cdots \mathbf{o}^N \}$$

$\mathbf{o}^n = o_{1:T_n}^n$ is the n^{th} observation sequence of length T_n

- We seek the HMM transition matrix \mathbf{A} , emission matrix \mathbf{B} , and initial vector \mathbf{a} that are most likely to have generated the data
- We assume for simplicity that we know the number of hidden states and the number of observed states

Learning HMM from data

- Parameter estimation
- If we knew the state sequence it would be easy to estimate the parameters
- But we need to work with hidden state sequences
- Use “expected” counts of state transitions

Learning without hidden information

- Transition probabilities


Number of transitions from state k to state l



$$a_{lk} = \frac{n_{k \rightarrow l}}{\sum_{l'} n_{k \rightarrow l'}}$$

- Emission probabilities

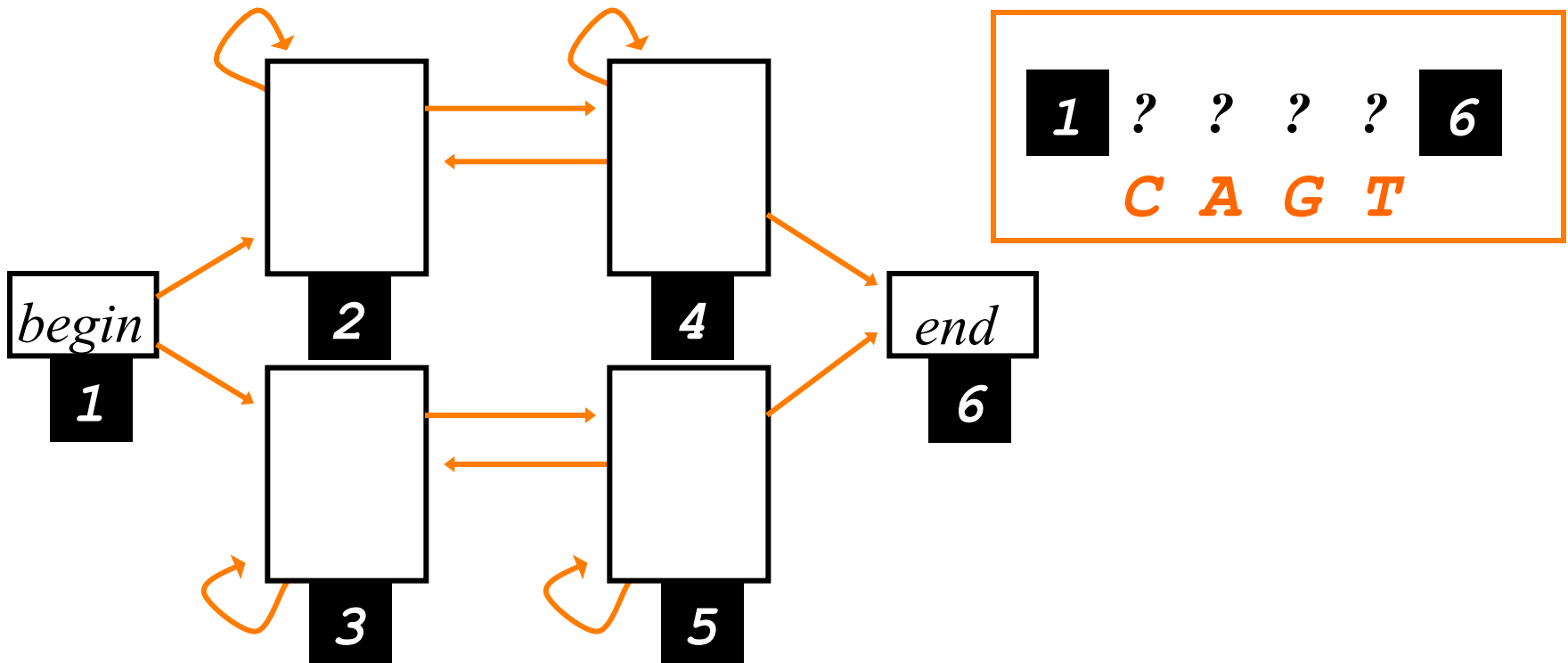
Number of times c is emitted from k



$$n_{kc} = \frac{n_{kc}}{\sum_{c'} n_{kc'}}$$

Learning in the presence of hidden states

- Since we don't know the true hidden sequence for each observation sequence, consider all possible hidden sequences



- Estimate parameters through a procedure that counts the expected number of times each parameter is used across the training set

The Baum-Welch algorithm

- Also known as Forward-backward algorithm
- An Expectation Maximization (EM) algorithm
 - EM is a family of algorithms for learning probabilistic models in problems that involve hidden states
 - Expectation: Estimate the “expected” number of times there are transitions and emissions (using current values of parameters)
 - Maximization: Estimate parameters given expected counts

Learning parameters: the Baum-Welch algorithm

- algorithm sketch:
 - initialize parameters of model
 - iterate until convergence
 - E-step: calculate the *expected* number of times each transition or emission is used
 - M-step: adjust the parameters to *maximize* the likelihood of these expected values

EM algorithm

M-step: Maximize:

$$\sum_{n=1}^N \left\langle \log p(o_1^n, o_2^n \cdots o_{T^n}^n, x_1^n, x_2^n, \cdots x_{T^n}^n) \right\rangle_{p^{old}(x_{1:T^n}^n | \mathbf{o}^n)} \quad \text{wrt. } \mathbf{A}, \mathbf{B}, \mathbf{a}.$$

which because of the form of the HMM is same as maximizing

$$\sum_{n=1}^N \left\{ \left\langle \log p(x_1^n) \right\rangle_{p^{old}(x_1^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n-1} \left\langle \log p(x_{t+1}^n | x_t^n) \right\rangle_{p^{old}(x_t^n, x_{t+1}^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n} \left\langle \log p(o_t^n | x_t^n) \right\rangle_{p^{old}(x_t^n | \mathbf{o}^n)} \right\}$$

Optimizing the above with respect to $p(x_1^n)$ i.e., \mathbf{a} ,

$$a_i^{new} = p^{new}(x_1 = i) = \frac{1}{N} \sum_{n=1}^N p^{old}(x_1^n = i | \mathbf{o}^n)$$

which is simply the averaged number of times (with respect to p^{old}) that the first hidden variable is in state i .

EM algorithm

M-step: Maximize:

$$\sum_{n=1}^n \left\{ \left\langle \log p(x_1^n) \right\rangle_{p^{old}(x_1^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n-1} \left\langle \log p(x_{t+1}^n | x_t^n) \right\rangle_{p^{old}(x_t^n, x_{t+1}^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n} \left\langle \log p(o_t^n | x_t^n) \right\rangle_{p^{old}(x_t^n | \mathbf{o}^n)} \right\}$$

Optimizing the above with respect to \mathbf{A}

$$A_{j,i}^{new} = p^{new}(x_{t+1} = j | x_t = i) \propto \sum_{n=1}^N \sum_{t=1}^{T_n-1} p^{old}(x_t^n = i, x_{t+1}^n = j | \mathbf{o}^n)$$

which is simply the number of times a transition from hidden state j to i occurs averaged over all times (\because of stationarity) and all training sequences.

Normalizing, we have:

$$A_{j,i}^{new} = \frac{\sum_{n=1}^N \sum_{t=1}^{T_n-1} p^{old}(x_t^n = i, x_{t+1}^n = j | \mathbf{o}^n)}{\sum_j \sum_{n=1}^N \sum_{t=1}^{T_n-1} p^{old}(x_t^n = i, x_{t+1}^n = j | \mathbf{o}^n)}$$

EM algorithm

- M-step: Maximize:

$$\sum_{n=1}^N \left\{ \langle \log p(x_1^n) \rangle_{p^{old}(x_1^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n-1} \langle \log p(x_{t+1}^n | x_t^n) \rangle_{p^{old}(x_t^n, x_{t+1}^n | \mathbf{o}^n)} + \sum_{t=1}^{T_n} \langle \log p(o_t^n | x_t^n) \rangle_{p^{old}(x_t^n | \mathbf{o}^n)} \right\}$$

Optimizing the above with respect to **B**

$$B_{k,i}^{new} = p^{new}(o_t = k | x_t = i) \propto \sum_{n=1}^N \sum_{t=1}^{T_n} \mathbf{I}[o_t^n = k] p^{old}(x_t^n = i | \mathbf{o}^n)$$

which is the expected number of times observation k occurs when the hidden state is i

Normalizing, we have:

$$B_{k,i}^{new} = \frac{\sum_{n=1}^N \sum_{t=1}^{T_n} \mathbf{I}[o_t^n = k] p^{old}(x_t^n = i | \mathbf{o}^n)}{\sum_k \sum_{n=1}^N \sum_{t=1}^{T_n} \mathbf{I}[o_t^n = k] p^{old}(x_t^n = i | \mathbf{o}^n)}$$

EM Algorithm

E-step: Calculate

$p^{old} \left(x_1^n = i \mid \mathbf{o}^n \right)$ smoothing (inferring the past)

$p^{old} \left(x_t^n = i, x_{t+1}^n = j \mid \mathbf{o}^n \right)$ the pairwise marginal

$p^{old} \left(x_t^n = i \mid \mathbf{o}^n \right)$ smoothing (inferring the past)

E-step

$$p(o_{1:T}, x_t = i) = \alpha(x_t = i) \beta(x_t = i)$$

$$p(x_t = i | o_{1:T}) = \frac{p(o_{1:T}, x_t = i)}{p(o_{1:T})} = \frac{\alpha(x_t = i) \beta(x_t = i)}{p(o_{1:T})}$$

$$p(x_t = i, x_{t+1} = j | o_{1:T}) = \frac{\alpha(x_t = i) a_{ji} b_{j, o_{t+1}} \beta(x_{t+1} = j)}{p(o_{1:T})}$$

EM Algorithm

Parameter initialization

- EM algorithm converges to a local maximum of the likelihood
- In general, there is no guarantee that the algorithm will find a global maximum
- The number of local maxima can be exponential in the number of hidden states
- Parameter initialization can be critical for the quality of solution
- In practice:
 - Try different random initializations
 - Randomize the order of presentation of training examples and perform online update
 - Initialize emission parameters based on results of fitting a simpler (non temporal) mixture model

HMM Parameter estimation in practice

Sparseness of data requires

- Smoothing of estimates using priors – replace ML estimates by MAP estimates
- Domain specific tricks – Feature decomposition (capitalized?, number?, etc. in text processing)
- Shrinkage allows pooling of estimates over multiple states of same type
- Well designed (or learned) HMM topology

HMM Variations

Continuous vector-valued observations

- Need a model $p(\mathbf{o}_t | x_t)$ mapping the hidden state at time t to a distribution over outputs
- Does not change any of the update equations for filtering, smoothing, etc.
- However, for learning, we need the normalization constant.

Mixture emissions

- Emission probability is a mixture:
$$p(o_t | x_t) = \sum_k p(o_t | k_t, x_t) p(k_t | x_t)$$
- EM algorithm has a nested “emission” EM loop.

HMM-GMM

- $$p(\mathbf{o}_t | k_t, x_t) = \mathcal{N}(\mathbf{o}_t | \mu_{k_t, x_t}, \Sigma_{k_t, x_t})$$

Dynamic Bayesian Networks

$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = \prod_{t=1}^T \prod_{i=1}^M p(x_i(t) \mid \mathbf{x}_{\setminus i}(t), \mathbf{x}(t-1))$$

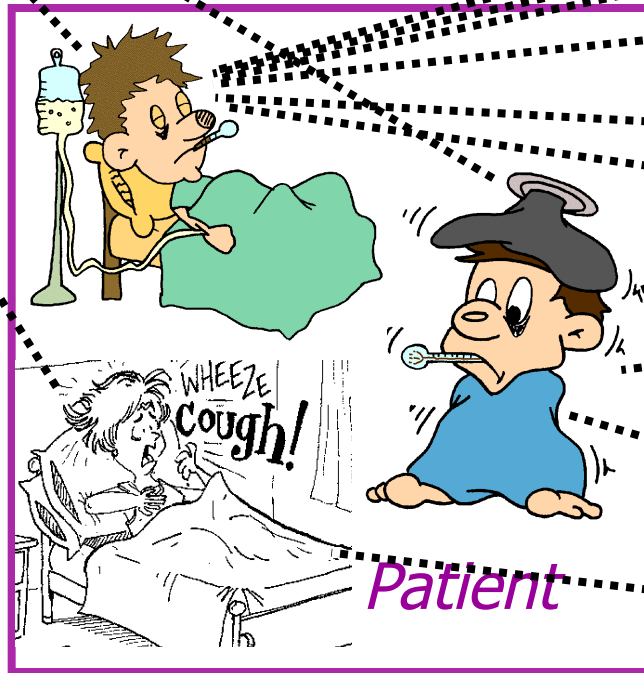
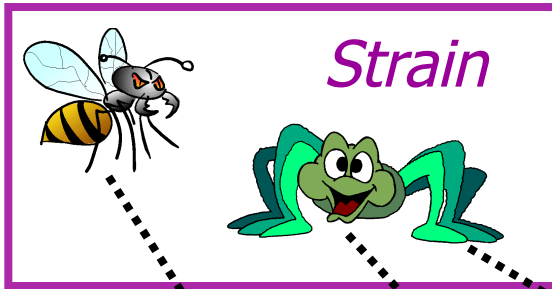
Probabilistic relational models

- Observation: the world consists of many distinct entities with similar properties and relations
- First order logic exploits this redundancy to make concise statements about the world
- $\forall s \in S \forall c \in C \text{ Student}(s,c) \text{ and Easy}(c) \Rightarrow \text{Happy}(s)$
- Unfortunately, the real world is not so clear-cut
- Need a probabilistic counterpart of First order logic
- Bayes networks : Propositional logic :: ? : First order logic?
- ? = Probabilistic Relational Models (under the finite domain assumption as in relational databases)

PRMs

- Developed by Daphne Koller's group at Stanford
 - representation: Avi Pfeffer
 - Builds on work in KBMC (knowledge-based model construction) by Haddawy, Poole, Wellman and others...
 - Object Oriented Bayesian Networks
 - Relational Probability Models
 - Learning: Lise Getoor, Nir Friedman, Avi Pfeffer, Ben Taskar
 - Attribute Uncertainty
 - Structural Uncertainty
 - Class Uncertainty
 - Identity Uncertainty
 - Undirected models: Ben Taskar, Eran Segal

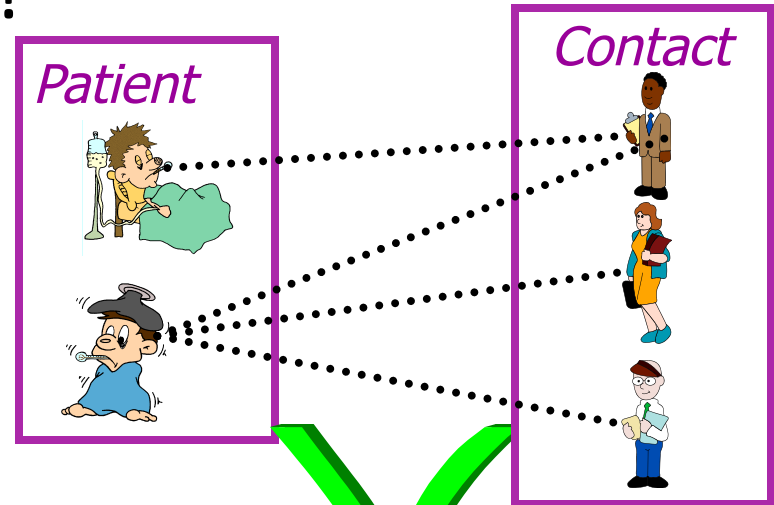
Probabilistic Relational Models



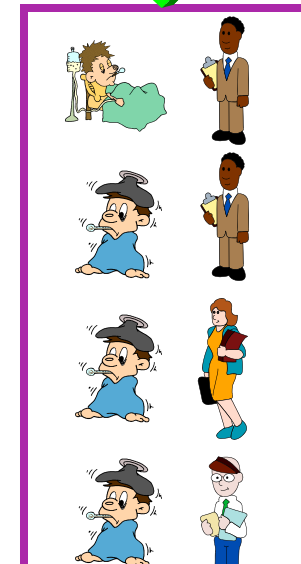
Why PRM?

Traditional approaches

- Flat representation
- Fixed number of features
- IID Samples



flatten

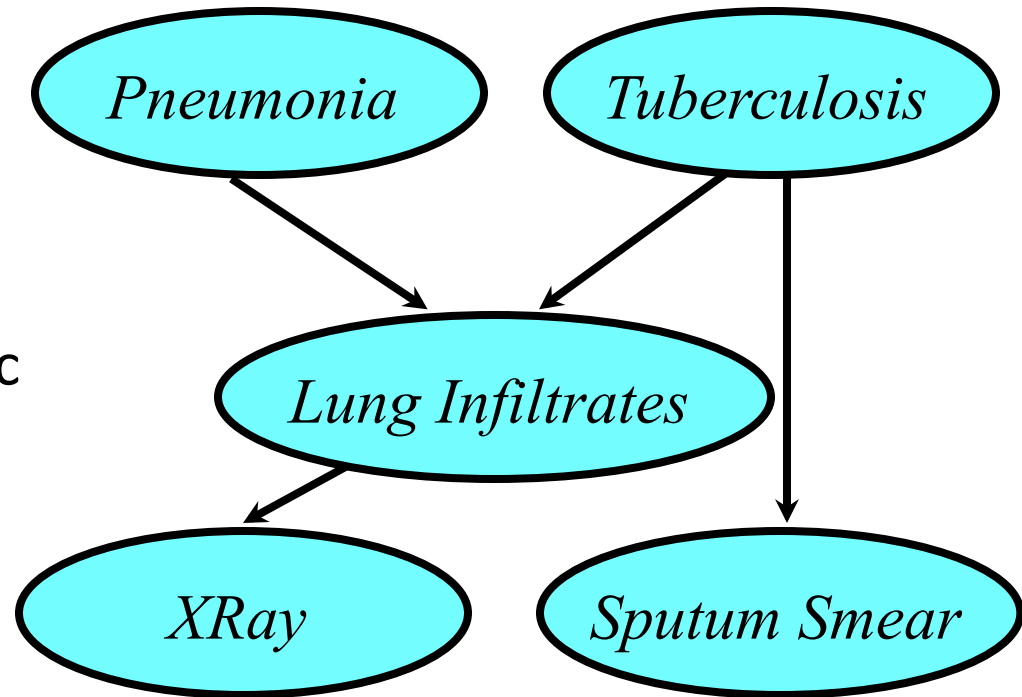


Problems:

- Introduce statistical skew
- Discard relational structure
- Must fix features in advance

Bayesian Networks

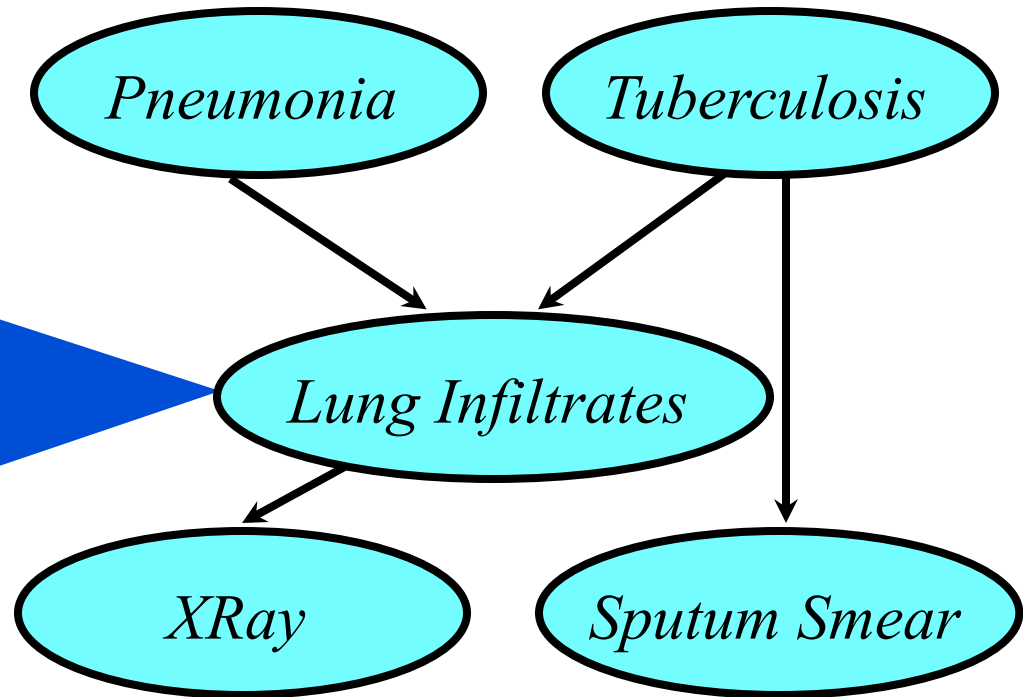
nodes = random variables
 edges = direct probabilistic
 influence



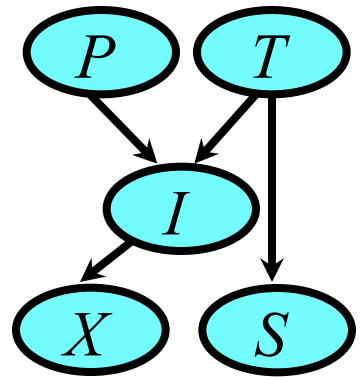
Network structure encodes independence assumptions:
XRay conditionally independent of *Pneumonia* given *Infiltrates*

Bayesian Networks

<i>P</i>	<i>T</i>	$P(I P, T)$	
<i>p</i>	<i>t</i>	0.8	0.2
<i>p</i>	\bar{t}	0.6	0.4
\bar{p}	<i>t</i>	0.2	0.8
\bar{p}	\bar{t}	0.01	0.99



BN Semantics



conditional independencies in BN structure

+ local probability models

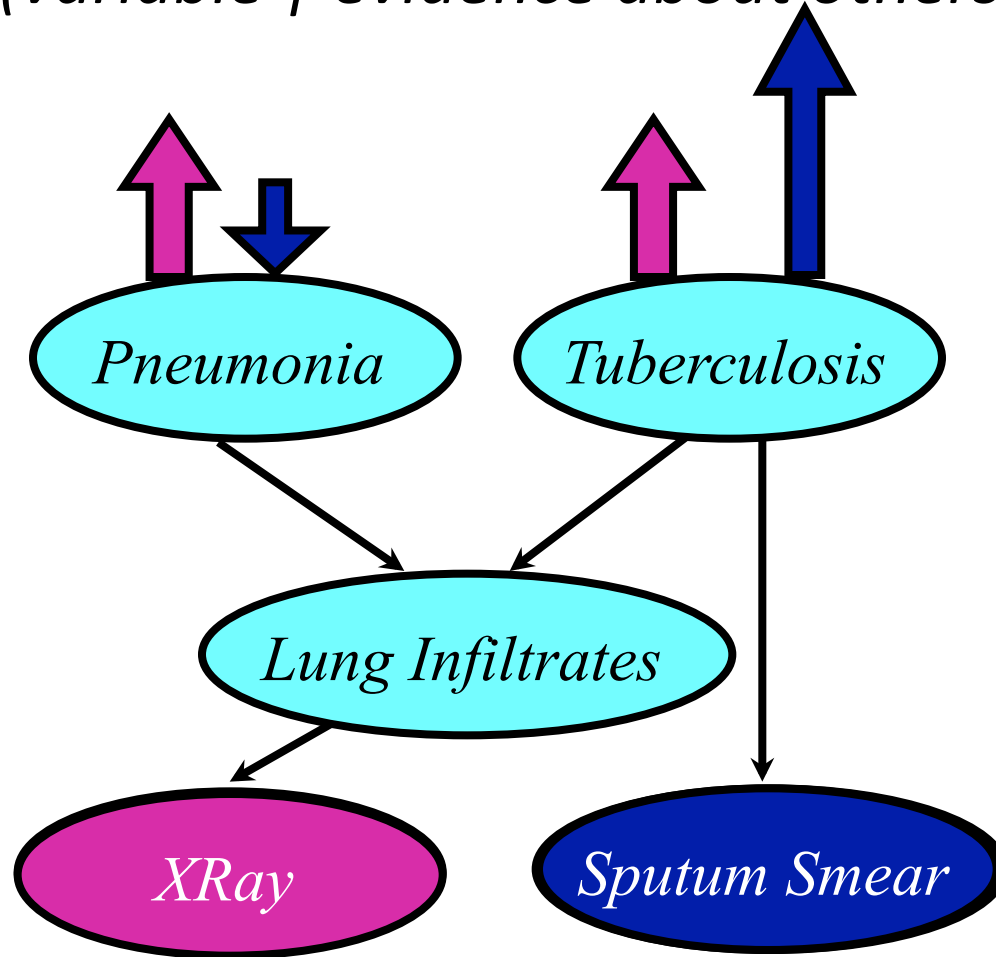
=

full joint distribution over domain

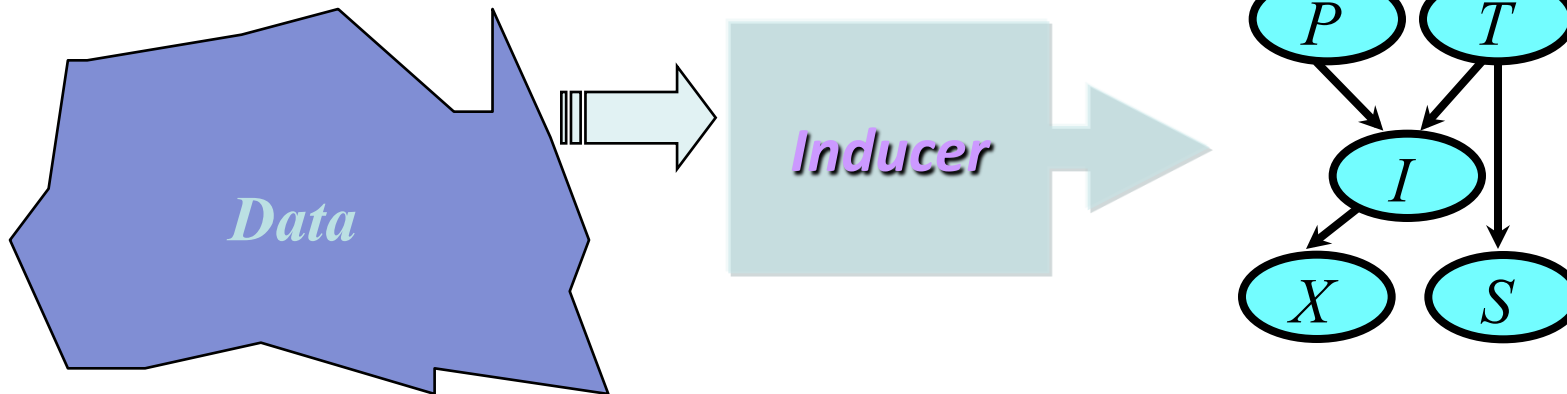
$$P(\bar{p}, t, i, x, \bar{s}) = P(\bar{p}) P(t) P(i | \bar{p}, t) P(x | i) P(\bar{s} | t)$$

Queries

*Full joint distribution specifies answer to any query:
 $P(\text{variable} \mid \text{evidence about others})$*



BN Learning



- BN models can be learned from empirical data
 - parameter estimation via numerical optimization
 - structure learning via combinatorial search.

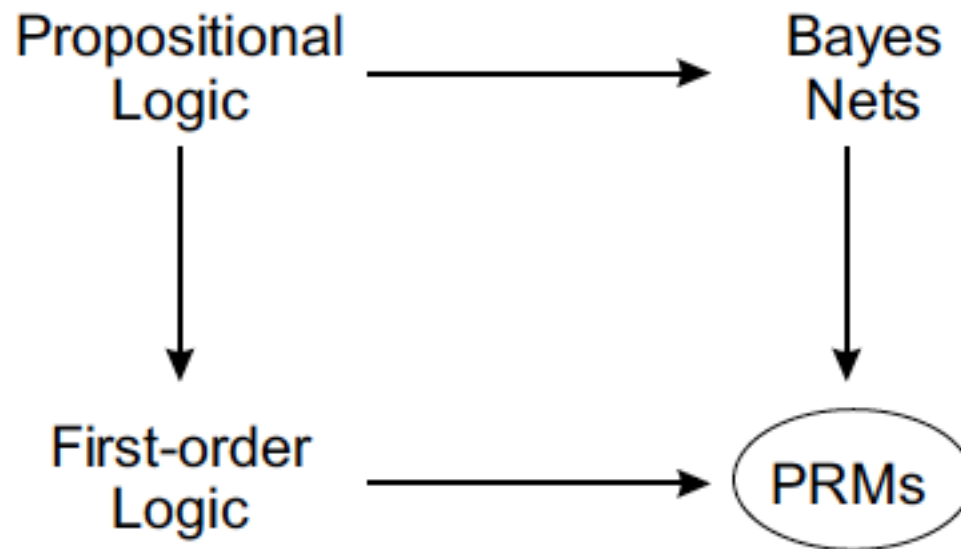
- BN hypothesis space biased towards distributions with independence structure.

Probabilistic Relational Models

- Combine advantages of relational representations and Bayesian networks:
 - World models that represent: objects, properties, relations
 - Compact, natural probability models
 - Integrate uncertainty with relational model:
 - Properties of objects can depend on properties of related objects
 - Can model uncertainty over the relational structure of domain

Motivation

- Unfortunately, the real world is not so clear-cut
- Need a probabilistic version of FOL
- Proposal: PRMs



Example

- a set of students, $\mathcal{S} = \{s_1, s_2, s_3\}$
- a set of professors, $\mathcal{P} = \{p_1, p_2, p_3\}$
- **Well-Funded, Famous** : $\mathcal{P} \rightarrow \{true, false\}$
- **Student-Of** : $\mathcal{S} \times \mathcal{P} \rightarrow \{true, false\}$
- **Successful** : $\mathcal{S} \rightarrow \{true, false\}$

Example

We can express a certain self-evident fact in one sentence of FOL:

$$\forall s \in \mathcal{S} \quad \forall p \in \mathcal{P}$$

Famous(p) and **Student-Of**(s, p)

\Rightarrow **Successful**(s)

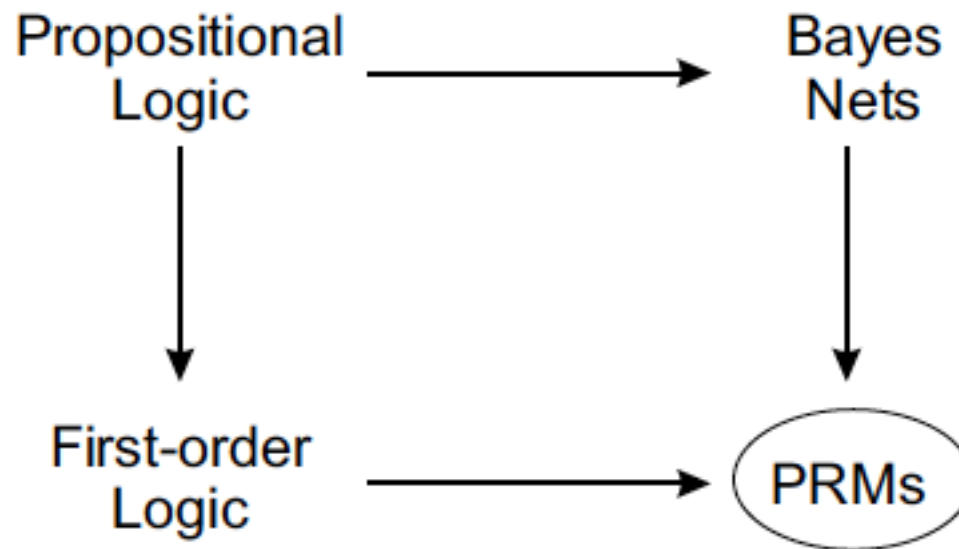
Example

The same sentence converted to propositional logic:

$(\neg(p_1_famous \text{ and } student_of_s_1_p_1) \text{ or } s_1_successful)$ and
 $(\neg(p_1_famous \text{ and } student_of_s_2_p_1) \text{ or } s_2_successful)$ and
 $(\neg(p_1_famous \text{ and } student_of_s_3_p_1) \text{ or } s_3_successful)$ and
 $(\neg(p_2_famous \text{ and } student_of_s_1_p_1) \text{ or } s_1_successful)$ and
 $(\neg(p_2_famous \text{ and } student_of_s_2_p_1) \text{ or } s_2_successful)$ and
 $(\neg(p_2_famous \text{ and } student_of_s_3_p_1) \text{ or } s_3_successful)$ and
 $(\neg(p_3_famous \text{ and } student_of_s_1_p_1) \text{ or } s_1_successful)$ and
 $(\neg(p_3_famous \text{ and } student_of_s_2_p_1) \text{ or } s_2_successful)$ and
 $(\neg(p_3_famous \text{ and } student_of_s_3_p_1) \text{ or } s_3_successful)$

Example

- Unfortunately, the real world is not so clear-cut
- Need a probabilistic version of FOL
- Proposal: PRMs



Motivation

- Most real-world data are stored in relational DBMS
- Few learning algorithms are capable of handling data in its relational form; thus we have to resort to “flattening” the data in order to do analysis
- As a result, we lose relational information which might be crucial to understanding the data

Motivation

- The world consists of base entities, partitioned into classes X_1, X_2, \dots, X_n
- Elements of these classes share connections via a collection of relations R_1, R_2, \dots, R_m
- Each entity type is characterized by a set of attributes, $\mathcal{A}(X_i)$. Each attribute $A_j \in \mathcal{A}(X_i)$ assumes values from a fixed domain, $V(A_j)$
- Defines the *schema* of a relational model

Motivation

We can modify the domain previously given to this new framework:

- 2 classes: \mathcal{S}, \mathcal{P}
- 1 relation: $\text{Student-Of} \subset \mathcal{S} \times \mathcal{P}$
- $\mathcal{A}(\mathcal{S}) = \{\text{Success}\}$
- $\mathcal{A}(\mathcal{P}) = \{\text{Well-Funded, Famous}\}$

Motivation

An instantiation \mathcal{I} of the relational schema defines

- a concrete set of base entities $\mathcal{O}^{\mathcal{I}}(X_i)$ for each class X_i
- values for the attributes of each base entity for each class
- $R_i(X_1, \dots, X_k) \subset \mathcal{O}^{\mathcal{I}}(X_1) \times \dots \times \mathcal{O}^{\mathcal{I}}(X_k)$ for each R_i .

What are PRMs?

- The starting point of this work is the structured representation of probabilistic models of Bayesian networks (BNs).
 - BNs for a given domain involves a pre-specified set of attributes whose relationship to each other is fixed in advance
- PRMs conceptually extend BNs to allow the specification of a probability model for *classes* of objects rather than a fixed set of attributes
- PRMs also allow properties of an entity to depend probabilistically on properties of other related entities

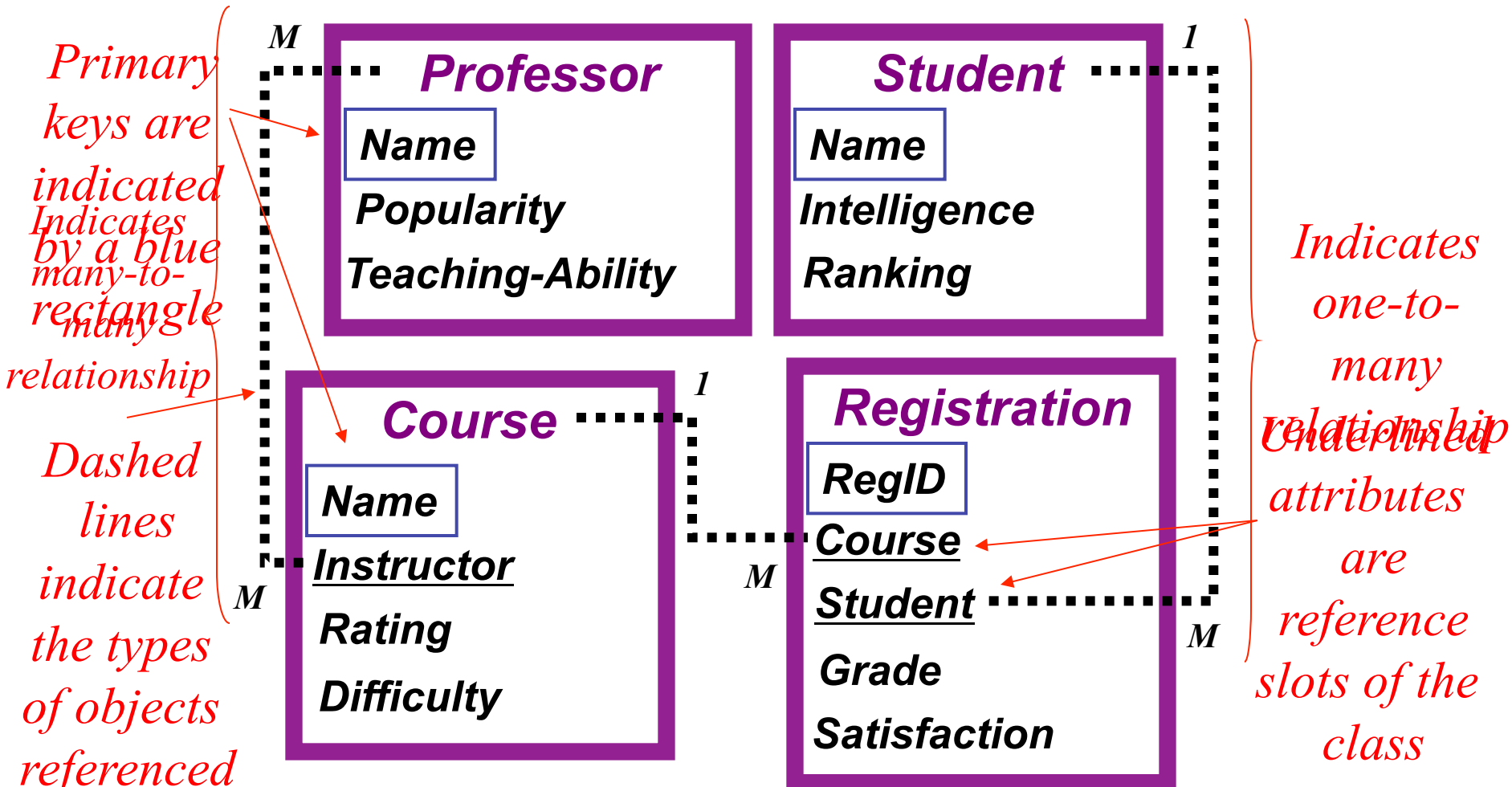
Mapping PRMs from Relational Models

- The representation of PRMs is a direct mapping from that of relational databases
- A relational model consists of a set of *classes* X_1, \dots, X_n and a set of *relations* R_1, \dots, R_m , where each relation R_i is typed
- Each class or entity type (corresponding to a single relational table) is associated with a set of *attributes* $\mathcal{A}(X_i)$ and a set of *reference slots* $\mathcal{R}(X)$

PRM Semantics

- Reference slots correspond to attributes that are foreign keys (key attributes of another table)
- $X.\rho$, is used to denote reference slot ρ of X . Each reference slot ρ is typed according to the relation that it references

University Domain Example - Relational Schema



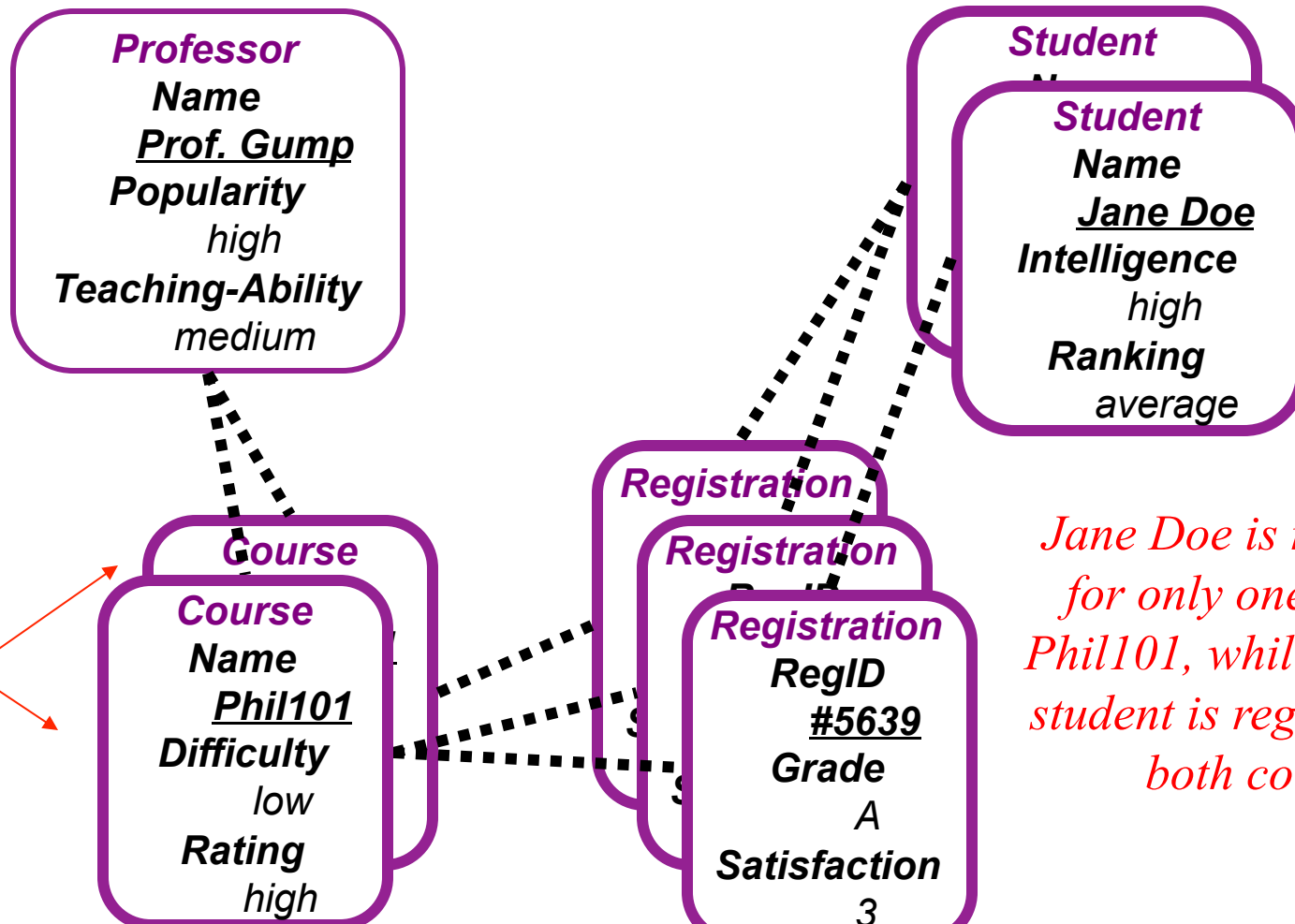
PRM Semantics Continued

- Each attribute $A_j \in \mathcal{A}(X_i)$ takes on values in some fixed domain of possible values denoted $V(A_j)$. We assume that value spaces are finite
- Attribute A of class X is denoted $X.A$
- For example, the **Student** class has an *Intelligence* attribute and the value space or domain for **Student.Intelligence** might be $\{high, low\}$

PRM Semantics Continued

- An *instance* I of a schema specifies a set of objects x , partitioned into classes; such that there is a value for each attribute $x.A$ and a value for each reference slot $x.\rho$
- $\mathcal{A}(x)$ is used as a shorthand for $\mathcal{A}(X)$, where x is of class X . For each object x in the instance and each of its attributes A , we use $I_{x.A}$ to denote the value of $x.A$ in I

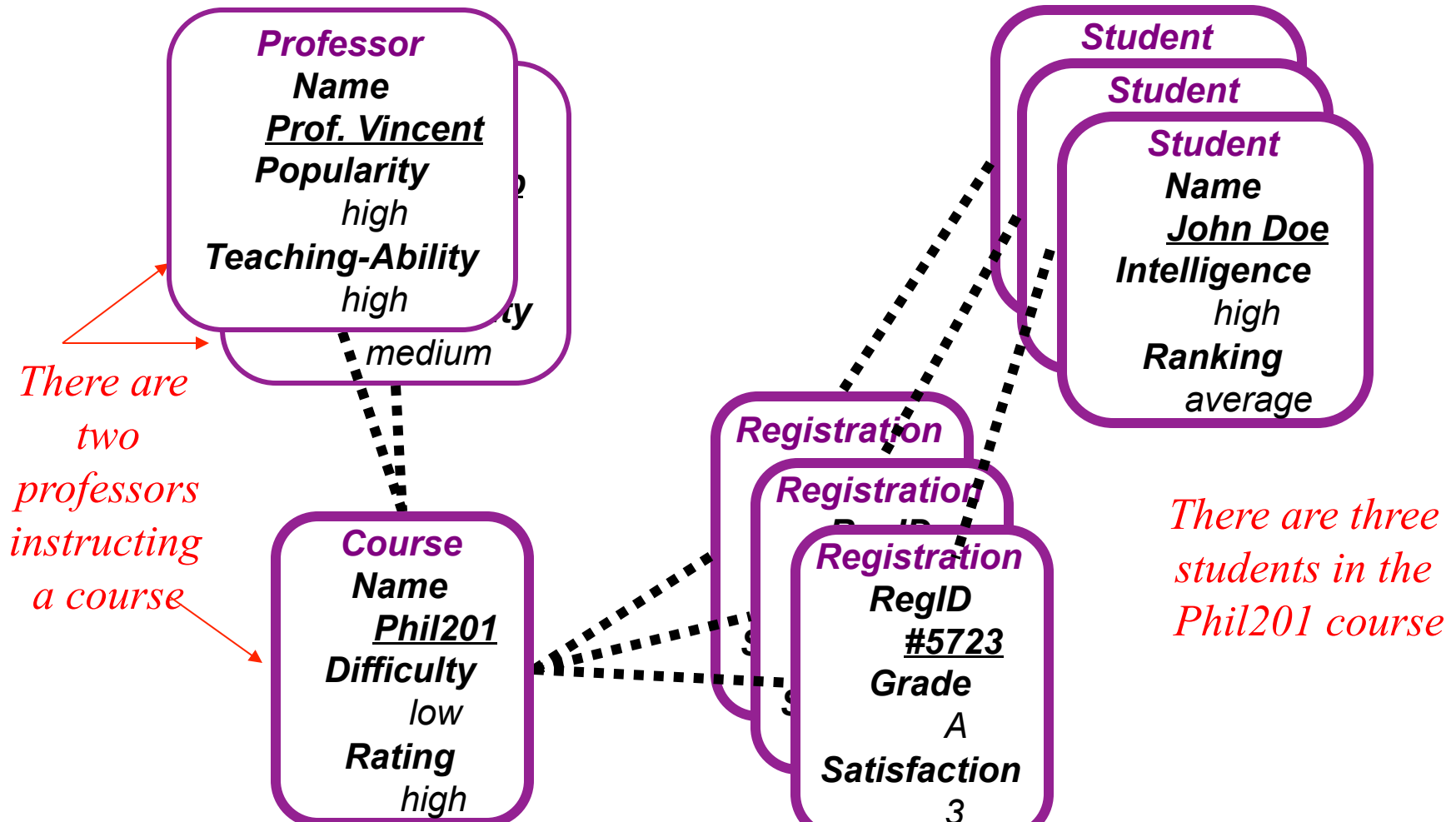
University Domain Example – An Instance of the Schema



One professor is the instructor for both courses

Jane Doe is registered for only one course, Phil101, while the other student is registered for both courses

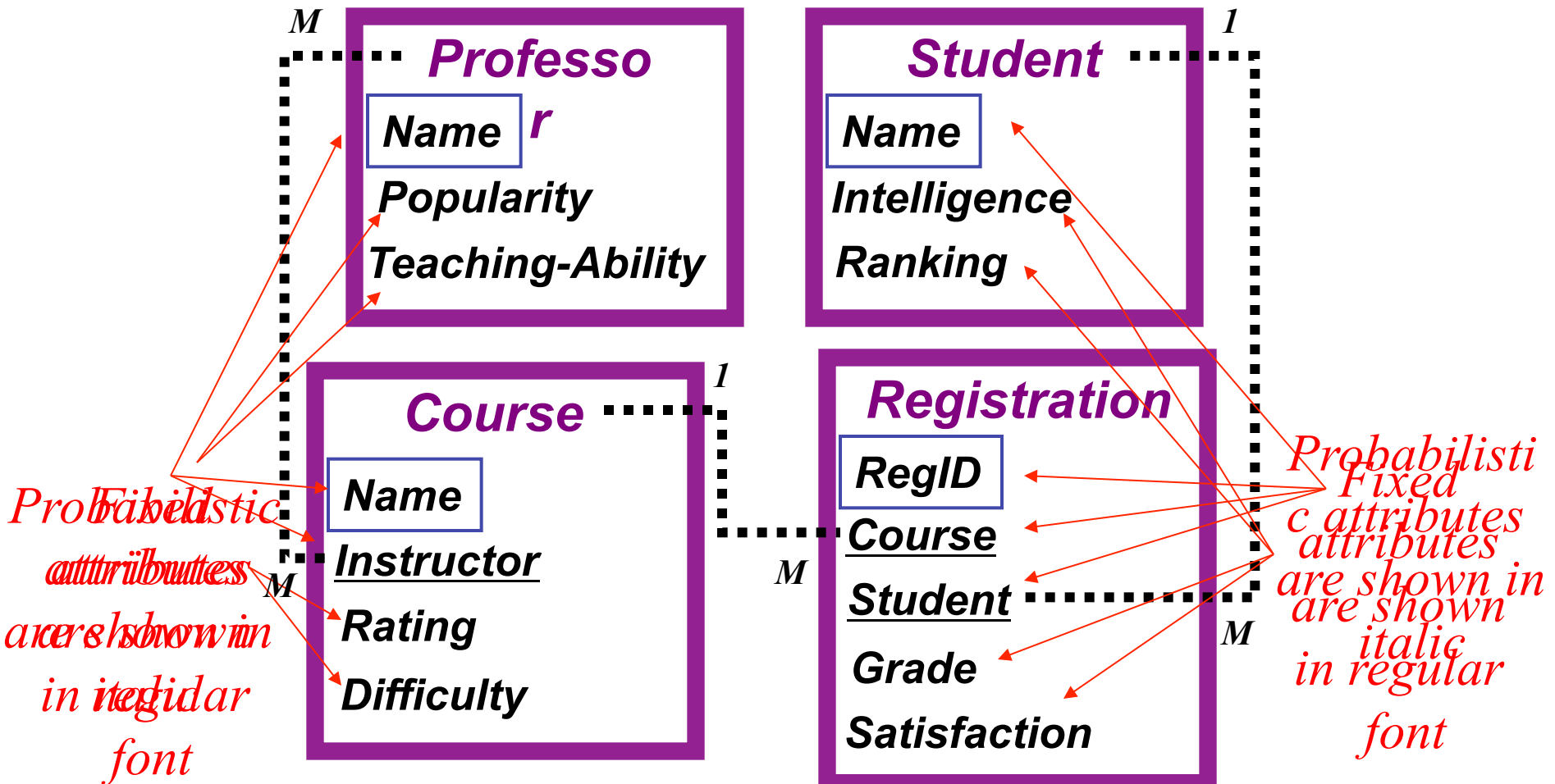
University Domain Example



PRM Semantics Continued

- Some attributes, such as name or social security number, are fully determined. Such attributes are labeled as *fixed*. Assume that they are known in any instantiation of the schema
- The other attributes are called *probabilistic*

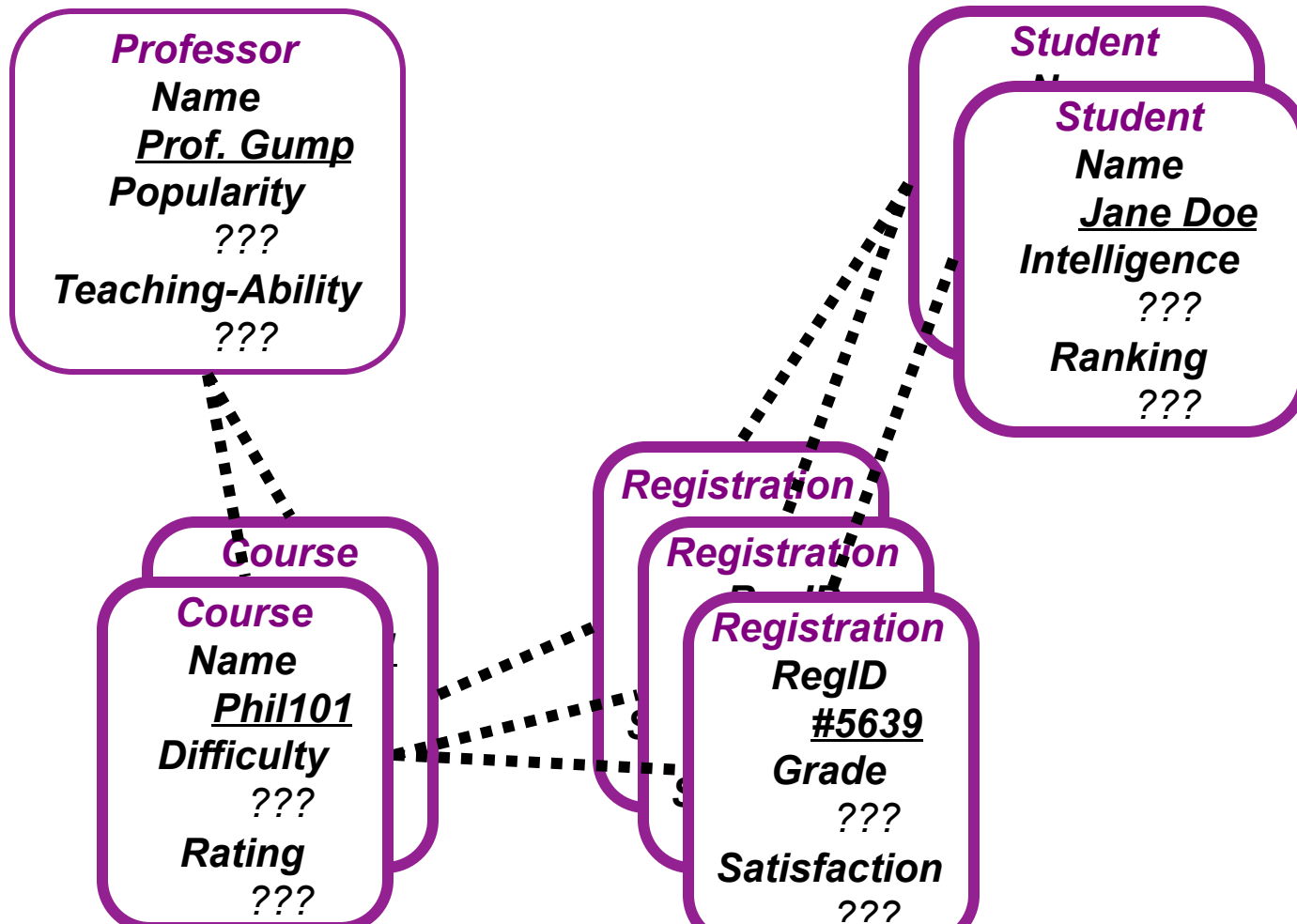
University Domain Example - Relational Schema



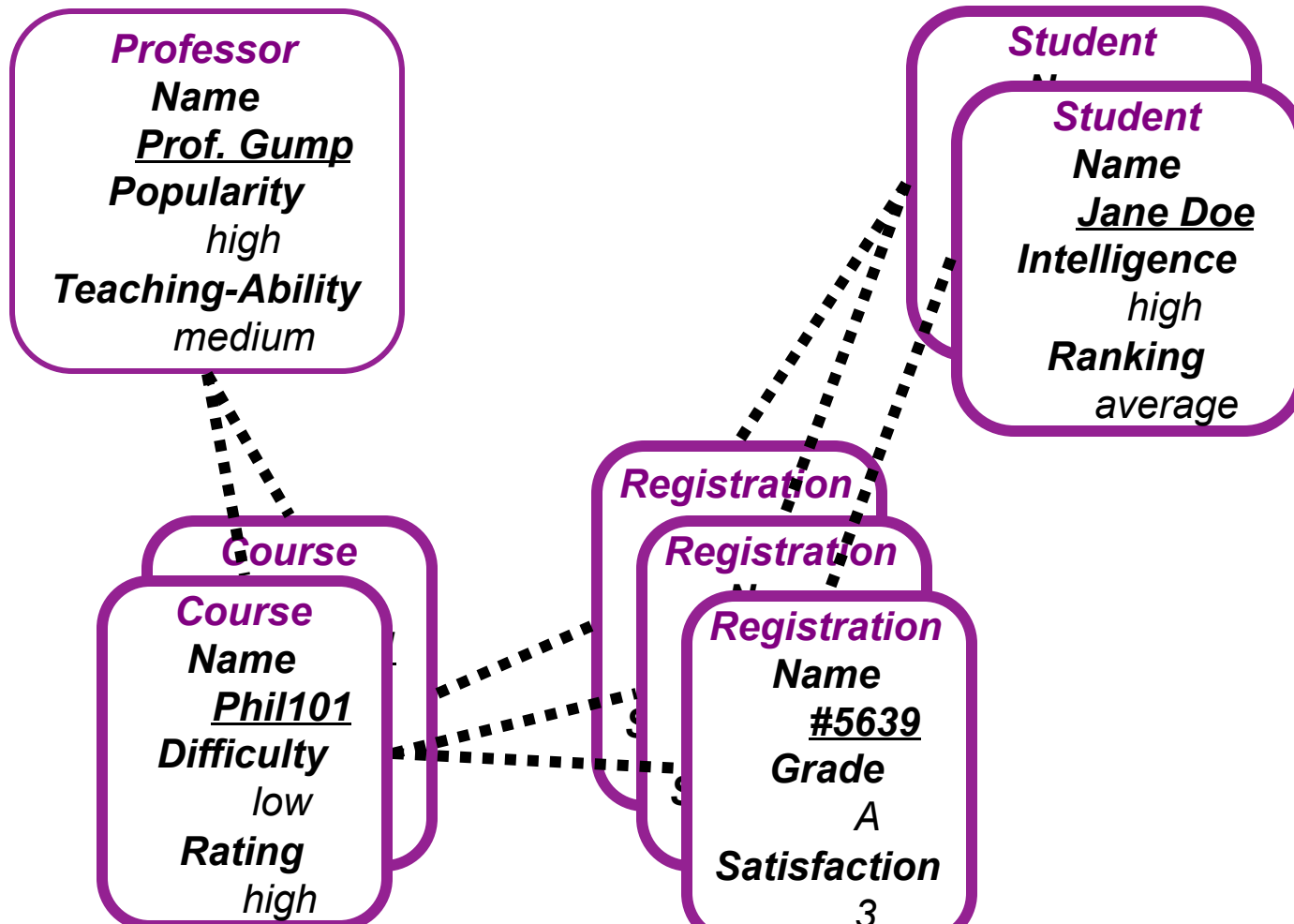
PRM Semantics Continued

- A *skeleton structure* σ of a relational schema is a partial specification of an instance of the schema. It specifies the set of objects $\mathcal{O}(X_i)$ for each class, the values of the fixed attributes of these objects, and the relations that hold between the objects
- The values of probabilistic attributes are left unspecified
- A *completion* I of the skeleton structure σ extends the skeleton by also specifying the values of the probabilistic attributes

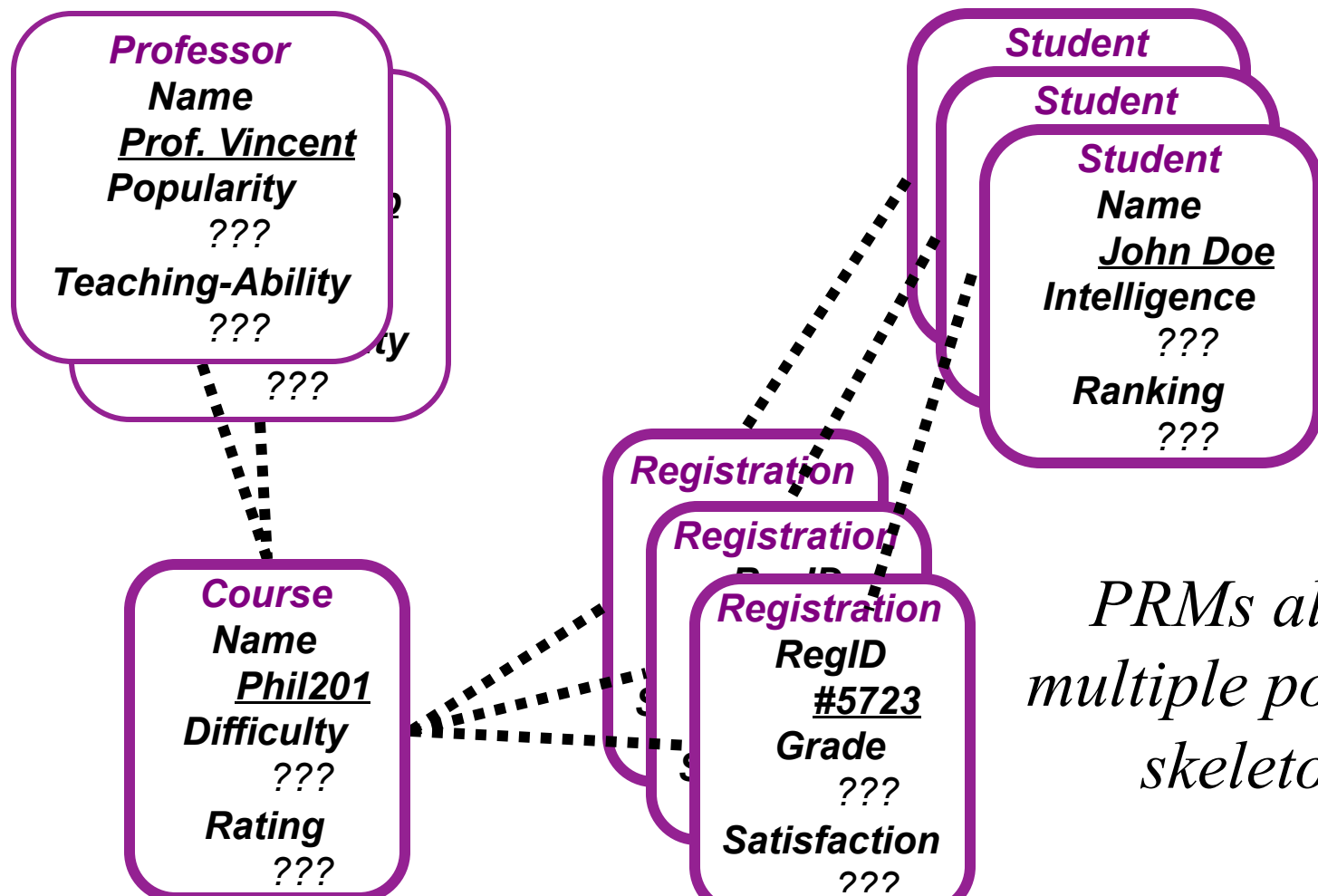
University Domain Example – Relational Skeleton



University Domain Example – The Completion Instance /

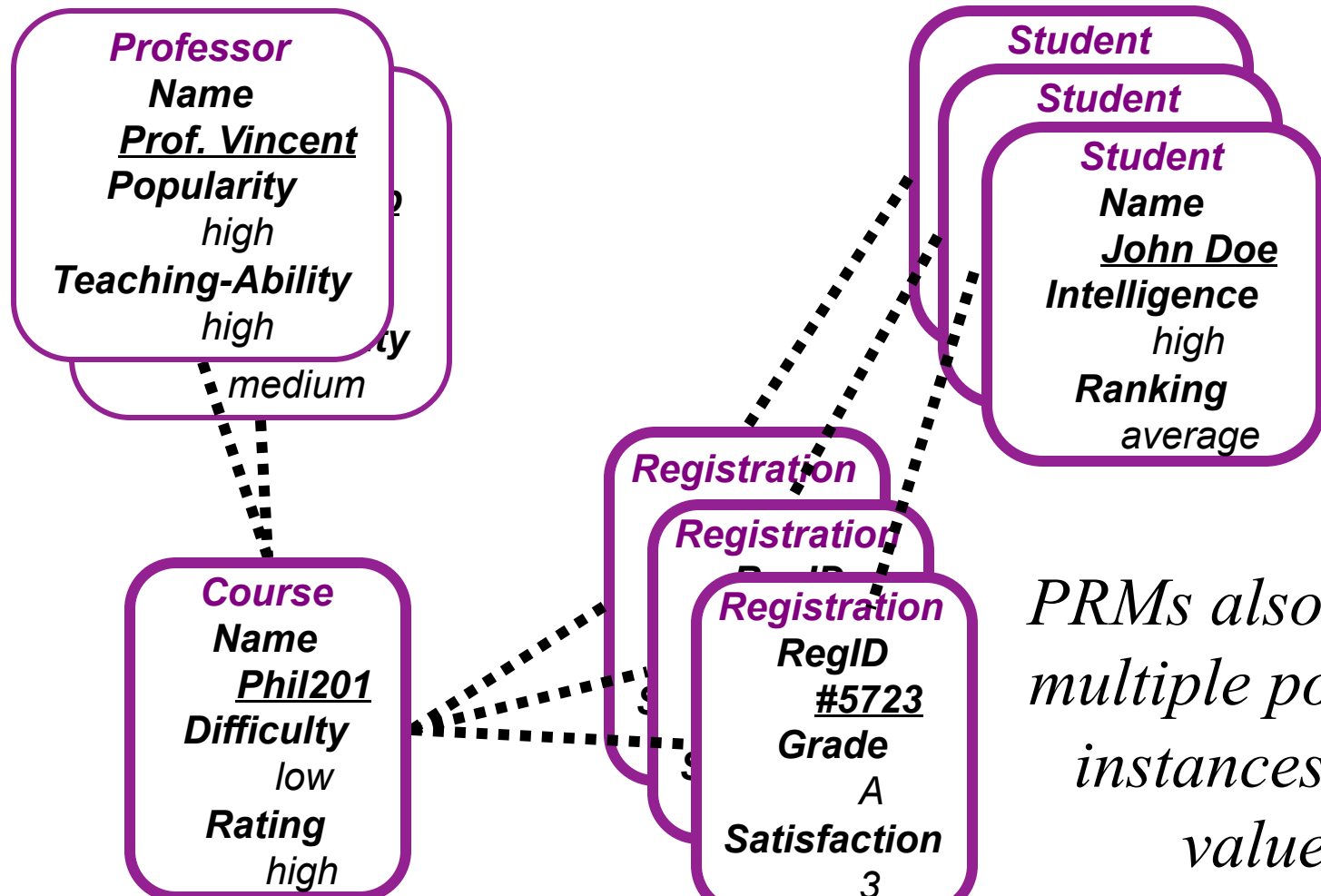


University Domain Example – Another Relational Skeleton



PRMs allow multiple possible skeletons

University Domain Example – The Completion Instance /



PRMs also allow multiple possible instances and values

More PRM Semantics

- For each reference slot ρ , we define an inverse slot, ρ^{-1} , which is the inverse function of ρ
- For example, we can define an inverse slot for the *Student* slot of **Registration** and call it *Registered-In*. Since the original relation is a one-to-many relation, it returns a set of **Registration** objects
- A final definition is the notion of a *slot chain* $\tau = \rho_1 \dots \rho_m$, which is a sequence of reference slots that defines functions from objects to other objects to which they are indirectly related. For example, **Student.Registered-In.Course.Instructor** can be used to denote a student's set of instructors

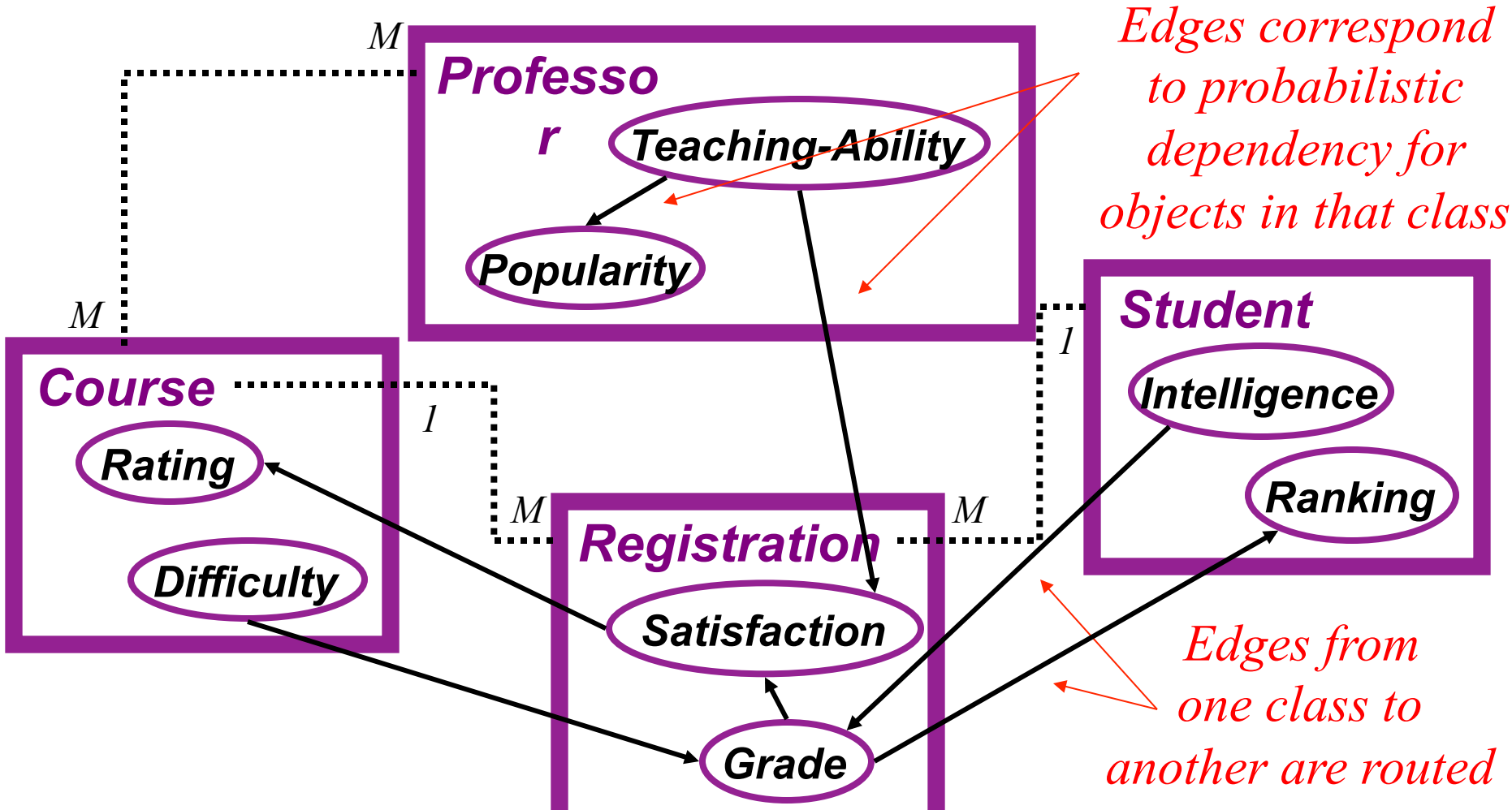
Definition of PRMs

- The probabilistic model consists of two components: the qualitative dependency structure, \mathcal{S} , and the parameters associated with it, $\theta_{\mathcal{S}}$
- The dependency structure is defined by associating with each attribute $X.A$ a set of *parents* $\text{Pa}(X.A)$; parents are attributes that are “direct influences” on $X.A$. This dependency holds for any object of class X

Definition of PRMs Cont' d

- The attribute $X.A$ can depend on another probabilistic attribute B of X . This dependence induces a corresponding dependency for individual objects
- The attribute $X.A$ can also depend on attributes of related objects $X.\tau.B$, where τ is a slot chain
- For example, given any **Registration** object r and the corresponding **Professor** object p for that instance, $r.Satisfaction$ will depend probabilistically on $r.Grade$ as well as $p.Teaching-Ability$

PRM Dependency Structure for the University Domain



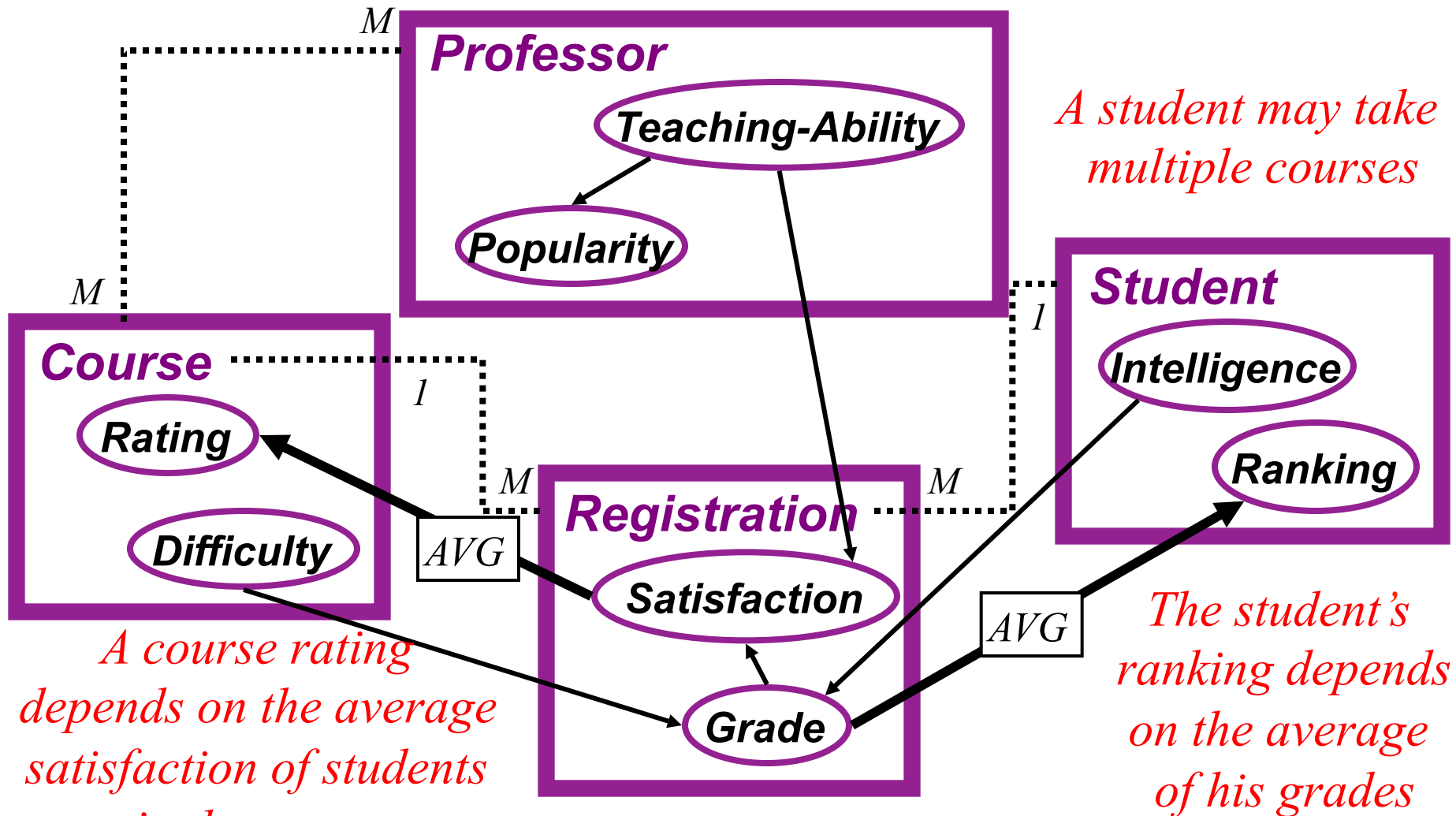
Dependency Structure in PRMs

- As mentioned earlier, $x.\tau$ represents the set of objects that are τ -relatives of x . Except in cases where the slot chain is guaranteed to be single-valued, we must specify the probabilistic dependence of $x.A$ on the multiset $\{y.B : y \in x.\tau\}$
- The notion of *aggregation* from database theory gives us the tool to address this issue; i.e., $x.a$ will depend probabilistically on some aggregate property of this multiset

Aggregation in PRMs

- Examples of aggregation are: the mode of the set (most frequently occurring value); mean value of the set (if values are numerical); median, maximum, or minimum (if values are ordered); cardinality of the set; etc
- An aggregate essentially takes a multiset of values of some ground type and returns a summary of it
- The type of the aggregate can be the same as that of its arguments, or any type returned by an aggregate. $X.A$ can have $\gamma(X.\tau.B)$ as a parent; the semantics is that for any $x \in X$, $x.a$ will depend on the value of $\gamma(x.\tau.b)$, $V(\gamma(x.\tau.b))$

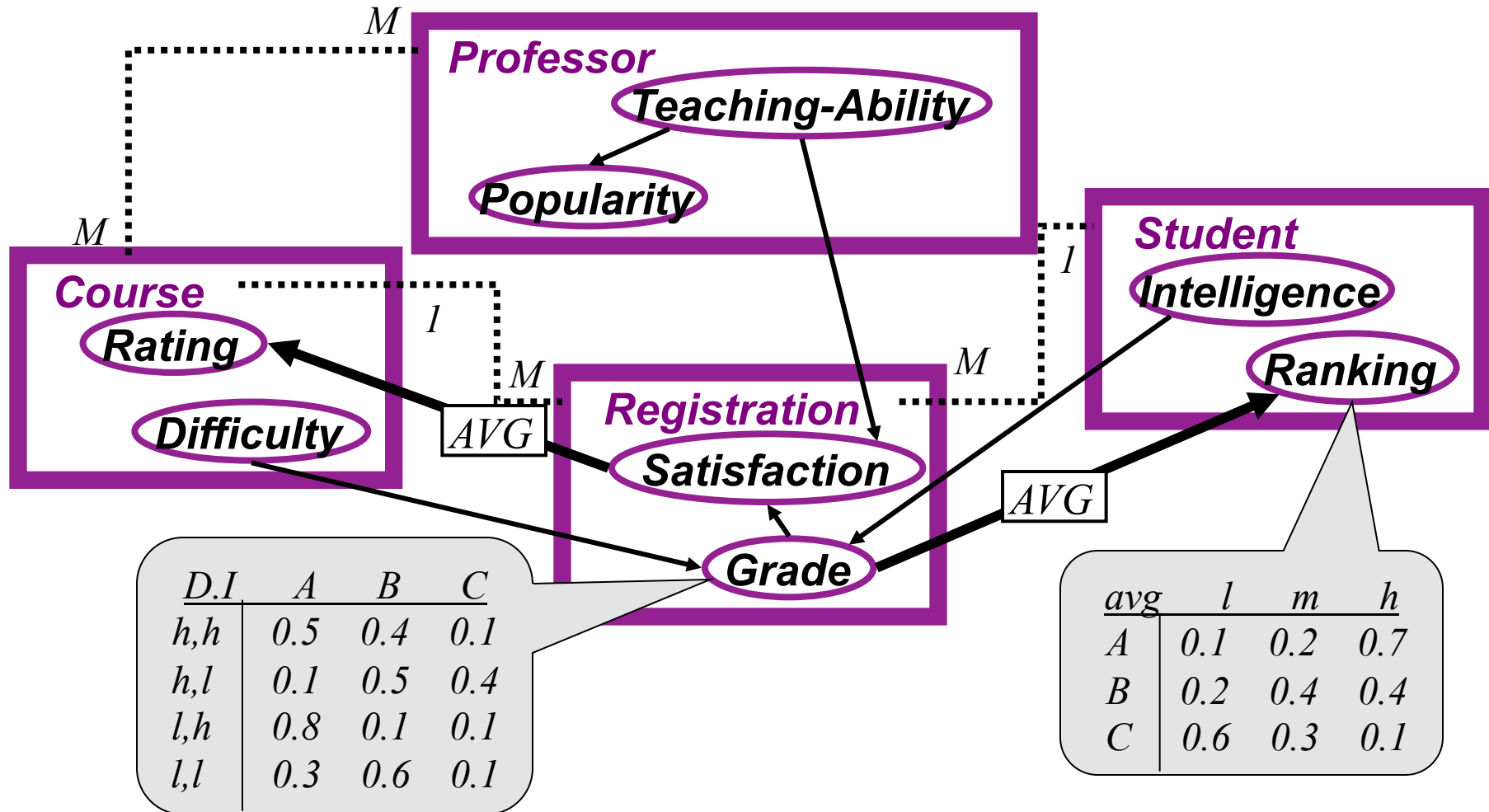
PRM Dependency Structure



Parameters of PRMs

- A PRM contains a *conditional probability distribution* (CPD) $P(X.A | Pa(X.A))$ for each attribute $X.A$ of each class
- More precisely, let \mathbf{U} be the set of parents of $X.A$. For each tuple of values $\mathbf{u} \in V(\mathbf{U})$, the CPD specifies a distribution $P(X.A | \mathbf{u})$ over $V(X.A)$. The parameters in all of these CPDs comprise θ_S

CPDs in PRMs



Parameters of PRMs Continued

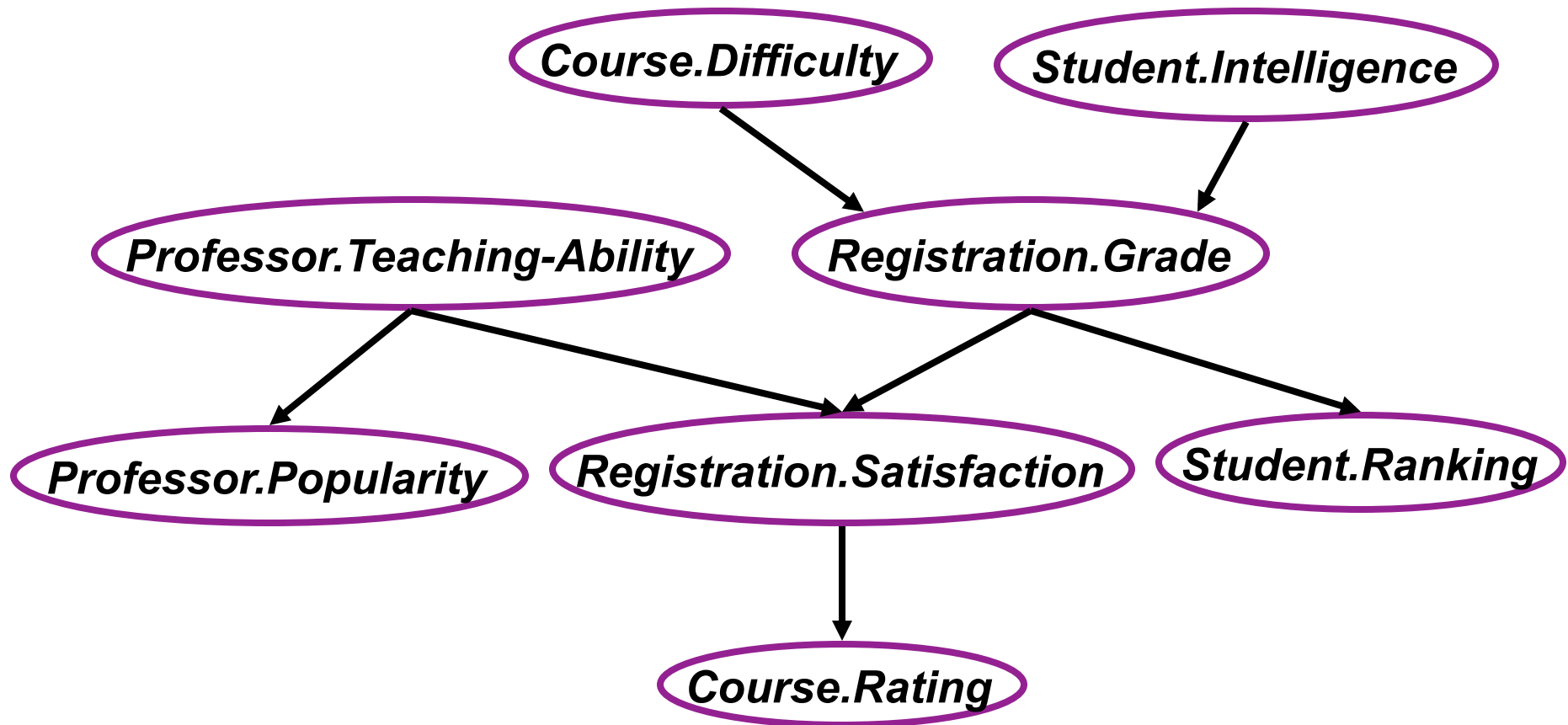
- Given a skeleton structure for our schema, we want to use these local probability models to define a probability distribution over all completions of the skeleton
- Note that the objects and relations between objects in a skeleton are always specified by σ , hence we are disallowing uncertainty over the relational structure of the model

Parameters of PRMs Continued

- To define a coherent probabilistic model, we must ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value
- A dependency structure \mathcal{S} is acyclic relative to a skeleton σ if the directed graph over all the parents of the variables $x.A$ is acyclic
- If \mathcal{S} is acyclic relative to σ , then the following defines a distribution over completions I of σ : $P(I | \sigma, \mathcal{S}, \theta_{\mathcal{S}}) =$

$$\prod_{X_i} \prod_{A \in A(X_i)} \prod_{x \in \mathcal{O}^{\sigma}(X_i)} P(I_{x.a} | I_{Pa(x.a)})$$

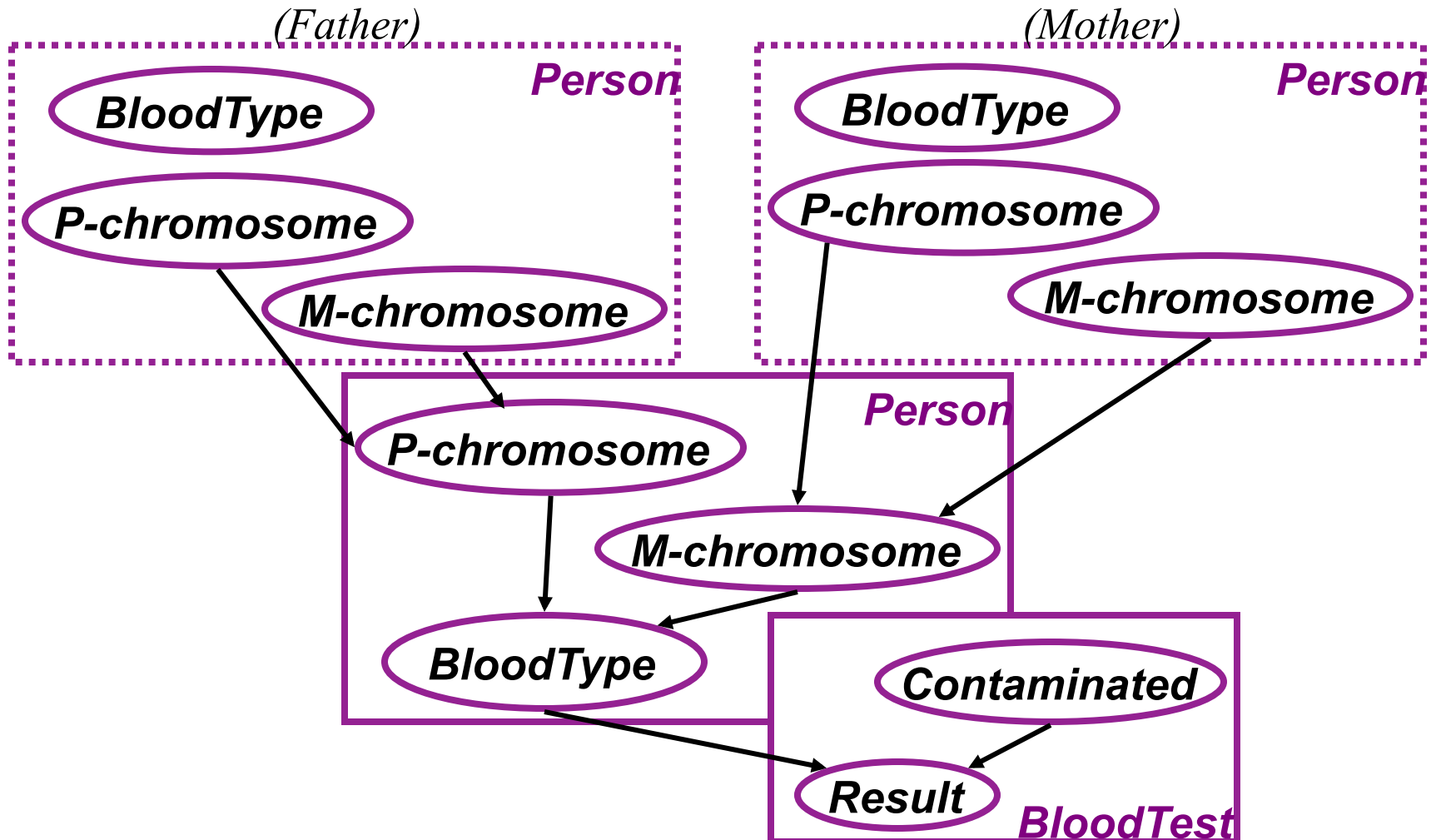
Class Dependency Graph for the University Domain



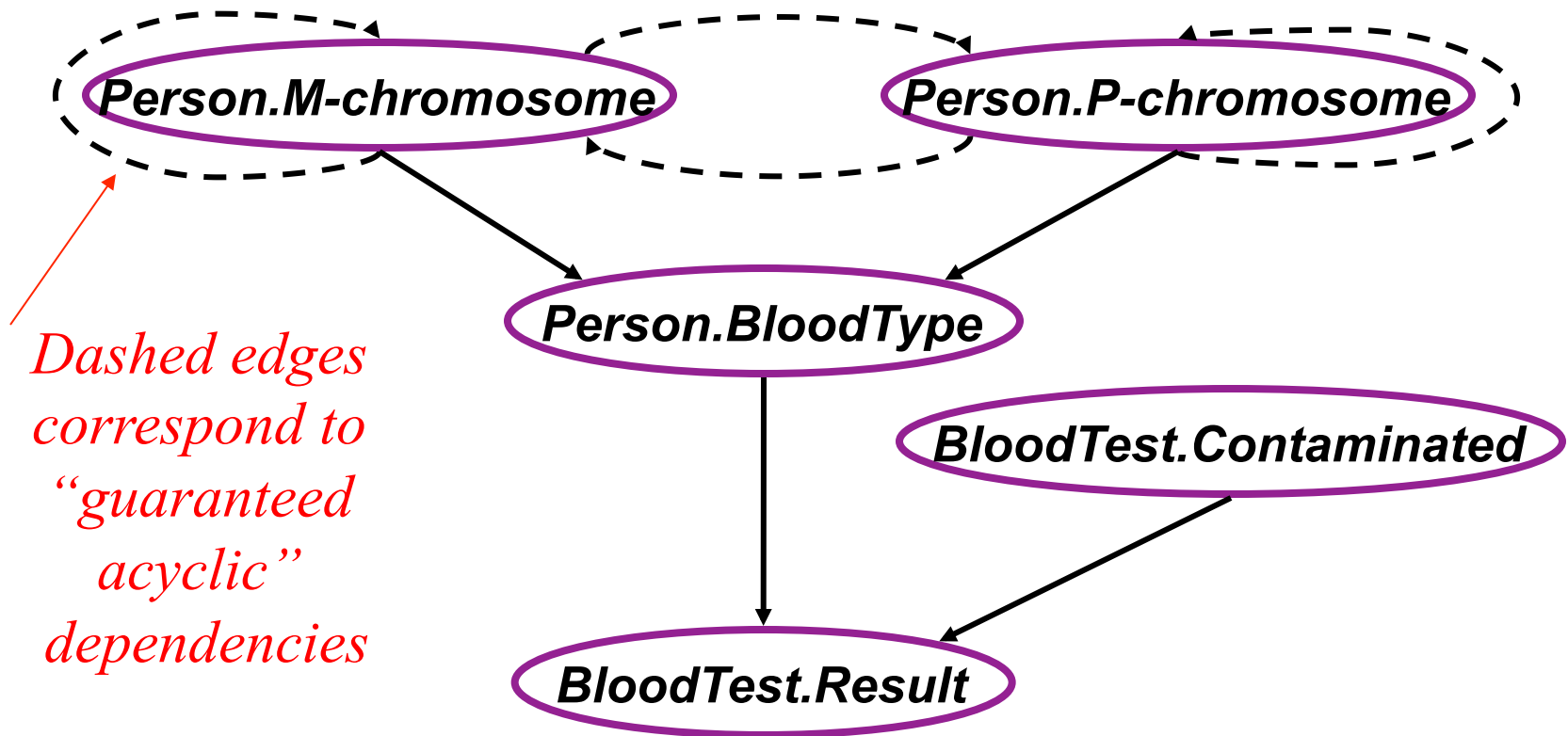
Ensuring Acyclic Dependencies

- In general, however, a cycle in the class dependency graph does not imply that all skeletons induce cyclic dependencies
- A model may appear to be cyclic at the class level, however, this cyclicity is always resolved at the level of individual objects
- The ability to guarantee that the cyclicity is resolved relies on some prior knowledge about the domain. The user can specify that certain slots are *guaranteed acyclic*

PRM for the Genetics Domain



Dependency Graph for Genetics Domain



Learning PRMs: Parameter Estimation

- Assume that the qualitative dependency structure \mathcal{S} of the PRM is known
- The parameters are estimated using the *likelihood function* which gives an estimate of the probability of the data given the model
- The likelihood function used is the same as that for Bayesian network parameter estimation. The only difference is that parameters for different nodes in the network – those corresponding to the $x.A$ for different objects x from the same class – are forced to be identical

Learning PRMs: Parameter Estimation

- Our goal is to find the parameter setting θ_S that maximizes the likelihood $L(\theta_S | I, \sigma, S)$ for a given I , σ and S : $L(\theta_S | I, \sigma, S) = P(I | \sigma, S, \theta_S)$. Working with the logarithm of this function:

$$l(\theta_S | I, \sigma, S) = \log P(I | \sigma, S, \theta_S) = \sum_{X_i} \sum_{A \in A(X_i)} \left[\sum_{x \in O^o(X_i)} \log P(I_{x.A} | I_{Pa(x.A)}) \right]$$

- This estimation is simplified by the decomposition of log-likelihood function into a summation of terms corresponding to the various attributes of the different classes. Each of the terms in the square brackets can be maximized independently of the rest
- Parameter priors can also be incorporated

Learning PRMs: Structure Learning

- We now move to the more challenging problem of learning a dependency structure automatically
- There are three important issues that need to be addressed: hypothesis space, scoring function, and search algorithm
- Our hypothesis specifies a set of parents for each attribute $X.A$. Note that this hypothesis space is infinite. Our hypothesis space is restricted by ensuring that the structure we are learning will generate a consistent probability model for any skeleton we are likely to see

Learning PRMs: Structure Learning Continued

- The second key component is the ability to evaluate different structures in order to pick one that fits the data well. Bayesian *model selection* methods were adapted
- Bayesian model selection utilizes a probabilistic scoring function. It ascribes a prior probability distribution over any aspect of the model about which we are uncertain
- The *Bayesian score* of a structure \mathcal{S} is defined as the *posterior* probability of the structure given the data I

Learning PRMs: Structure Learning Continued

- Using Bayes rule: $P(S|I,\sigma) \propto P(I|S,\sigma) P(S|\sigma)$
- It turns out that marginal likelihood is a crucial component, which has the effect of penalizing models with a large number of parameters. Thus this score automatically balances the complexity of the structure with its fit to the data
- Now we need only provide an algorithm for finding a high-scoring hypotheses in our space

Learning PRMs: Structure Learning Continued

- The simplest heuristic search algorithm is greedy hill-climbing search, using the scoring function as a metric. Maintain the current candidate structure and iteratively improve it
- Local maxima can be dealt with using random restarts, i.e., when a local maximum is reached, we take a number of random steps, and then continue the greedy hill-climbing process

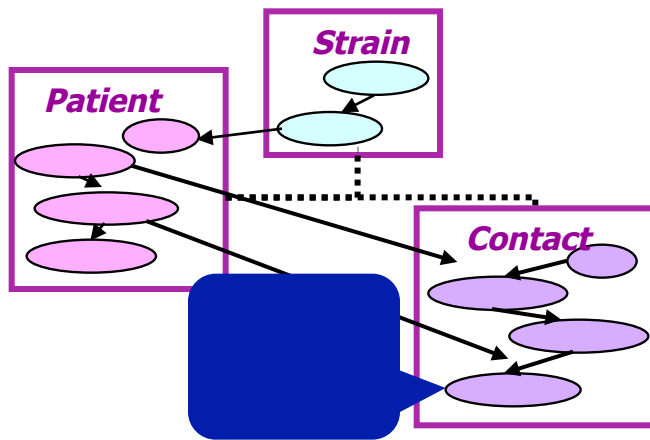
Learning PRMs: Structure Learning Continued

- The problems with this simple approach is that there are infinitely many possible structures, and it is very costly in computational operations
- A heuristic search algorithm addresses these issues. At a high level, the algorithm proceeds in phases

Learning PRMs: Structure Learning Continued

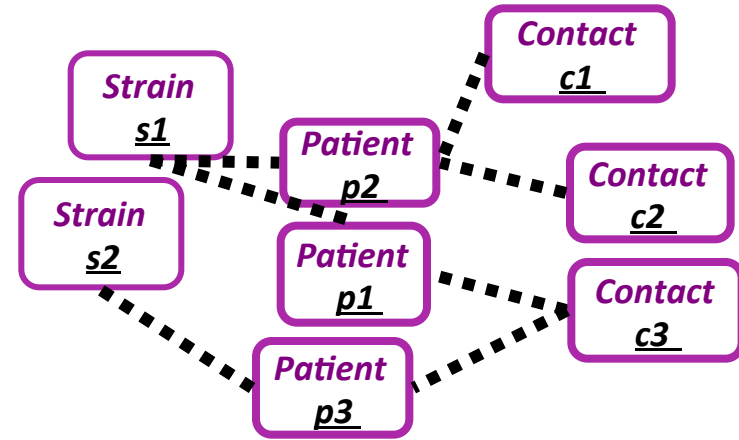
- At each phase k , we have a set of potential parents $Pot_k(X.A)$ for each attribute $X.A$
- Then apply a standard structure search restricted to the space of structures in which the parents of each $X.A$ are in $Pot_k(X.A)$. The phased search is structured so that it first explores dependencies within objects, then between objects that are directly related, then between objects that are two links apart, etc

PRM with AU Semantics



PRM

+



relational skeleton σ

=

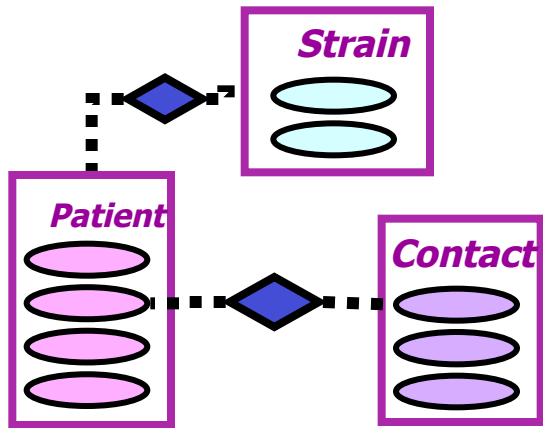
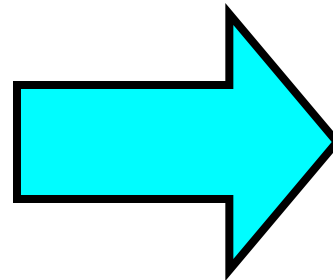
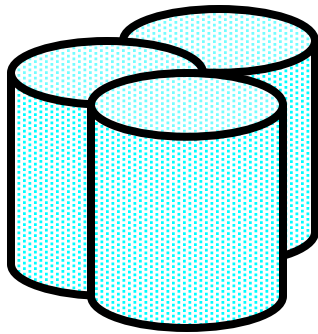
probability distribution over completions \mathcal{I} :

$$P(\mathcal{I} \mid \sigma, \mathcal{S}, \Theta) = \prod_{x \in \sigma} \prod_{x.A} P(x.A \mid \text{parents}_{\mathcal{S}, \sigma}(x.A))$$

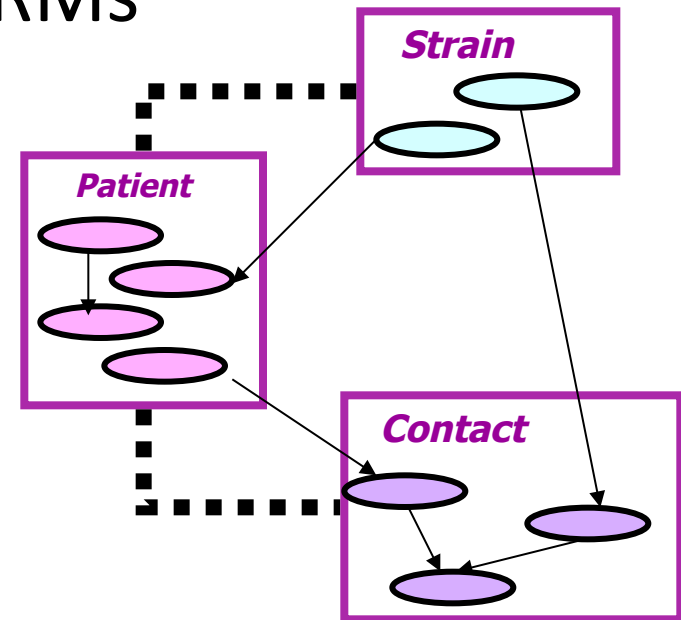
↑ ↑
Objects *Attributes*

Learning PRMs

Database



Relational Schema



PRM

- Parameter estimation
- Structure selection

Parameter Estimation in PRMs

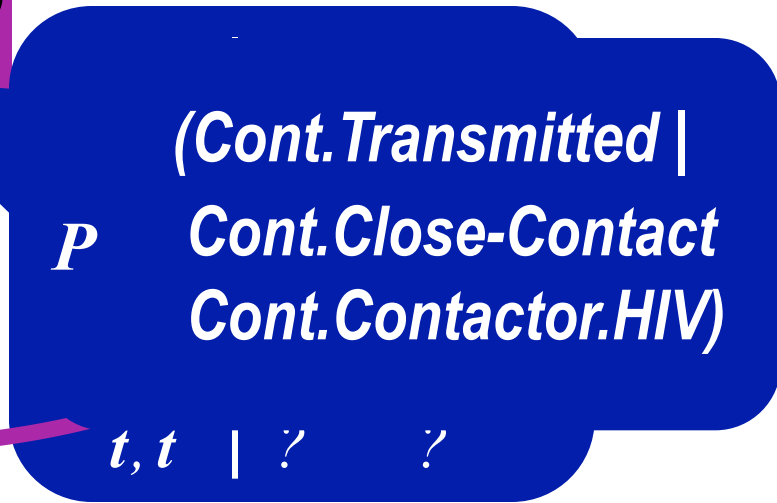
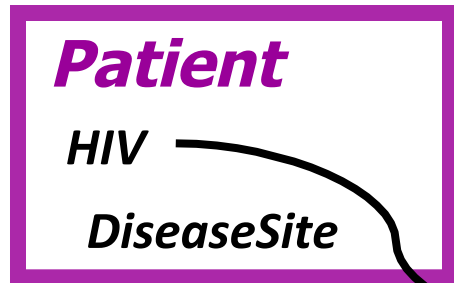
- Assume known dependency structure S
- Goal: estimate PRM parameters θ
 - entries in local probability models, $\theta_{x.A|parents(x.A)}$
- θ is good if it is likely to generate the observed data, instance I .

$$l(\theta : I, S) = \log P(I | S, \theta)$$

- MLE Principle: Choose θ^* so as to maximize l

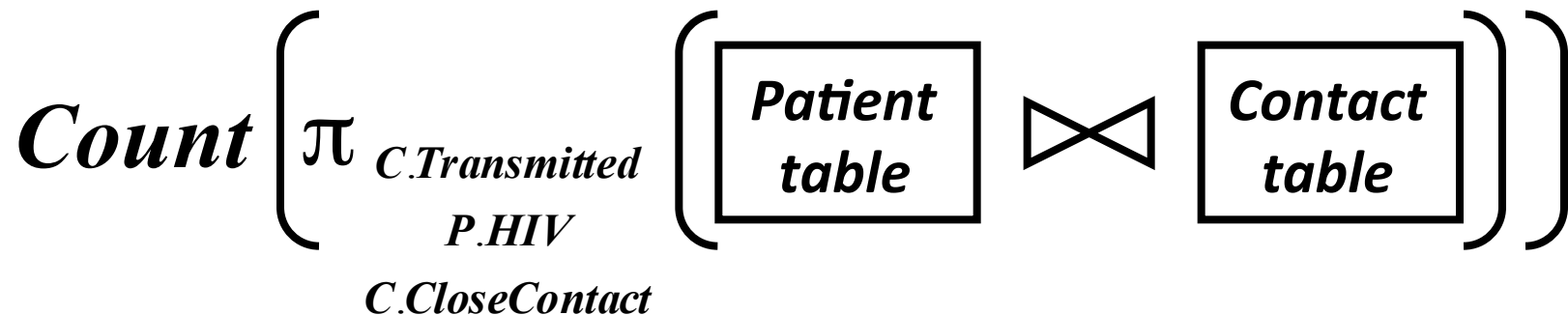
As in Bayesian network learning, decomposition plays a crucial role: separate terms for different $X.A$

ML Parameter Estimation



$$\theta^* = \frac{N(C.T=f, P.H=f, C.C=t)}{N(P.H=f, C.C=t)}$$

Query for counts:

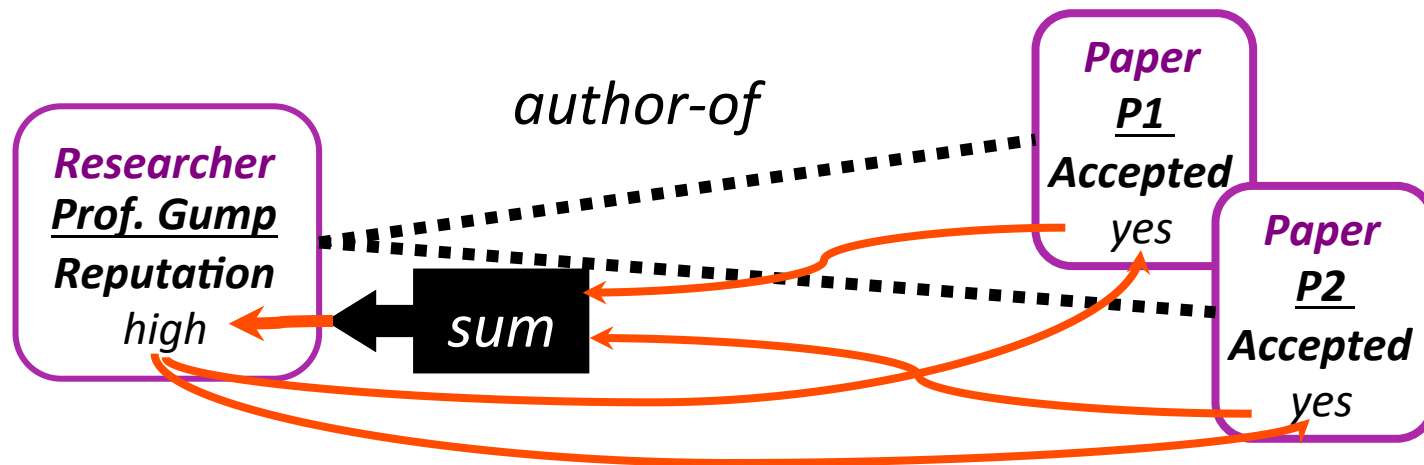


Structure Selection

- **Idea:**
 - define scoring function
 - do local search over legal structures
- **Key Components:**
 - “legal” models
 - scoring models
 - searching model space

Legal Models

- PRM defines a coherent probability model over a skeleton σ if the dependencies between object attributes is acyclic



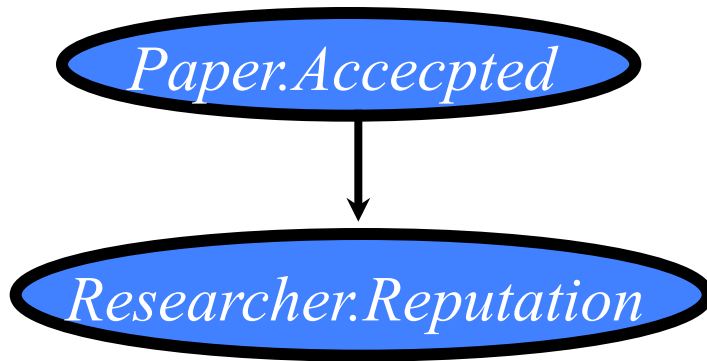
How do we guarantee that a PRM is acyclic for **every** skeleton?

Attribute Stratification

PRM dependency structure S



Dependency graph



if *Researcher.Reputation* depends directly on *Paper.Accepted*

dependency graph acyclic \Rightarrow acyclic for any σ

Scoring Models

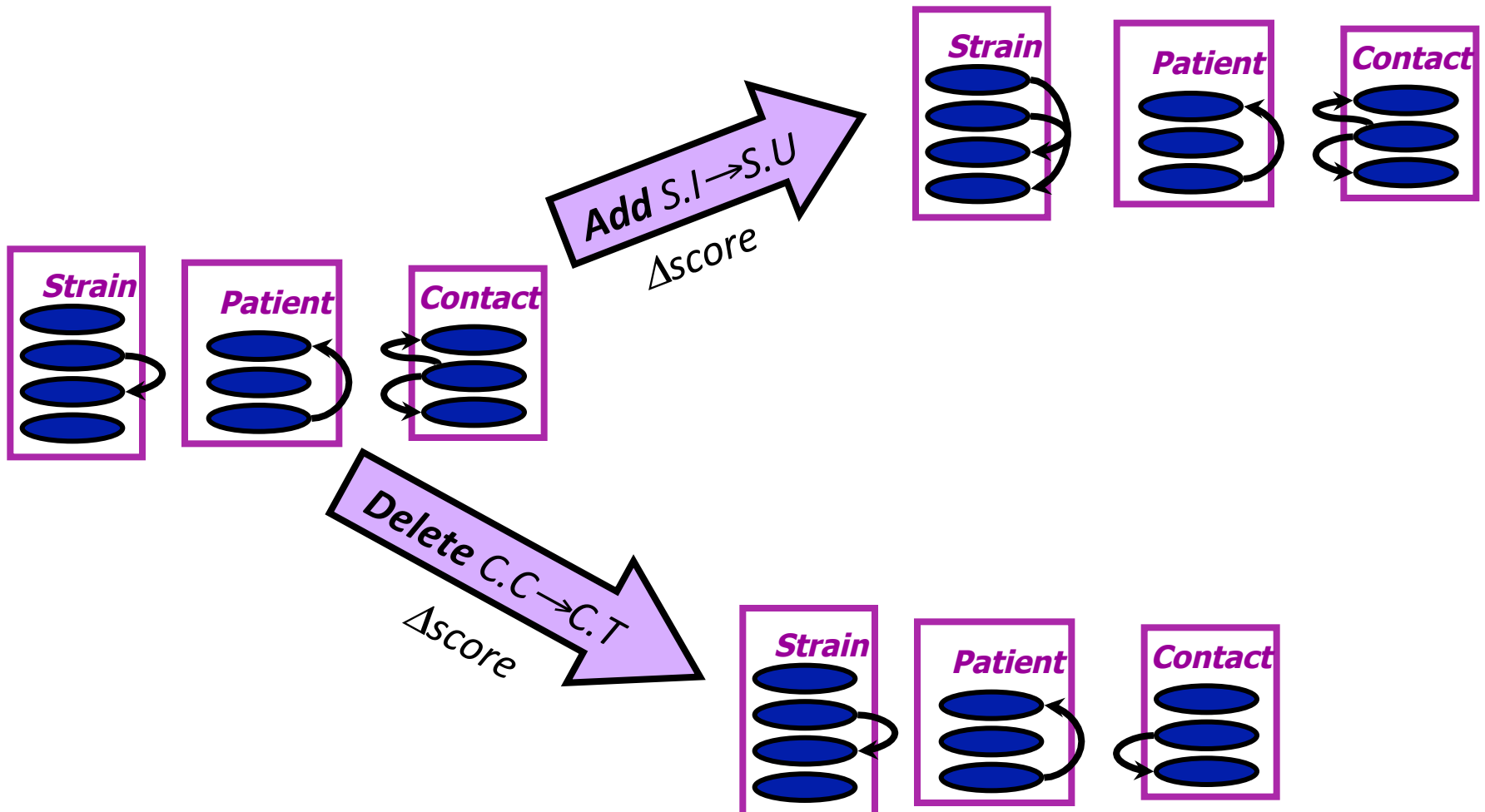
- Bayesian approach:

$$\text{Score}(S : \mathbf{I}) = \log P(S | \mathbf{I}) \propto \log \left[\overbrace{P(\mathbf{I} | S)}^{\text{marginal likelihood}} \overbrace{P(S)}^{\text{prior}} \right]$$

- Standard approach to scoring models; used in Bayesian network learning

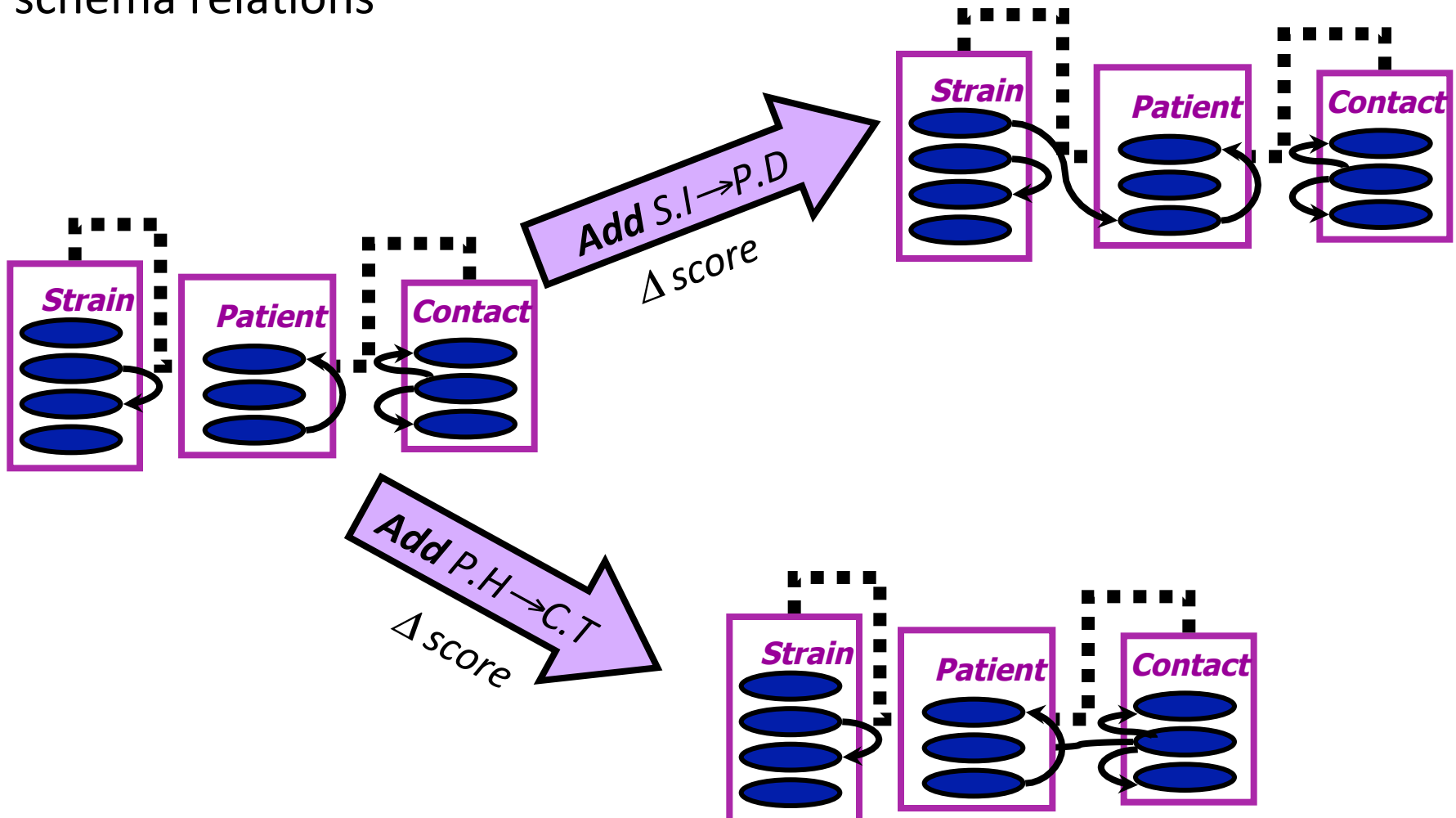
Search over Models

Phase 0: consider only dependencies within a class



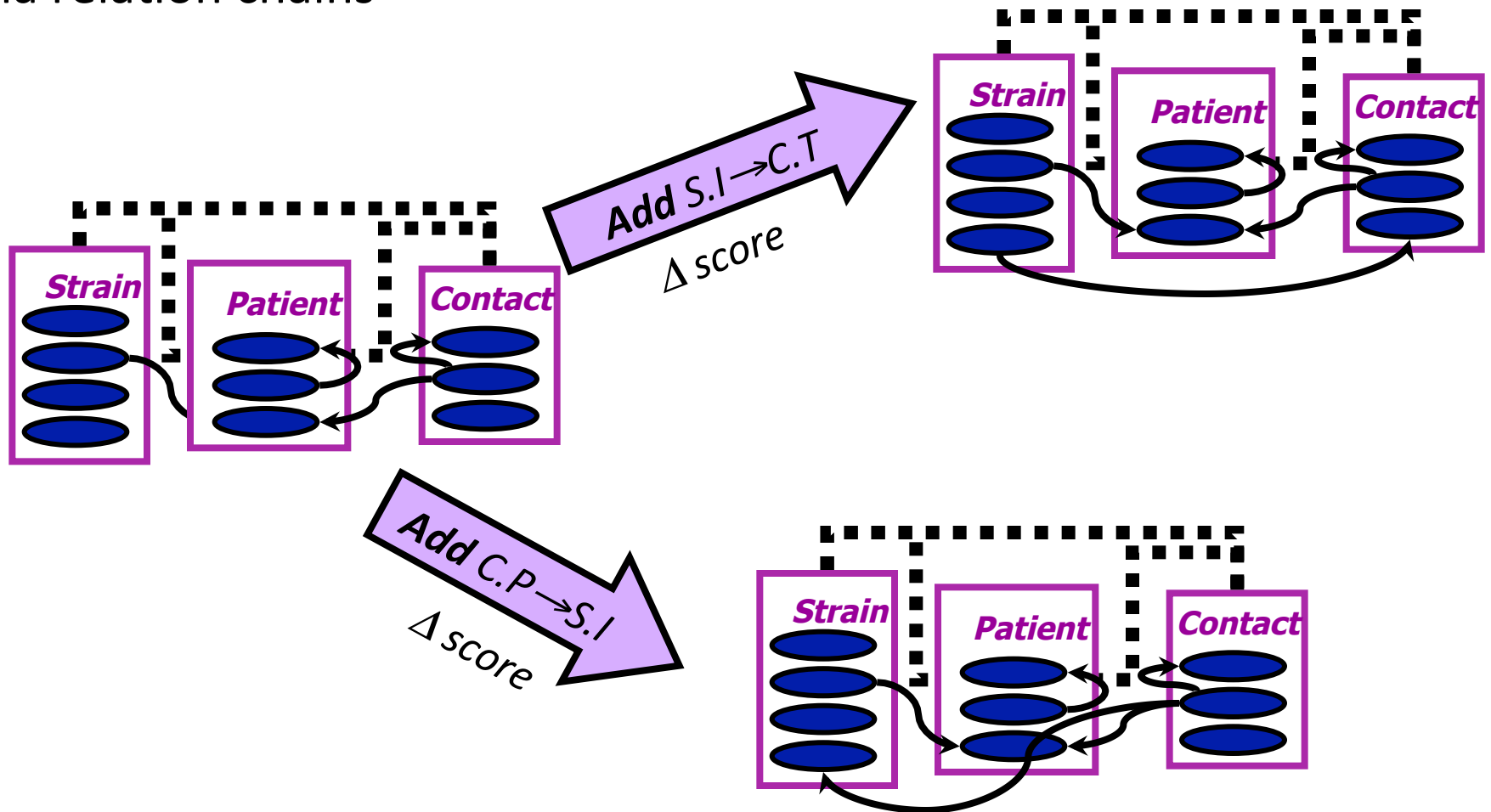
Phased Structure Search

Phase 1: consider dependencies from “neighboring classes, via schema relations

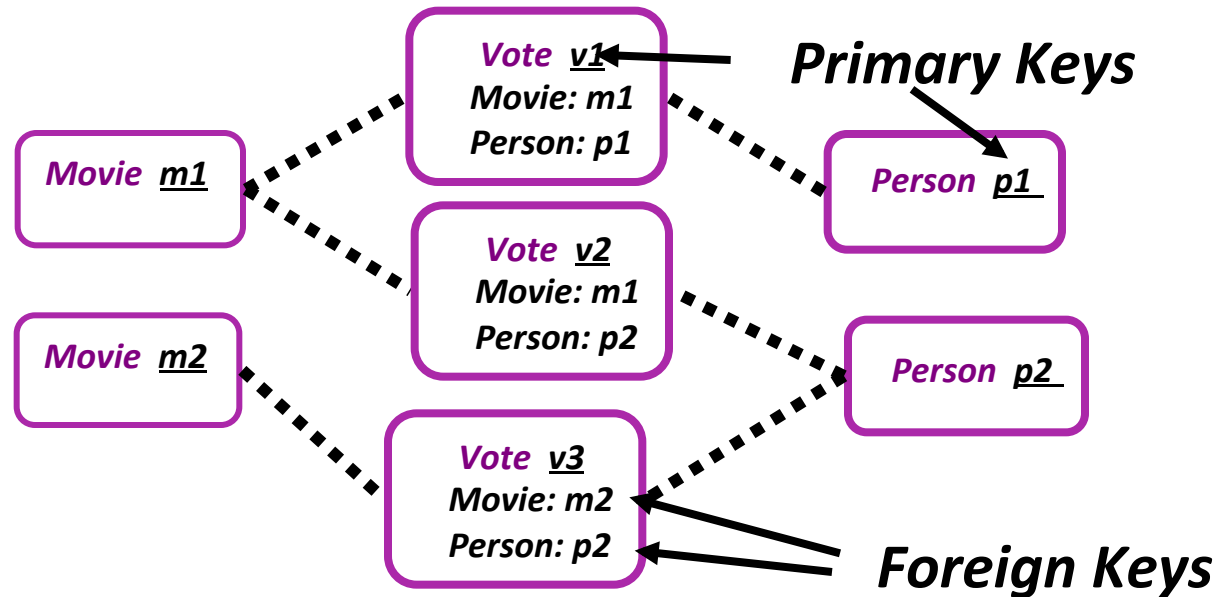


Phased Structure Search

Phase 2: consider dependencies from “further classes,
 via relation chains



So far we have considered PRM with Attribute Uncertainty

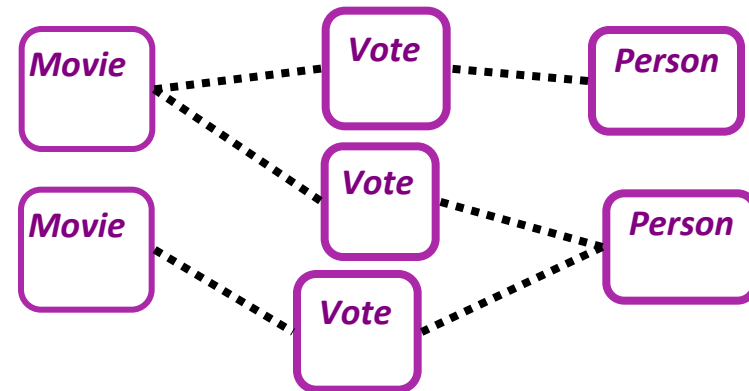
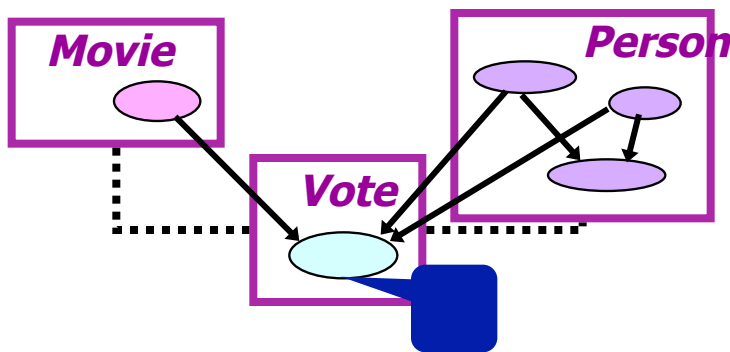


Fixed relational skeleton σ :

- set of objects in each class
- relations between them

Uncertainty over assignment of values to attributes

PRM w/ AU Semantics



PRM + *relational skeleton σ* =

Ground BN defining distribution over complete instantiations of attributes \mathcal{I} :

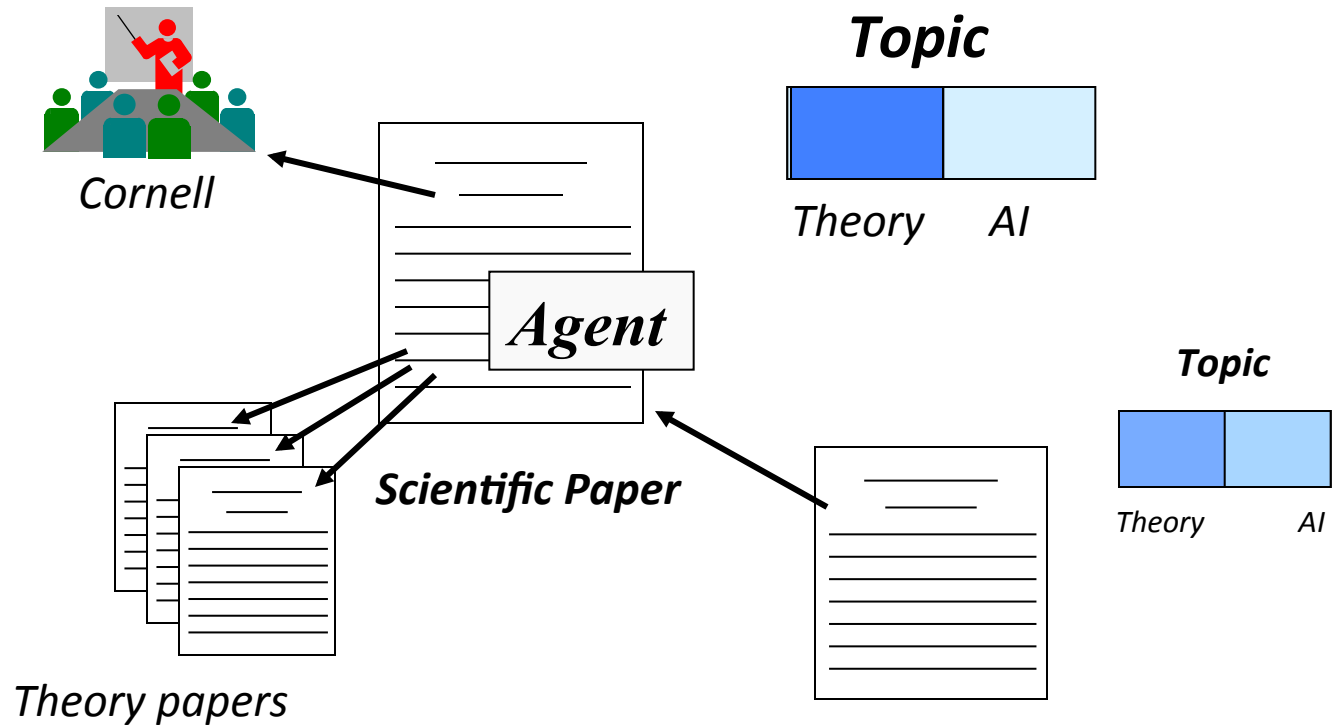
$$P(\mathcal{I} \mid \sigma, \mathcal{S}, \Theta) = \prod_{x \in \sigma} \prod_{x.A} P(x.A \mid \text{parents}_{\mathcal{S}, \sigma}(x.A))$$

↑ **Objects** ↑ **Attributes**

Problem

- **Relational structure** provides useful information for probability estimation and prediction
- PRM with attribute uncertainty applicable only in domains where we have full knowledge of the relational structure
- Need probabilistic models of relational structure that capture **structural uncertainty**
 - Reference uncertainty
 - Existence uncertainty

PRM With Structural Uncertainty



- Need probabilistic models of relational structure that capture **structural uncertainty**
 - Reference uncertainty
 - Existence uncertainty

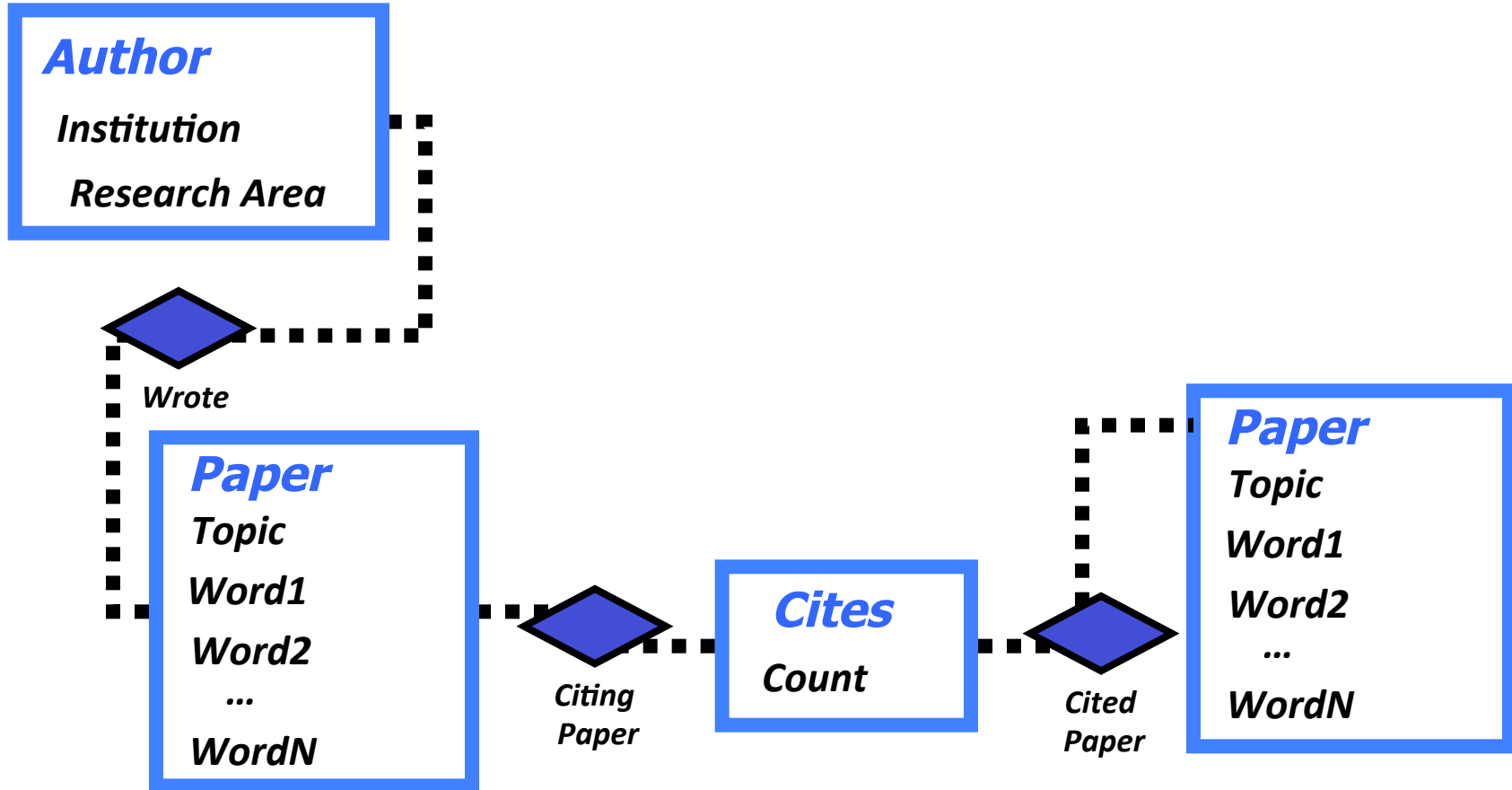
PRMs with Structural Uncertainty

- Applicable in cases where we do not have full knowledge of relational structure
- Incorporating uncertainty over relational structure into probabilistic model can improve predictive accuracy

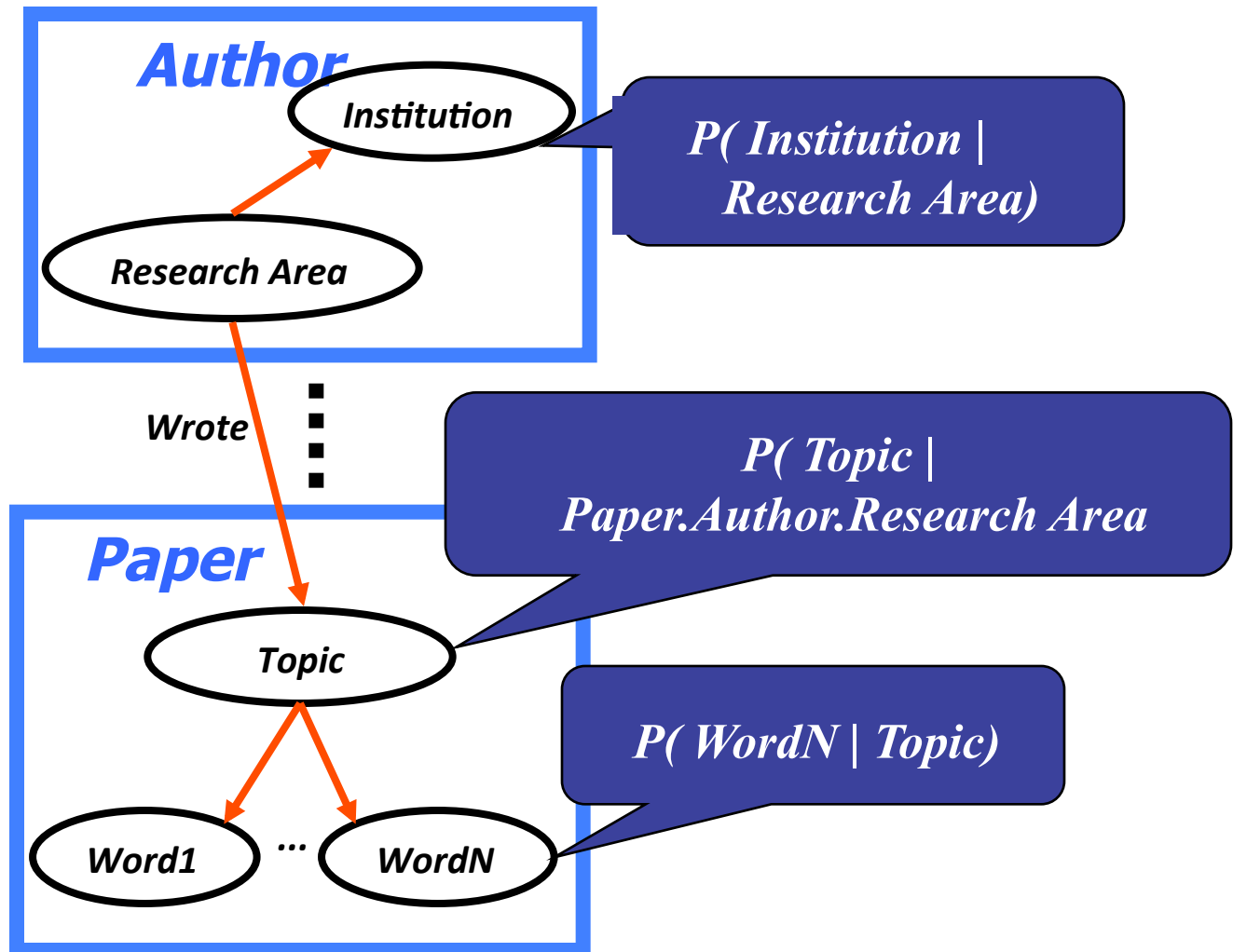
Two cases:

- Reference uncertainty
- Existence uncertainty
- Different probabilistic models
- Varying amount of background knowledge required for each

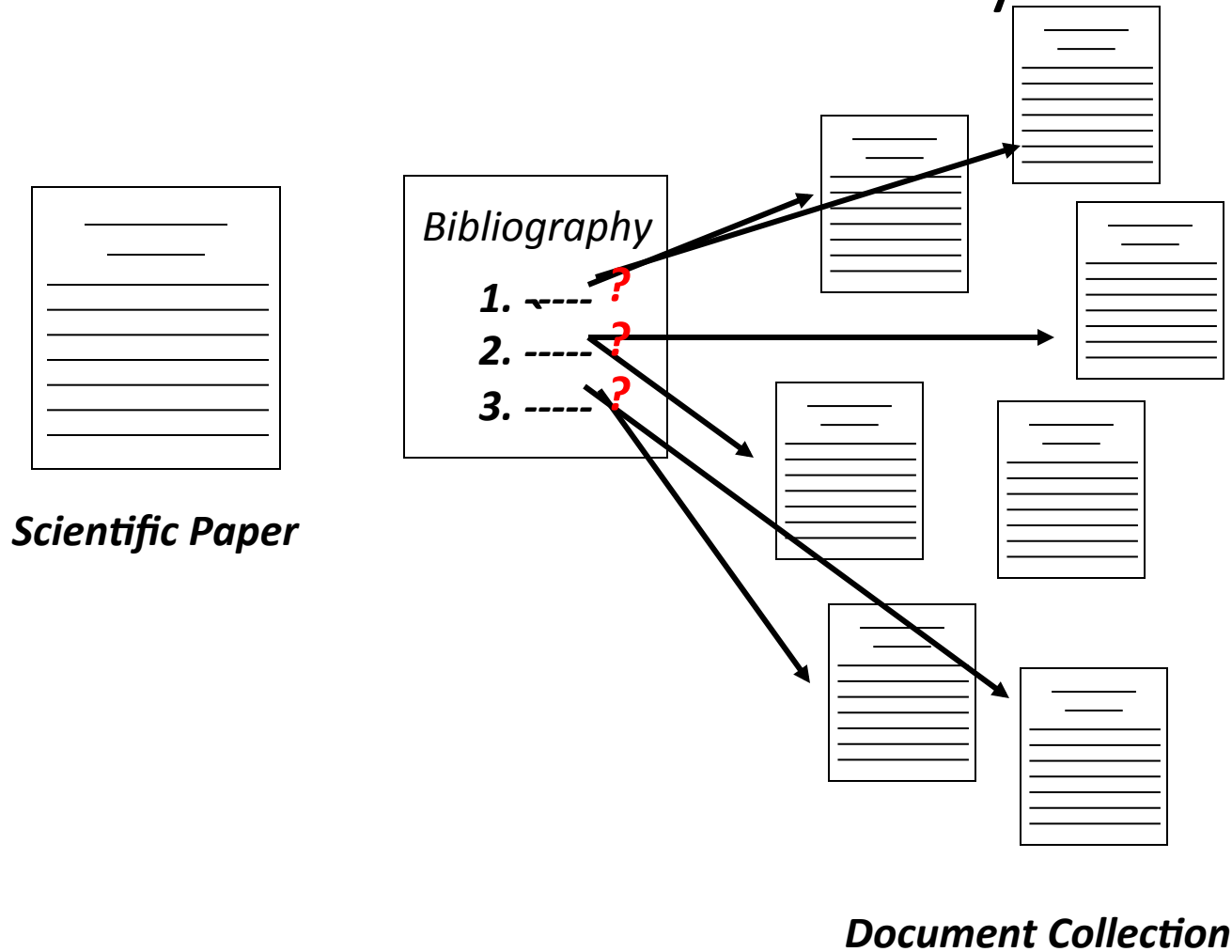
Citation Relational Schema



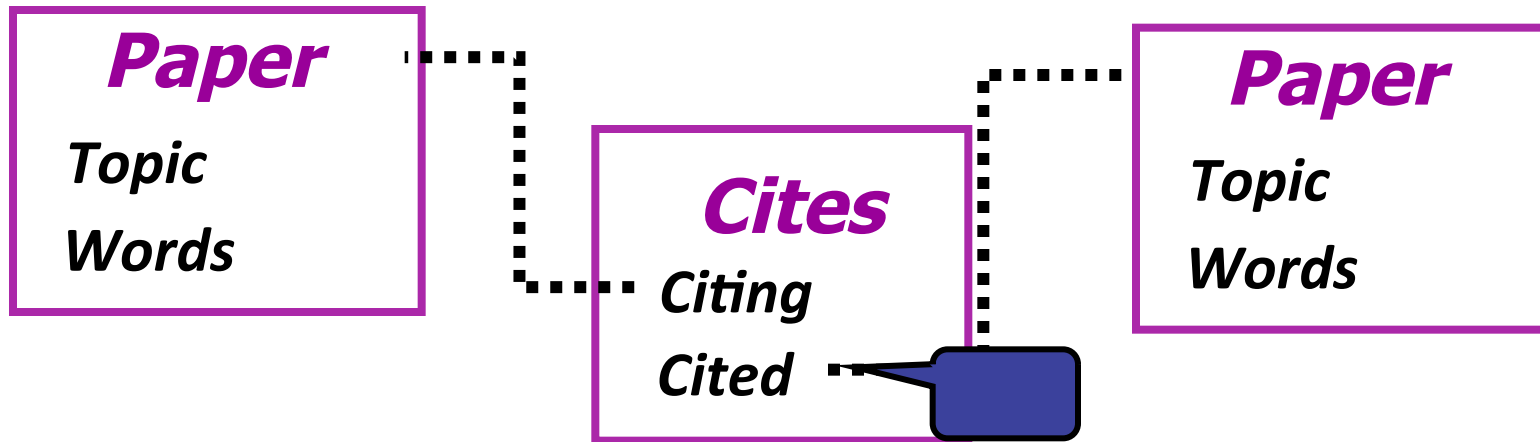
Attribute Uncertainty



Reference Uncertainty

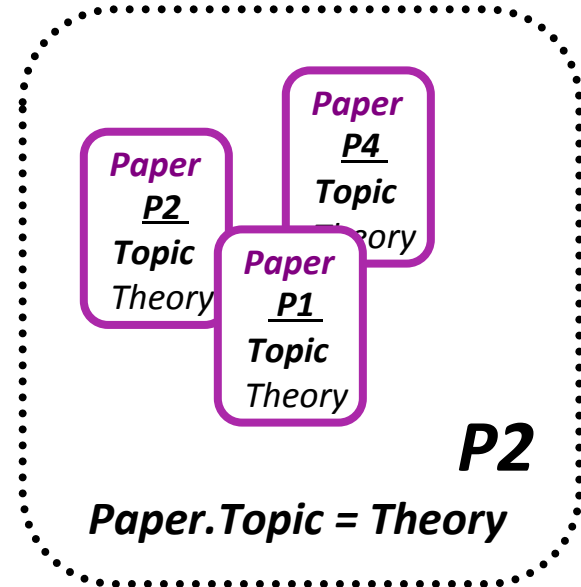
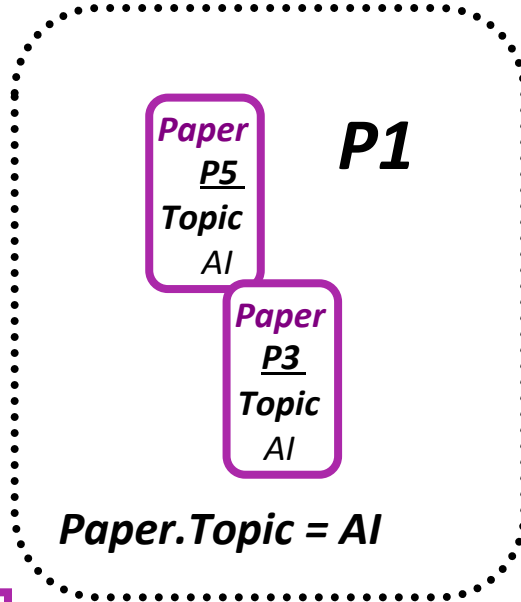
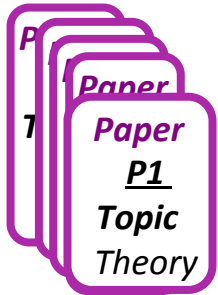


PRM w/ Reference Uncertainty



Dependency model for foreign keys

Reference Uncertainty Example

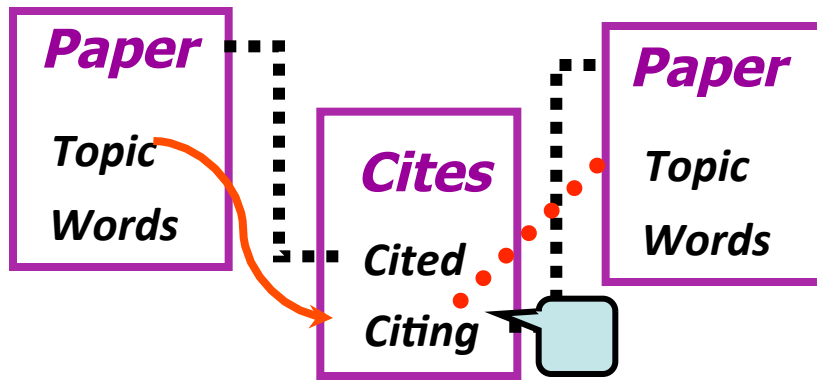


Paper
Topic
Words

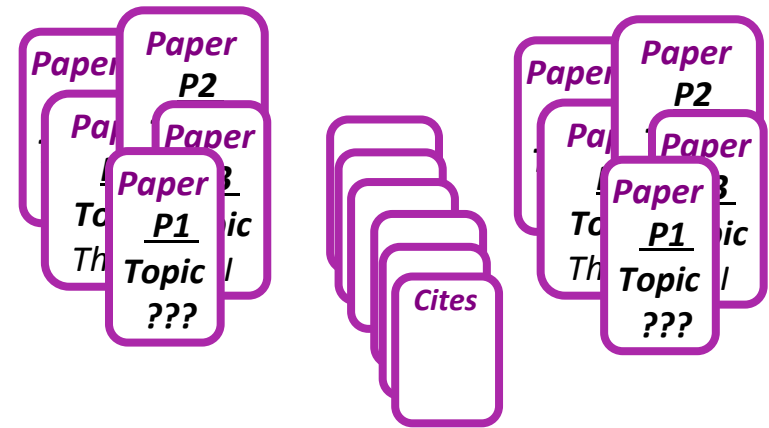
Cites
Citing
Cited

Topic	P1	P2
Theory	0.1	0.9
AI	0.99	0.01

PRMs w/ RU Semantics



PRM RU



entity skeleton σ

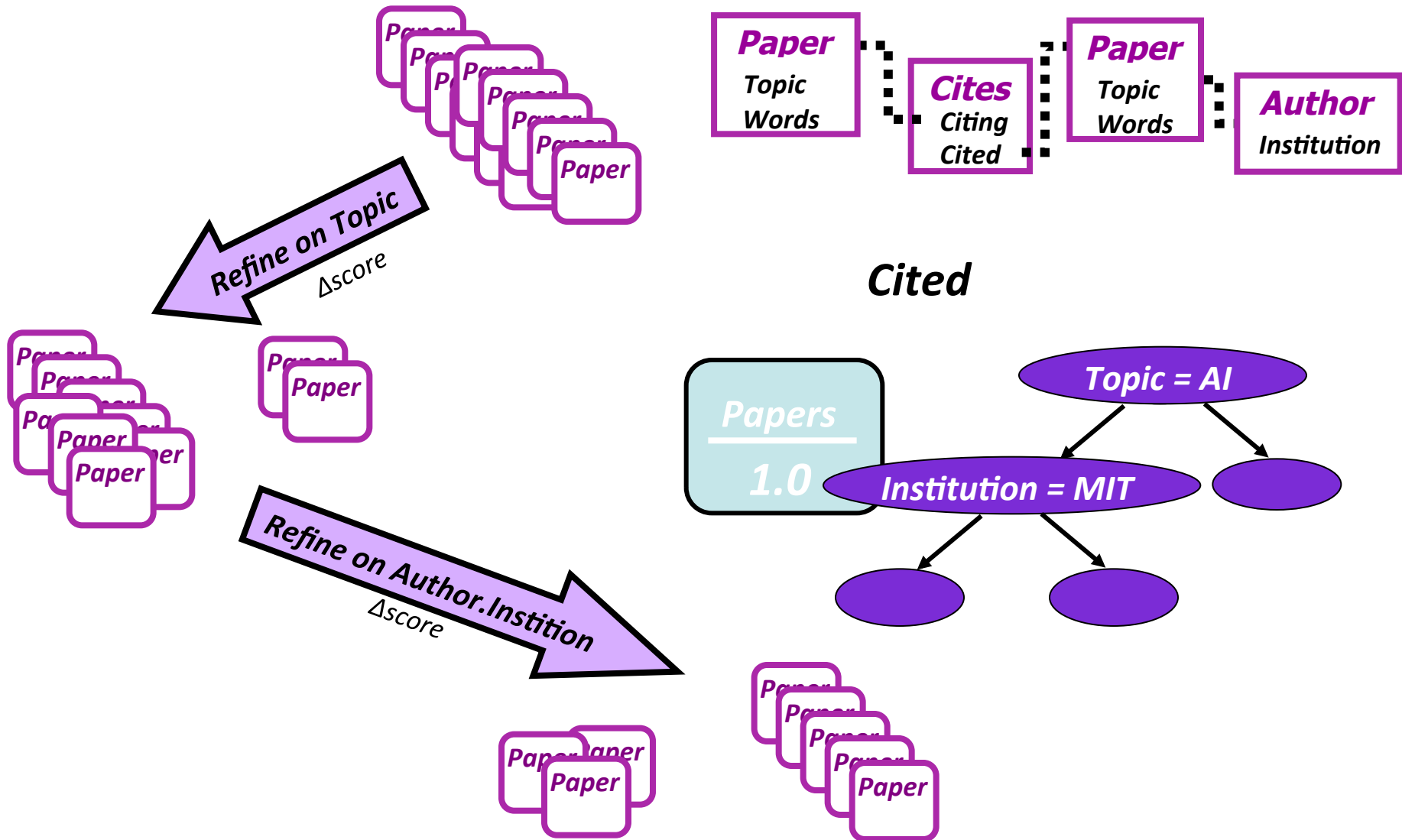
PRM-RU + **entity skeleton σ**

\Rightarrow probability distribution over full instantiations I

Learning PRMs with Reference Uncertainty

- **Idea:**
 - define scoring function
 - do phased local search over legal structures
- **Key Components:**
 - legal models
 - **Model new dependencies**
 - scoring models
 - **Unchanged**
 - searching model space
 - **New operators**

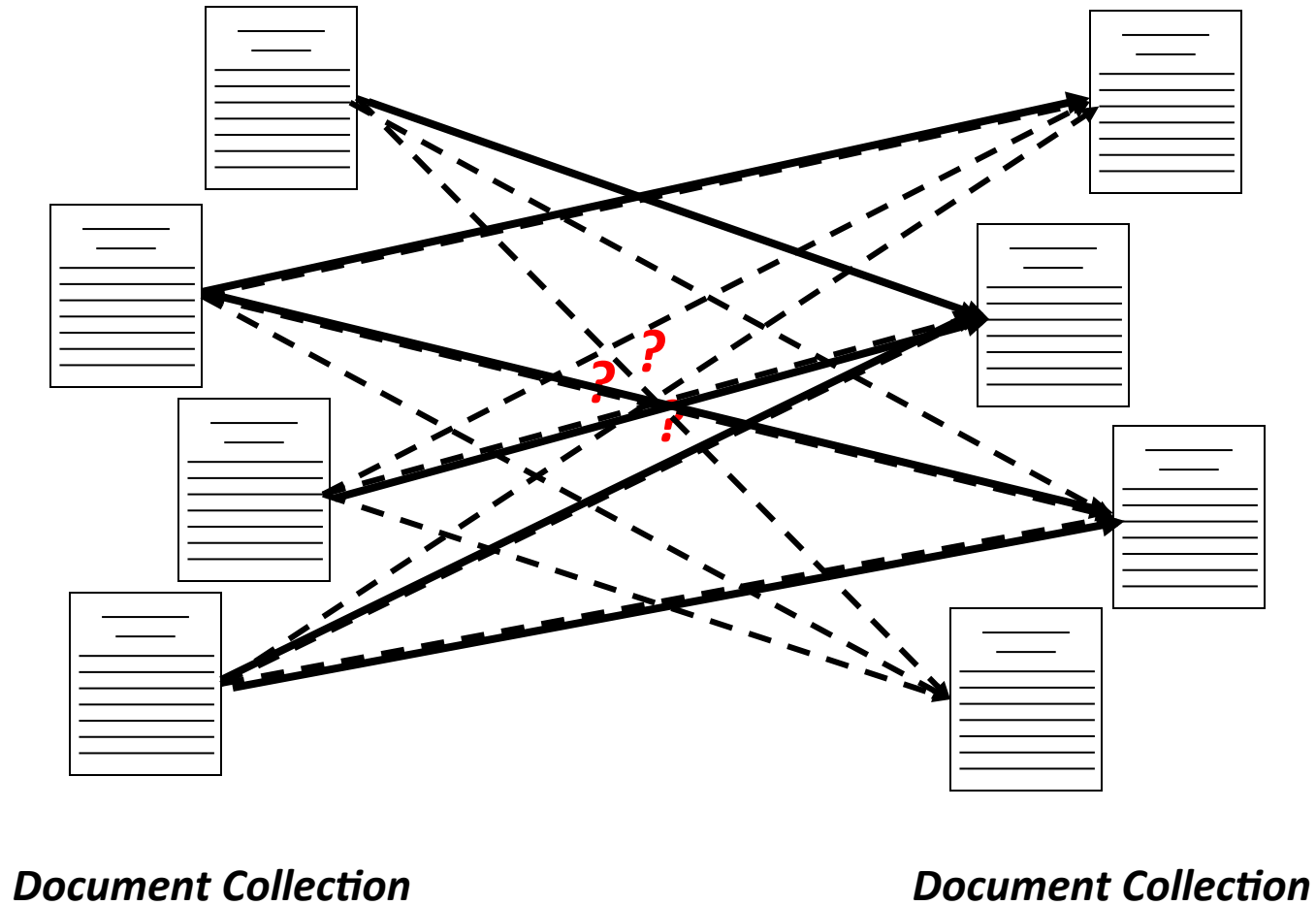
Structure Search: New Operators



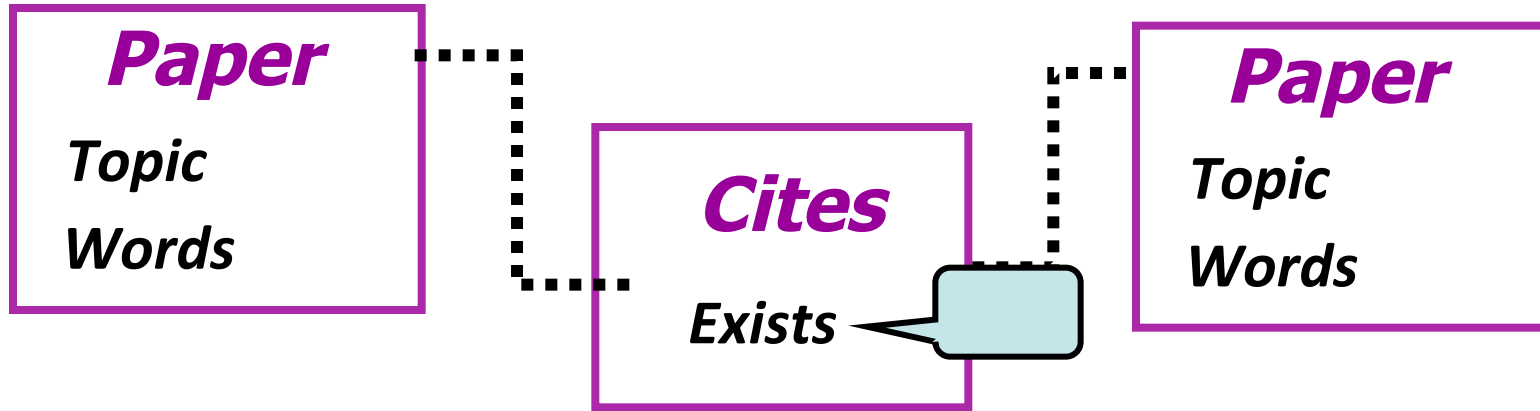
PRMs w/ RU Summary

- Define semantics for uncertainty over foreign-key values
- Search now includes operators **Refine** and **Abstract** for constructing foreign-key dependency model
- Provides one simple mechanism for link uncertainty

Existence Uncertainty

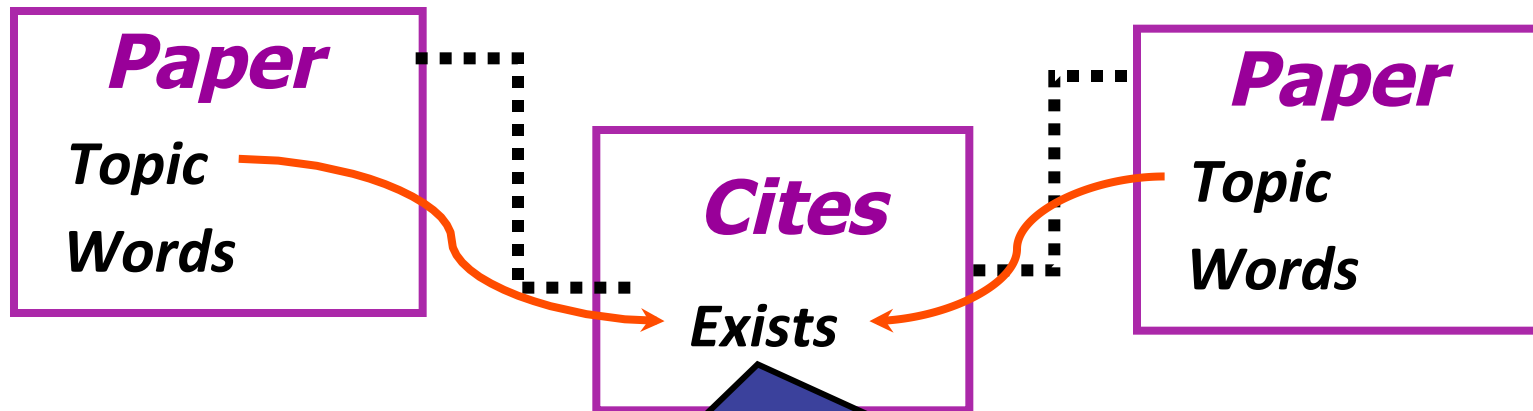


PRM with Existence Uncertainty



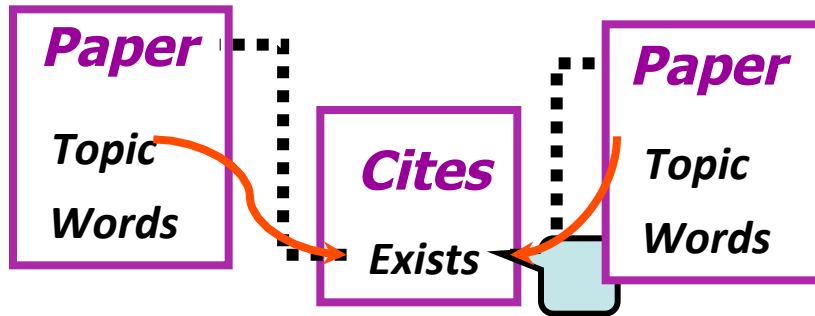
Dependency model for existence of relationship

Existence Uncertainty Example

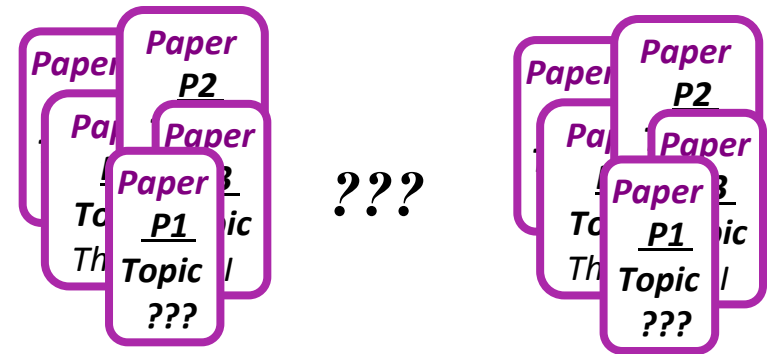


<i>Citer.Topic</i>	<i>Cited.Topic</i>	<i>False</i>	<i>True</i>
<i>Theory</i>	<i>Theory</i>	<i>0.995</i>	<i>0005</i>
<i>Theory</i>	<i>AI</i>	<i>0.999</i>	<i>0001</i>
<i>AI</i>	<i>Theory</i>	<i>0.997</i>	<i>0003</i>
<i>AI</i>	<i>AI</i>	<i>0.993</i>	<i>0008</i>

PRMs w/ EU Semantics



PRM EU



object skeleton σ

PRM-EU + object skeleton σ

\Rightarrow *probability distribution over full instantiations \mathcal{I}*

Learning PRMs **with Existence uncertainty**

- **Idea:** just like in PRMs w/ AU
 - define scoring function
 - do greedy local structure search

Structure Selection: PRM

- **Idea:**
 - define scoring function
 - do phased local search over legal structures
- **Key Components:**
 - legal models: model new dependencies
 - scoring models: unchanged
 - searching model space

Prediction Accuracy

	Cora	WebKB
baseline	75 ± 2.0	74 ± 2.5
RU Citing	81 ± 1.7	78 ± 2.3
RU Cited	79 ± 1.3	77 ± 1.5
EU	85 ± 0.09	82 ± 1.3

Inference in Unrolled BN

- Prediction requires inference in “unrolled” network
 - Infeasible for large networks
 - Use approximate inference for E-step
- Local message passing
 - Belief messages transferred between related instances
 - Induces a natural “influence” propagation behavior
 - Instances give information about related instances

Conclusions

- PRMs can represent distribution over attributes from multiple tables
- PRMs can capture link uncertainty
- PRMs allow inferences about individuals while taking into account relational structure