Learning Bayesian Networks

www.biostat.wisc.edu/~page/cs760/

Goals for the lecture

you should understand the following concepts

- the Bayesian network representation
- inference by enumeration
- variable elimination inference
- junction tree (clique tree) inference
- Markov chain Monte Carlo (MCMC)
- Gibbs sampling
- the parameter learning task for Bayes nets
- the structure learning task for Bayes nets
- maximum likelihood estimation
- Laplace estimates
- *m*-estimates

Goals for the lecture

you should understand the following concepts

- missing data in machine learning
  - hidden variables
  - missing at random
  - missing systematically
- the EM approach to imputing missing values in Bayes net parameter learning
- K-means clustering algorithm as another example of EM
- the Chow-Liu algorithm for structure search
- structure learning as search
- Kullback-Leibler divergence
- the Sparse Candidate algorithm
- the naïve Bayes classifier
- the Tree Augmented Network (TAN) algorithm

Bayesian network example

- Consider the following 5 binary random variables:

  B = a burglary occurs at your house

  E = an earthquake occurs at your house

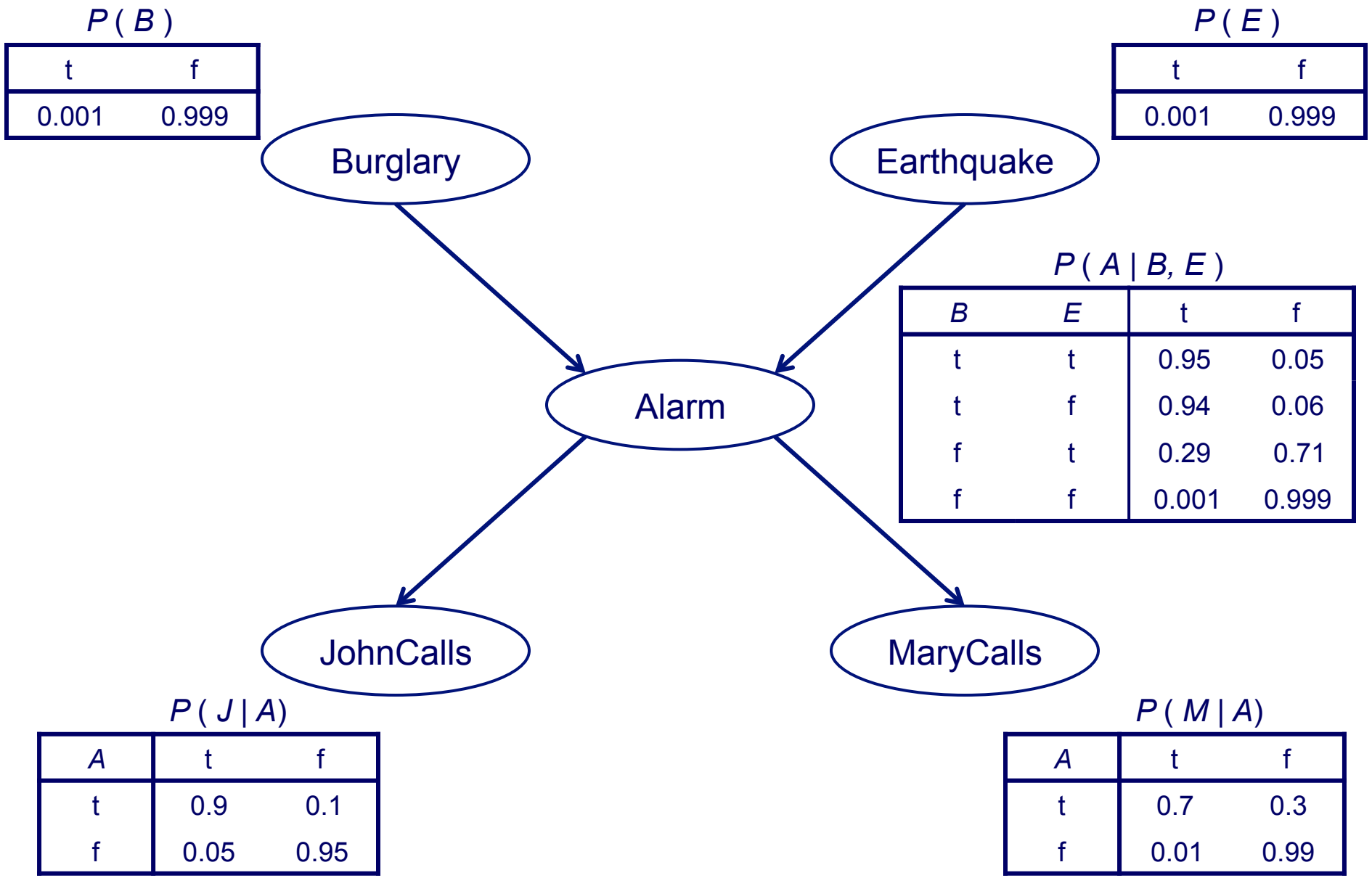
  A = the alarm goes off

  J = John calls to report the alarm

  M = Mary calls to report the alarm

- Suppose we want to answer queries like what is $P(B \mid M, J)$?

Bayesian network example



Bayesian networks

- a BN consists of a Directed Acyclic Graph (DAG) and a set of conditional probability distributions

- in the DAG
  - each node denotes random a variable
  - each edge from X to Y represents that X *directly influences* Y
  - formally: each variable X is independent of its non-descendants given its parents

- each node X has a *conditional probability distribution* (CPD) representing $P(X \mid Parents(X))$

Bayesian networks

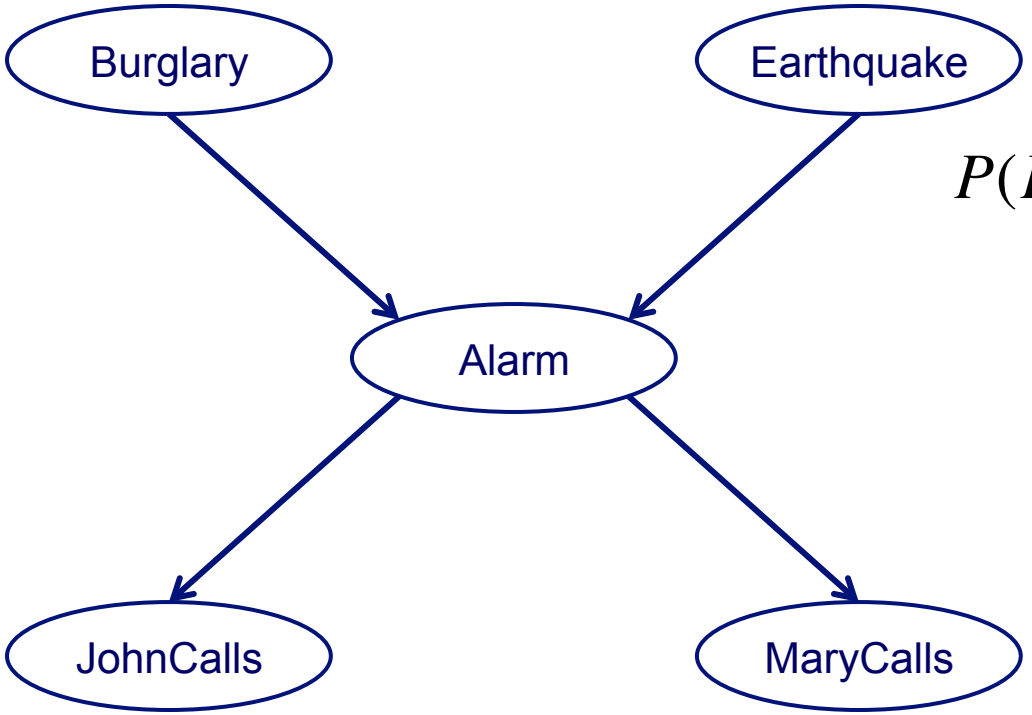
- using the chain rule, a joint probability distribution can be expressed as

$$P(X_1, \ldots, X_n) = \mathrm{P}(X_1)\prod_{i=2}^{n} P(X_i \mid X_1, \ldots, X_{i-1}))$$

- a BN provides a compact representation of a joint probability distribution

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid Parents(X_i))$$

Bayesian networks



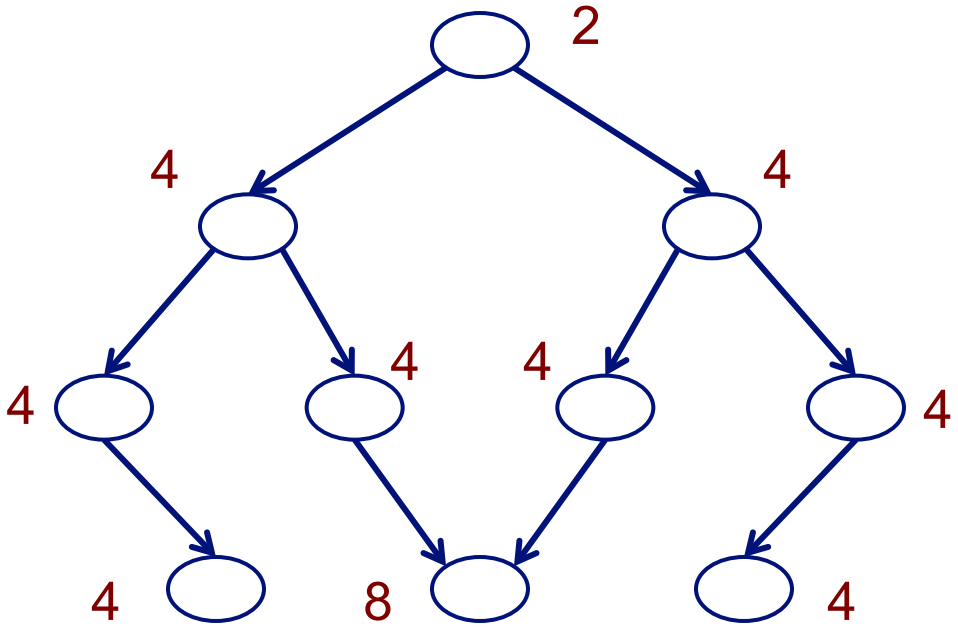
$$P(B,E,A,J,M) = P(B) \times P(E) \times P(A \mid B,E) \times P(J \mid A) \times P(M \mid A)$$

- a standard representation of the joint distribution for the Alarm example has $2^5 = 32$ parameters
- the BN representation of this distribution has 20 parameters

Bayesian networks

- consider a case with 10 binary random variables

- How many parameters does a BN with the following graph structure have?



= 42

- How many parameters does the standard table representation of the joint distribution have?

= 1024

Advantages of the Bayesian network representation

- Captures independence and conditional independence where they exist
- Encodes the relevant portion of the full joint among variables where dependencies exist
- Uses a graphical representation which lends insight into the complexity of inference

The inference task in Bayesian networks

Given: values for some variables in the network (*evidence*), and a set of *query* variables

Do: compute the posterior distribution over the query variables

- variables that are neither evidence variables nor query variables are *hidden* variables
- the BN representation is flexible enough that any set can be the evidence variables and any set can be the query variables

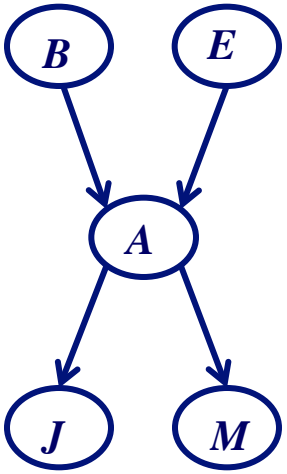
Inference by enumeration

- Fix the evidence variables (that is, ignore CPD entries inconsistent with the given setting for the evidence variables)
- For each setting of the query variable(s) sum out the remaining variables
- Normalize the resulting numbers

Inference by enumeration example

- let a denote $\boldsymbol{A}$=true, and $\neg a$ denote $\boldsymbol{A}$=false
- suppose we're given the query: $P(b \mid j, m)$

  "probability the house is being burglarized given that John and Mary both called"
- from the graph structure we can first compute:



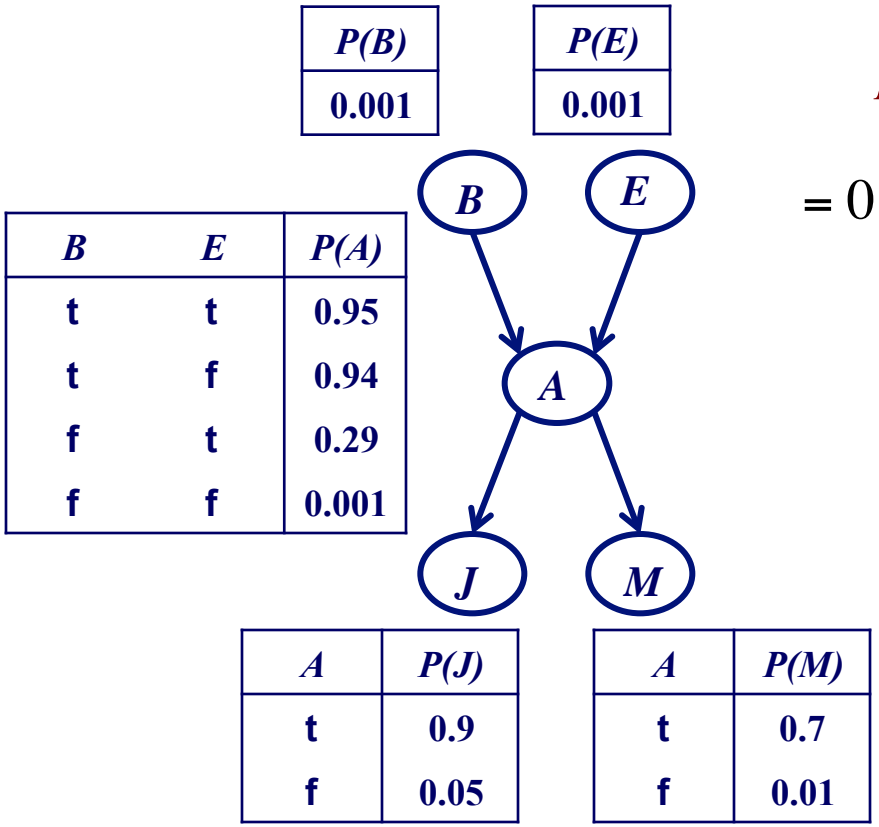
$$P(b, j, m) = \sum_e \sum_a P(b)P(e)P(a \mid b, e)P(j \mid a)P(m \mid a)$$

sum over possible values for E and A variables (e, $\neg e$, a, $\neg a$)

Inference by enumeration

$$P(b,j,m) = \sum_e \sum_a P(b)P(e)P(a \mid b,e)P(j \mid a)P(m \mid a)$$

$$= P(b) \sum_e \sum_a P(e)P(a \mid b,e)P(j \mid a)P(m \mid a)$$



$B \quad E \quad A \quad J \quad M$

$$= 0.001 \times (0.001 \times 0.95 \times 0.9 \times 0.7 + \quad e, a$$

$$0.001 \times 0.05 \times 0.05 \times 0.01 + \quad e, \neg a$$

$$0.999 \times 0.94 \times 0.9 \times 0.7 + \quad \neg e, a$$

$$0.999 \times 0.06 \times 0.05 \times 0.01) \quad \neg e, \neg a$$

Inference by enumeration

- now do equivalent calculation for $P(\neg b,\ j, m)$
- and determine $P(b \mid j, m)$

$$P(b \mid j,m) = \frac{P(b,j,m)}{P(j,m)} = \frac{P(b,j,m)}{P(b,j,m) + P(\neg b,j,m)}$$

We may write a CPT or a full joint distribution as a table.

B	E	A
b	e	.95
b	¬e	.94
¬b	e	.24
¬b	¬e	.05

⇒

	b		¬b	
	e	¬e	e	¬e
a	.95	.94	.24	.05
¬a	.05	.06	.76	.95

For full joint, all values sum to 1.0. For CPT, all values agreeing for all variables except consequent sum to 1.0.

When we (temporarily) ignore meaning of table, we call it a <u>potential</u> — table of values (non-negative) with no further constraints and to which we attach no meaning.

Algorithm to Multiply Two Potentials

Domain of (variables in) result is union of domains of input potentials. For each cell of result, multiply all cells of inputs that agree on variable settings.

	a	¬a
b	.1	.5
¬b	.2	.8

X

	c	¬c
b	.2	.4
¬b	.3	.5

=

	a		¬a	
	c	¬c	c	¬c
b	.02	.04	.10	.20
¬b	.06	.10	.24	.40

More Extreme Examples

	a	¬a
b	.1	.3
¬b	.2	.4

X

	d	¬d
c	.5	.8
¬c	.3	.6

=

		a		¬a	
		b	¬b	b	¬b
c	d	.05	.10	.15	.20
	¬d	.08	.16	.24	.32
¬c	d	.03	.06	.09	.12
	¬d	.06	.12	.18	.24

	a	¬a
b	.1	.4
¬b	.5	.1

X

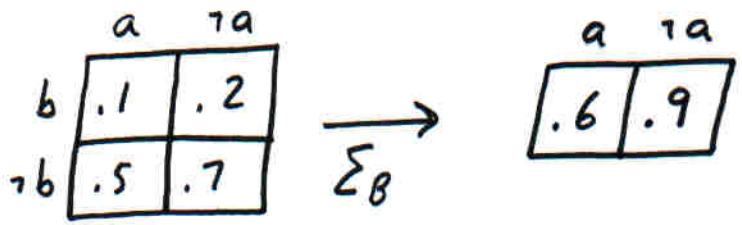
	a	¬a
b	.8	.9
¬b	.7	.8

=

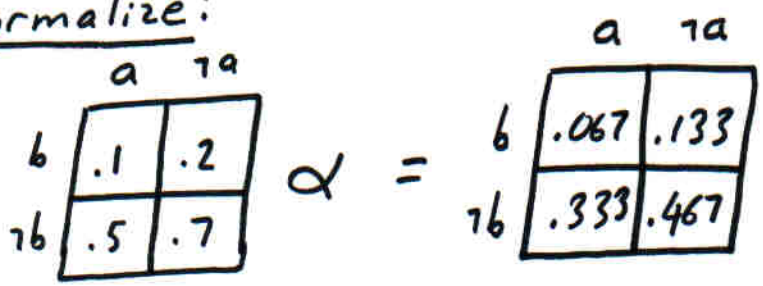
	a	¬a
b	.08	.36
¬b	.35	.08

Can also marginalize, normalize, divide potentials.

Marginalize (sum out):



Normalize:

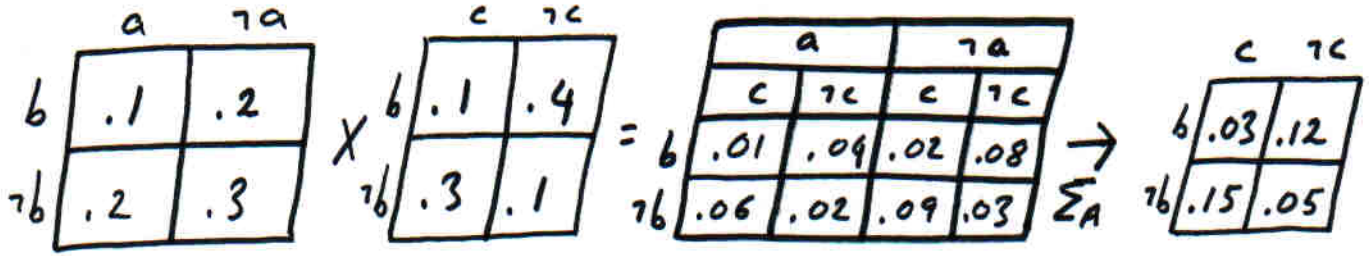


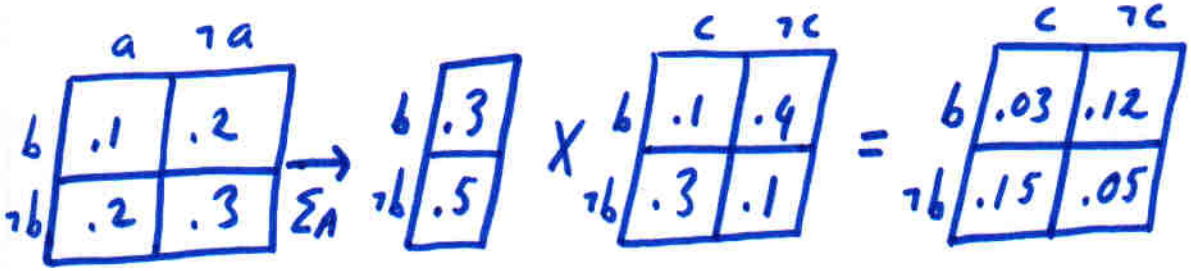
Will cover division later... won't need it until we come to junction tree inference.

Key Observation:

$\Sigma_A (P_1 \times P_2) = (\Sigma_A P_1) \times P_2$ if A is not in P_2.

Can marginalize (sum out A) before multiplying in this case — smaller intermediate table.





Inference by Enumeration:

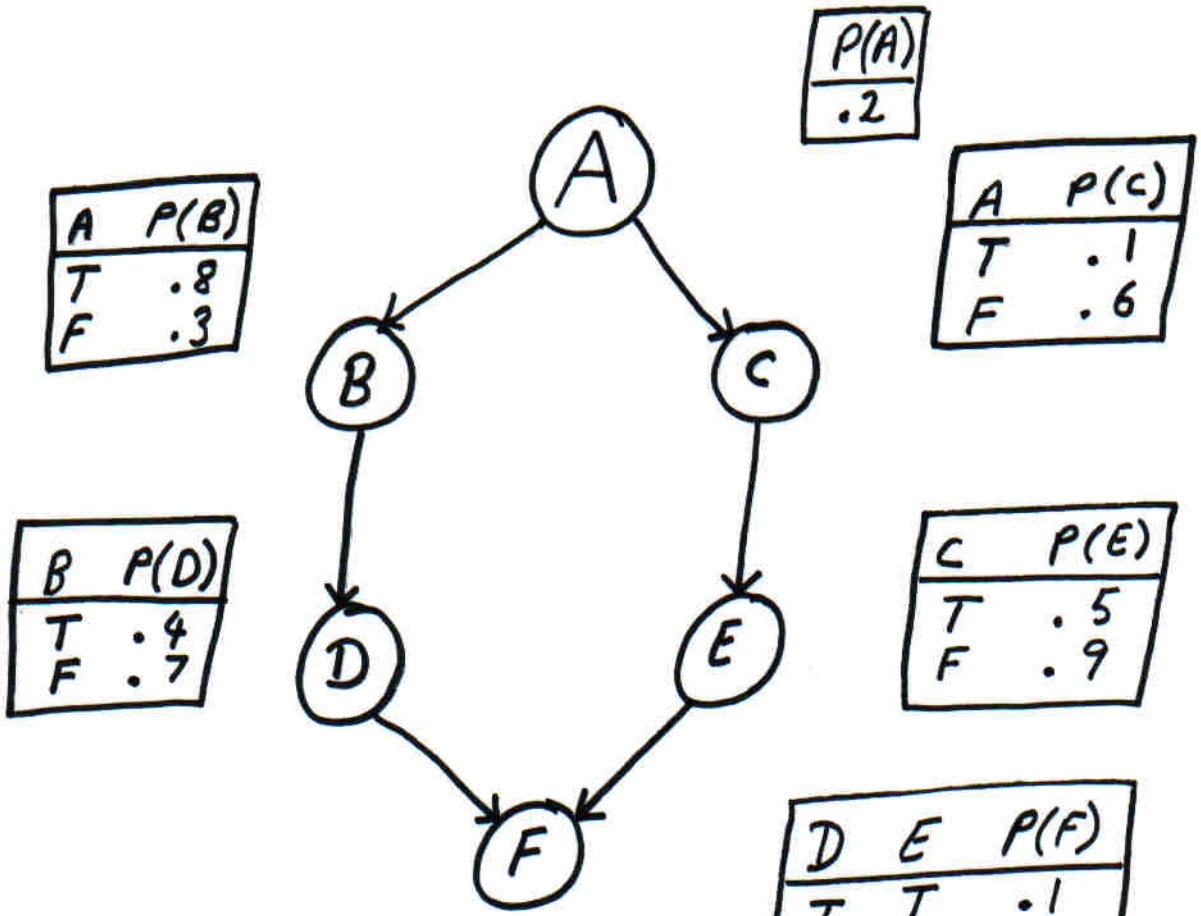
- Multiply all CPTs in Bayes net (any order).
- Eliminate some cells based on evidence.
- Marginalize down to query variable (sum out all others in any order).

Variable Elimination

While multiplying, once we've included all tables using a given variable, go ahead and sum it out then rather than waiting.

Variable Elimination Procedure

- The initial potentials are the CPTs in BN.
- Repeat until only query variable remains:
  - Choose another variable to eliminate.
  - Multiply all potentials that contain the variable.
  - If no evidence for the variable then sum the variable out and replace original potential by the new result.
  - Else, remove variable based on evidence.
- Normalize remaining potential to get the final distribution over the query variable.



P(A)
.2

A	P(B)
T	.8
F	.3

A	P(C)
T	.1
F	.6

B	P(D)
T	.4
F	.7

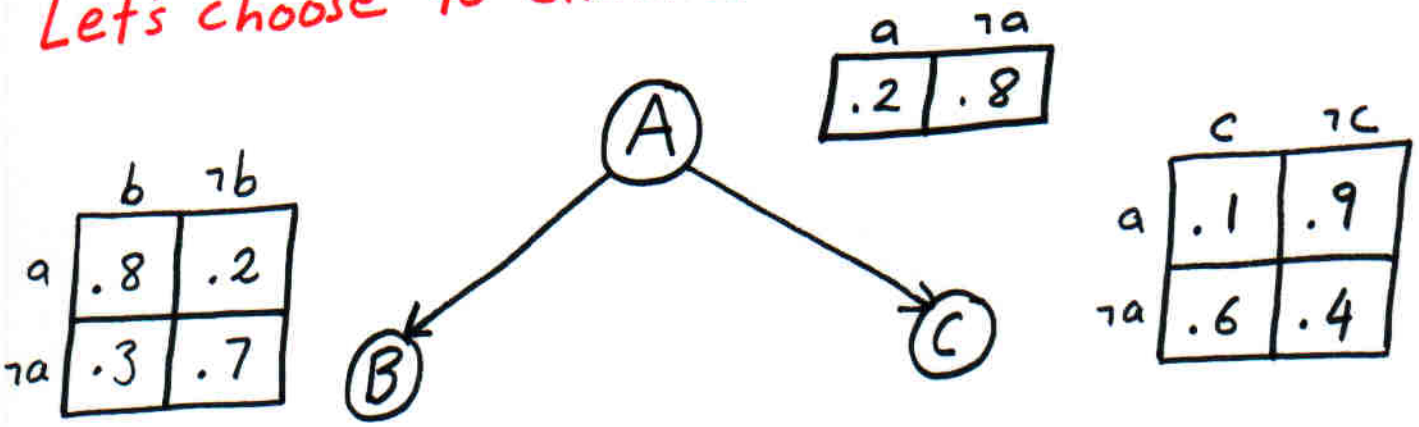
C	P(E)
T	.5
F	.9

D	E	P(F)
T	T	.1
T	F	.3
F	T	.4
F	F	.9

$$P(A,B,C,D,E,F) = P(A)\,P(B|A)\,P(C|A)\,P(D|B)\,P(E|C)\,P(F|D,E)$$

Query: $P(F \mid C = \text{true})$

Let's choose to eliminate in order A, B, C, D, E.

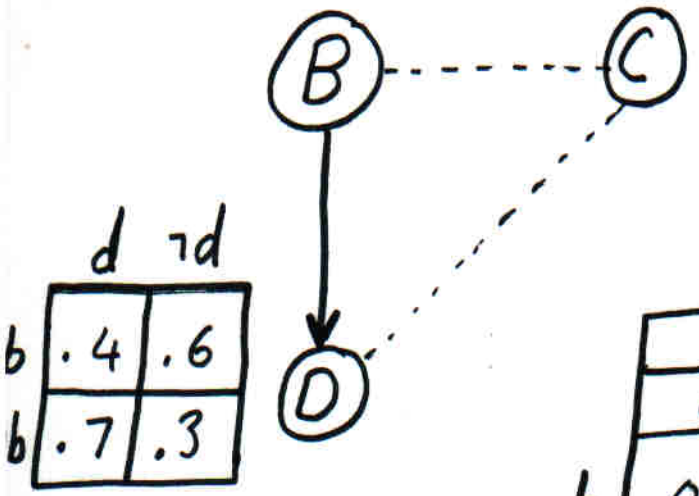


Before eliminating A, must multiply all potentials that involve A.

	b		¬b	
	c	¬c	c	¬c
a	.016	.144	.004	.036
¬a	.144	.096	.336	.224
	.16	.24	.34	.26

Eliminate A by summing it out.

Next eliminate B. Multiply potentials involving B...



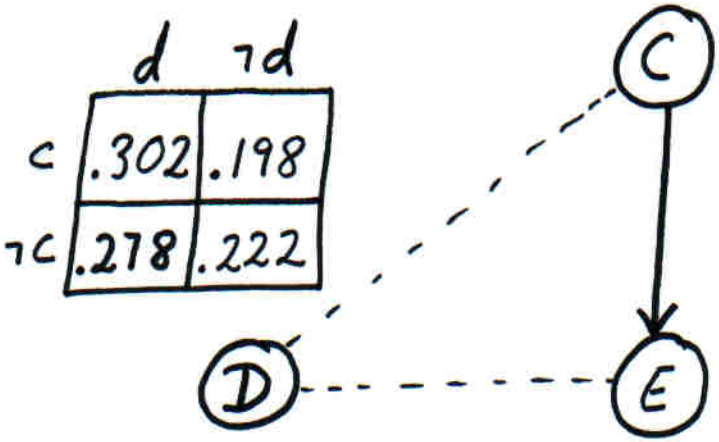
	c	¬c
b	.16	.24
¬b	.34	.26

	d	¬d
b	.4	.6
¬b	.7	.3

	c		¬c	
	d	¬d	d	¬d
b	.064	.096	.096	.144
¬b	.238	.102	.182	.078
	.302	.198	.278	.222

Sum out B

Next eliminate C ...



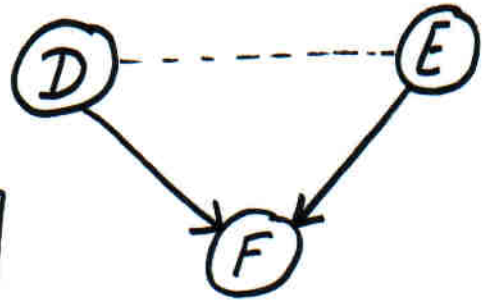
	d	¬d
c	.302	.198
¬c	.278	.222

	e	¬e
c	.5	.5
¬c	.9	.1

	d		¬d	
	e	¬e	e	¬e
c	.151	.151	.099	.099
¬c	.250	.028	.200	.022

We have evidence for C, so eliminate C by ← evidence.

Next eliminate D...

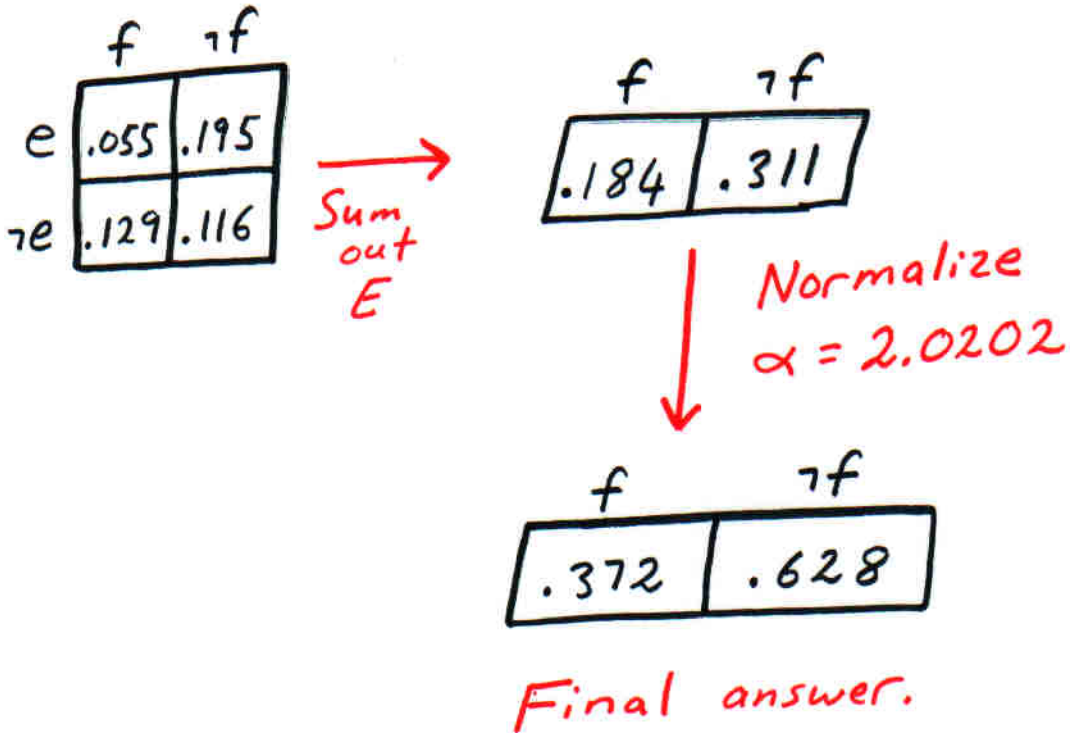


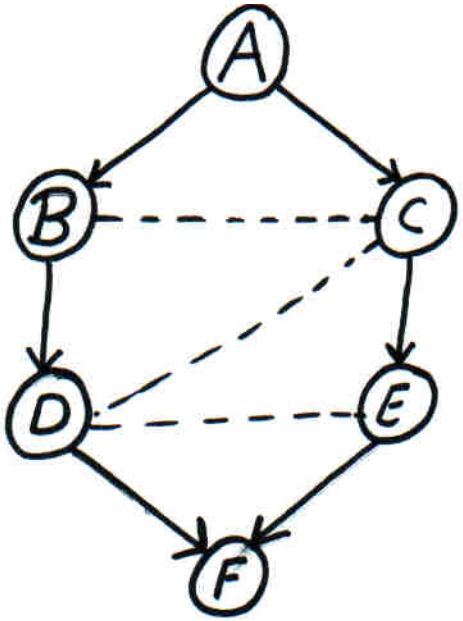
	d	¬d
e	.151	.099
¬e	.151	.099

	d		¬d	
	e	¬e	e	¬e
f	.1	.3	.4	.9
¬f	.9	.7	.6	.1

	e		¬e	
	f	¬f	f	¬f
d	.015	.136	.040	.106
¬d	.040	.059	.089	.010
	.055	.195	.129	.116

Sum out D


f ¬f
e .055 .195
¬e .129 .116
Sum out E
f ¬f
.184 .311
Normalize
α = 2.0202
f ¬f
.372 .628
Final answer.



Each product of potentials that we computed happens to correspond to a clique in ~~the~~ a "triangulated" graph.

(may be several triangulations corresponding to different elimination orders)

Intuition of junction trees ... store the potentials in an initial computation, given some evidence; then can quickly answer for any query variable. Lower amortized cost.

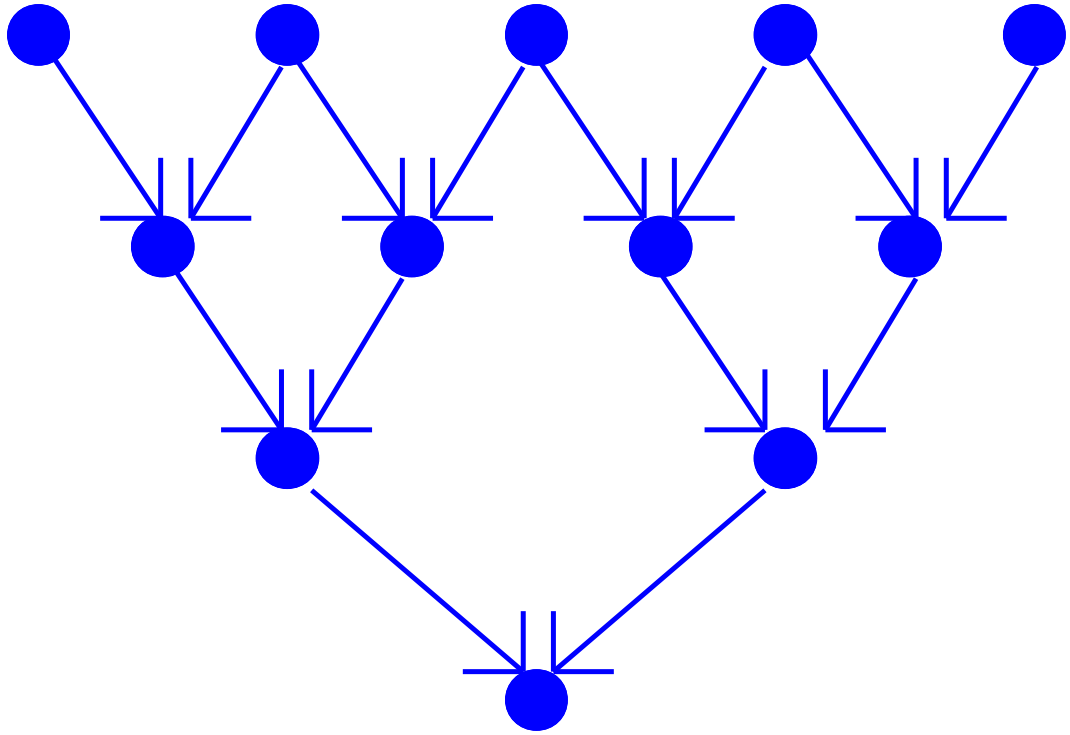
This link between V.E. and J.T. due to d'Ambrosio.

Junction Trees: Motivation

- Standard algorithms (e.g., variable elimination) are inefficient if the undirected graph underlying the Bayes Net contains cycles.
- We can avoid cycles if we turn highly-interconnected subsets of the nodes into "supernodes."

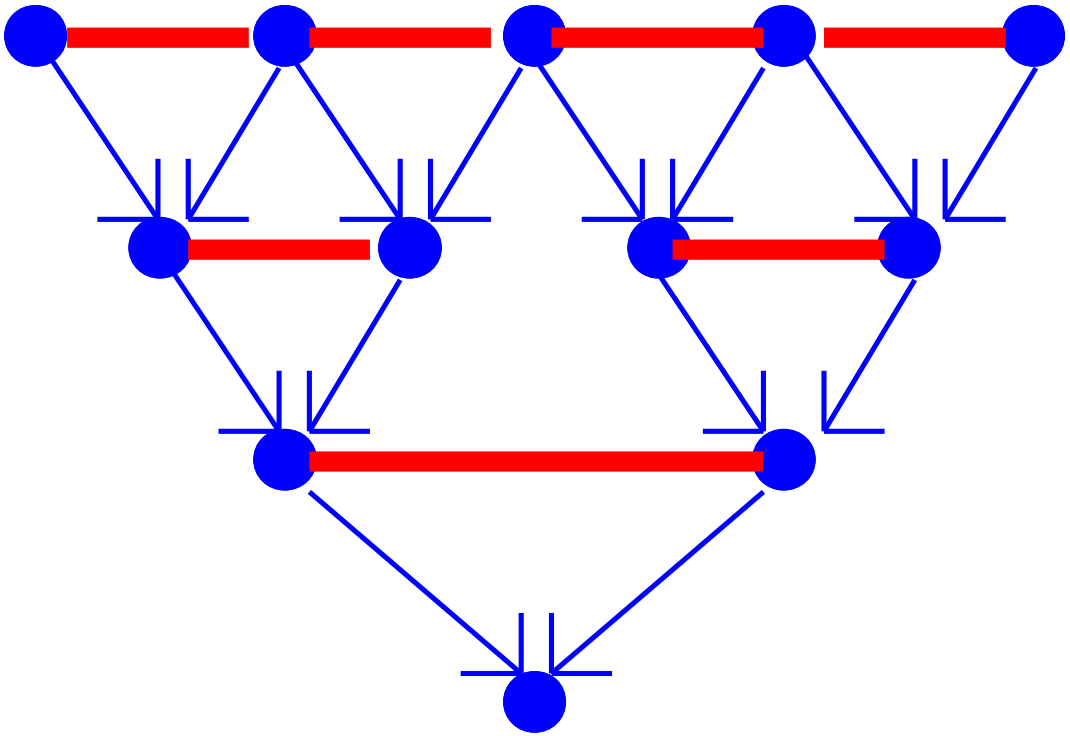
A Running Example for the Steps in Constructing a Junction Tree

Imagine we start with a Bayes Net having the following structure

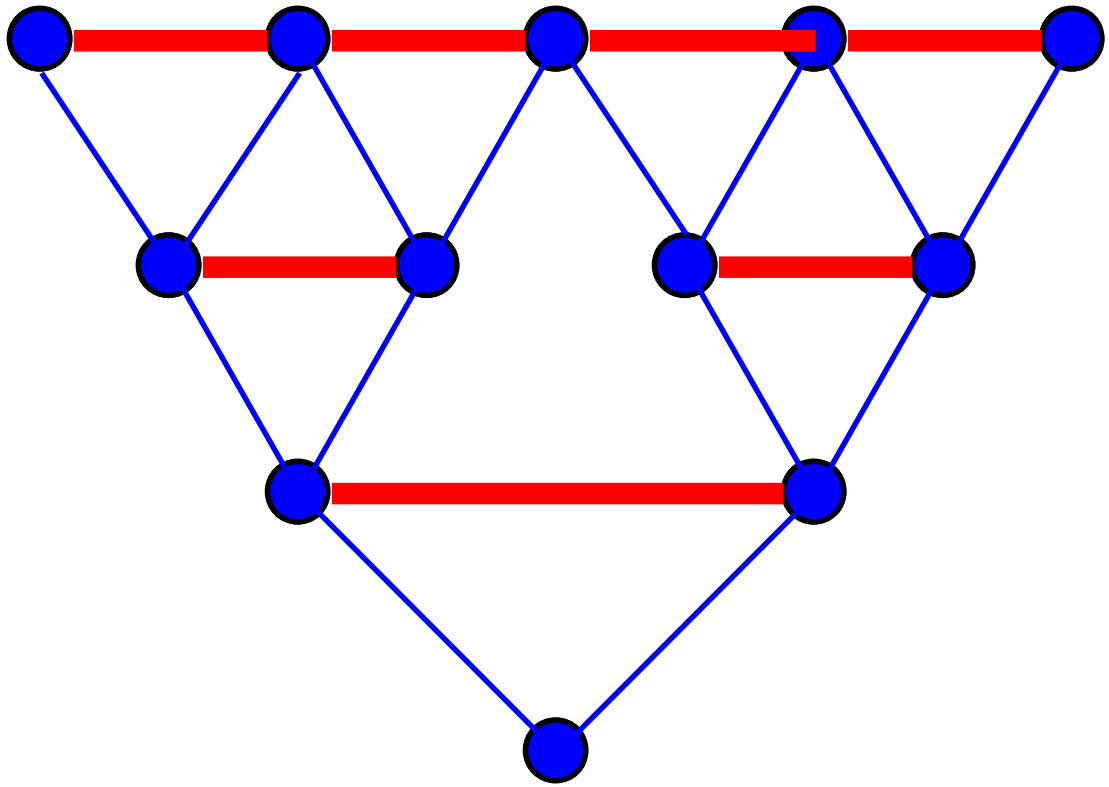


Step 1: Make the Graph Moral

Add an edge between non-adjacent (unmarried) parents of the same child.

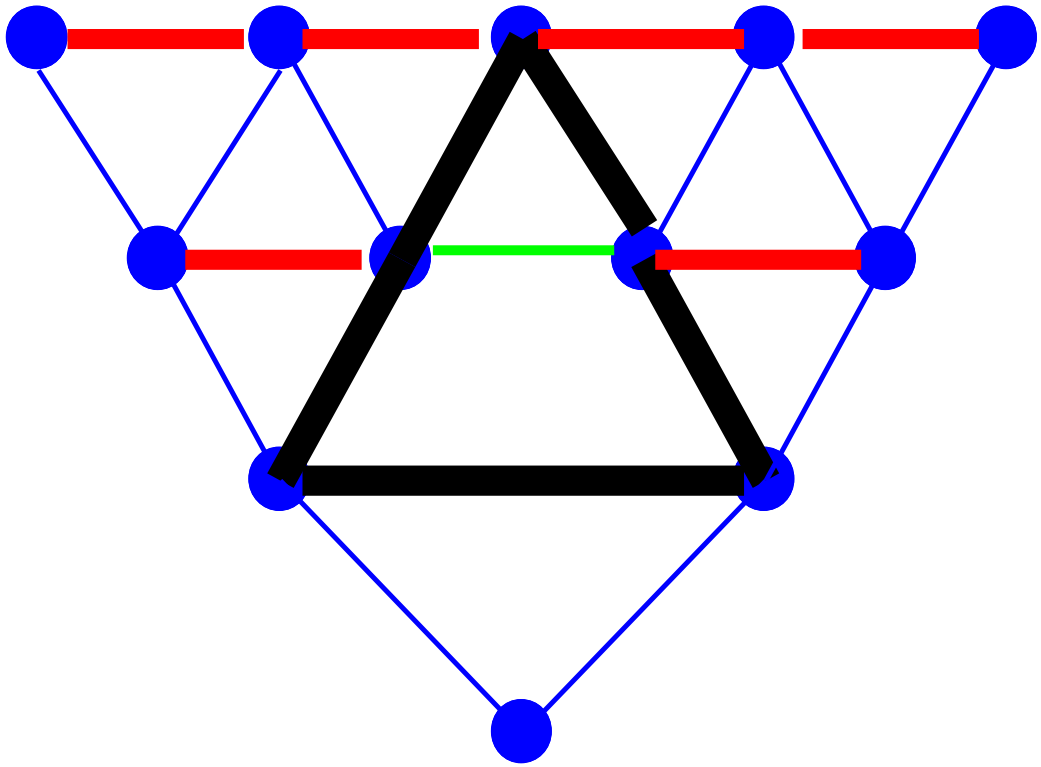


Step 2: Remove Directionality



Step 3: Triangulate the Graph

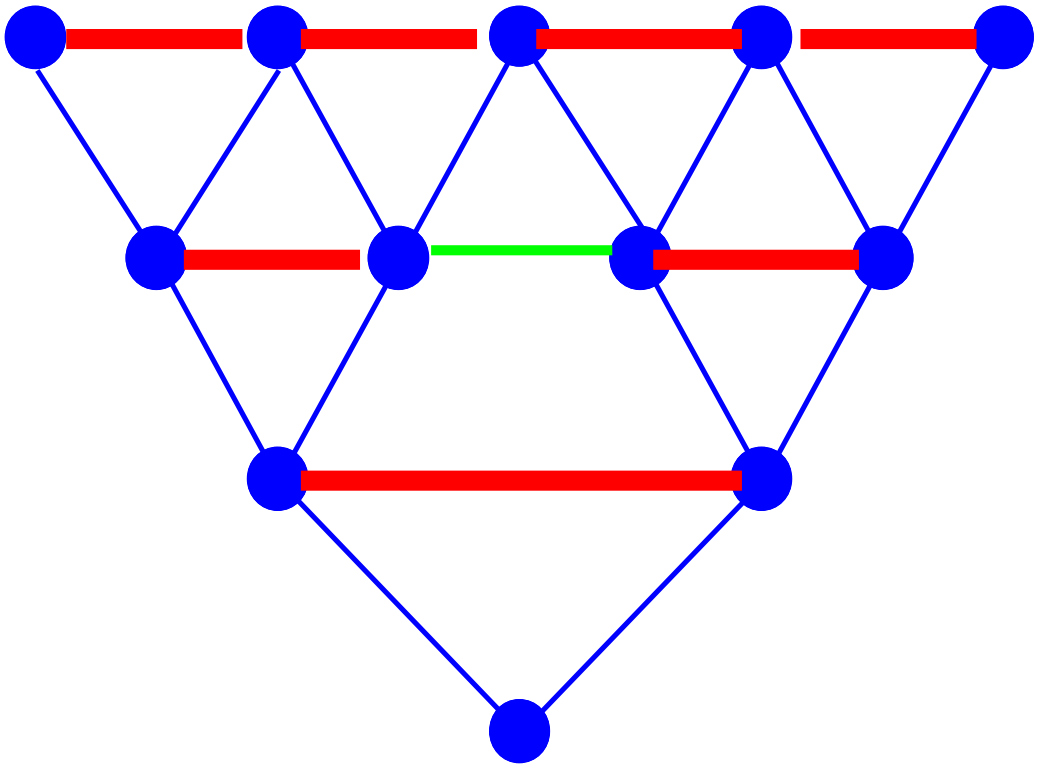
Repeat while there exists a cycle of length > 3 with no chord:
Add a chord (edge between two non-adjacent vertices in such a cycle).



Step 3: Triangulate the Graph

Repeat while there exists a cycle of length > 3 with no chord joining vertices that are not adjacent in the cycle:

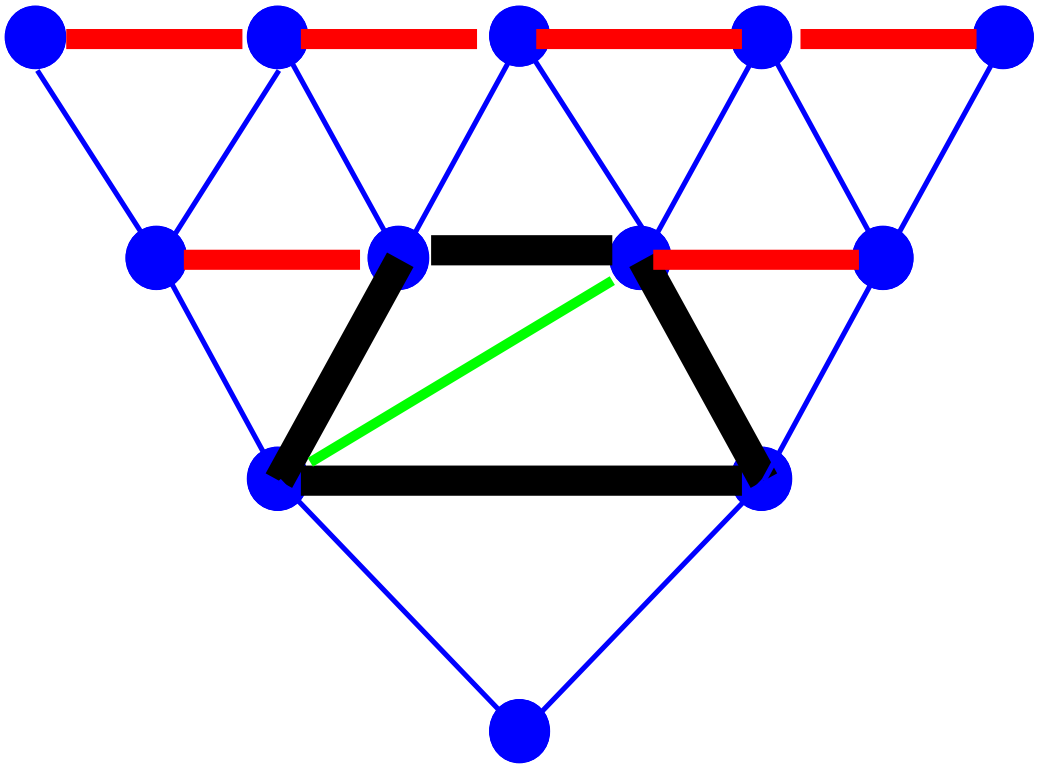
Add an edge (chord) between two non-adjacent vertices in such a cycle.



Step 3: Triangulate the Graph

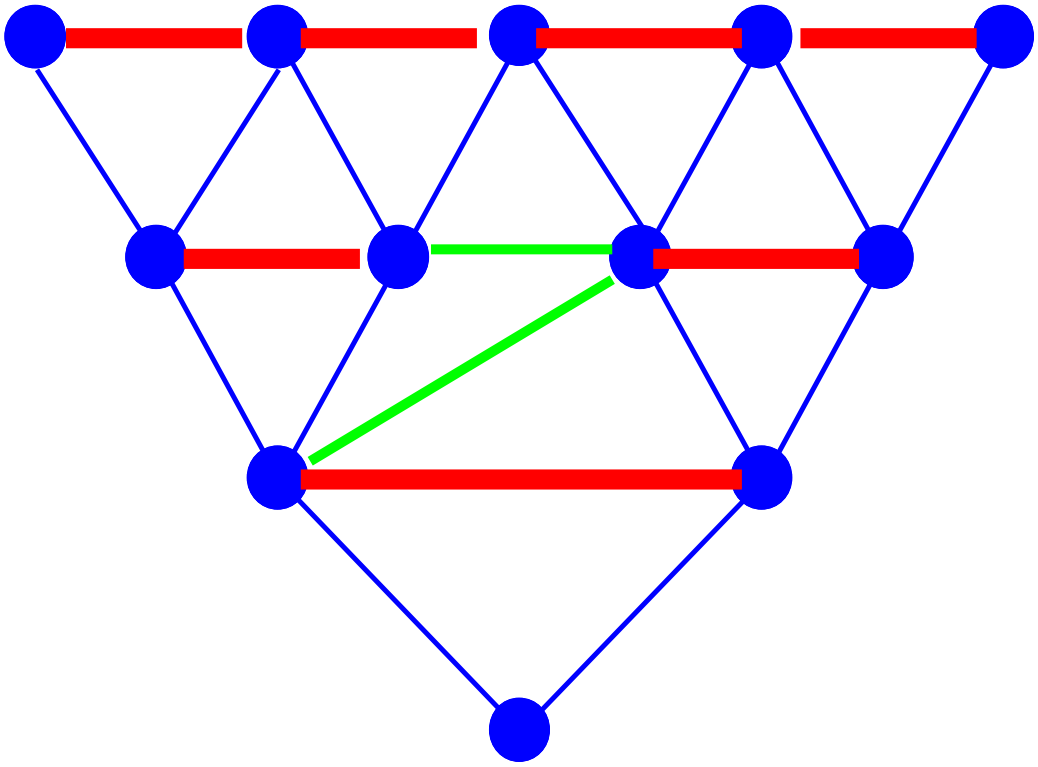
Repeat while there exists a cycle of length > 3 with no chord joining vertices that are not adjacent in the cycle:

Add an edge (chord) between two non-adjacent vertices in such a cycle.



Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



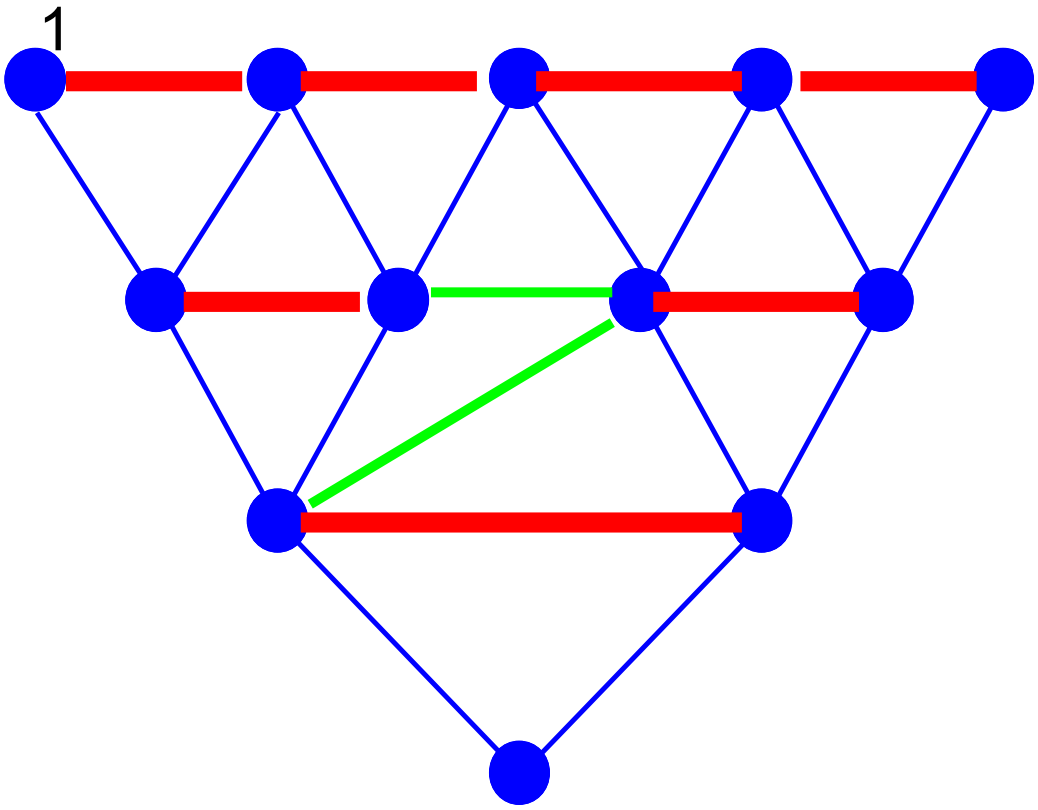
Triangulation Checking

The following *MaximumCardinalitySearch* algorithm terminates with success if and only if the graph is triangulated. It processes each node, and the time to process a node is quadratic in the number of adjacent nodes.

Choose any node in the graph and label it 1.
For i = 2 to n (total number of nodes in the graph):
Choose the node with the most labeled neighbors and label it i.
If any two labeled neighbors of i are not adjacent to each other, FAIL.
SUCCEED.

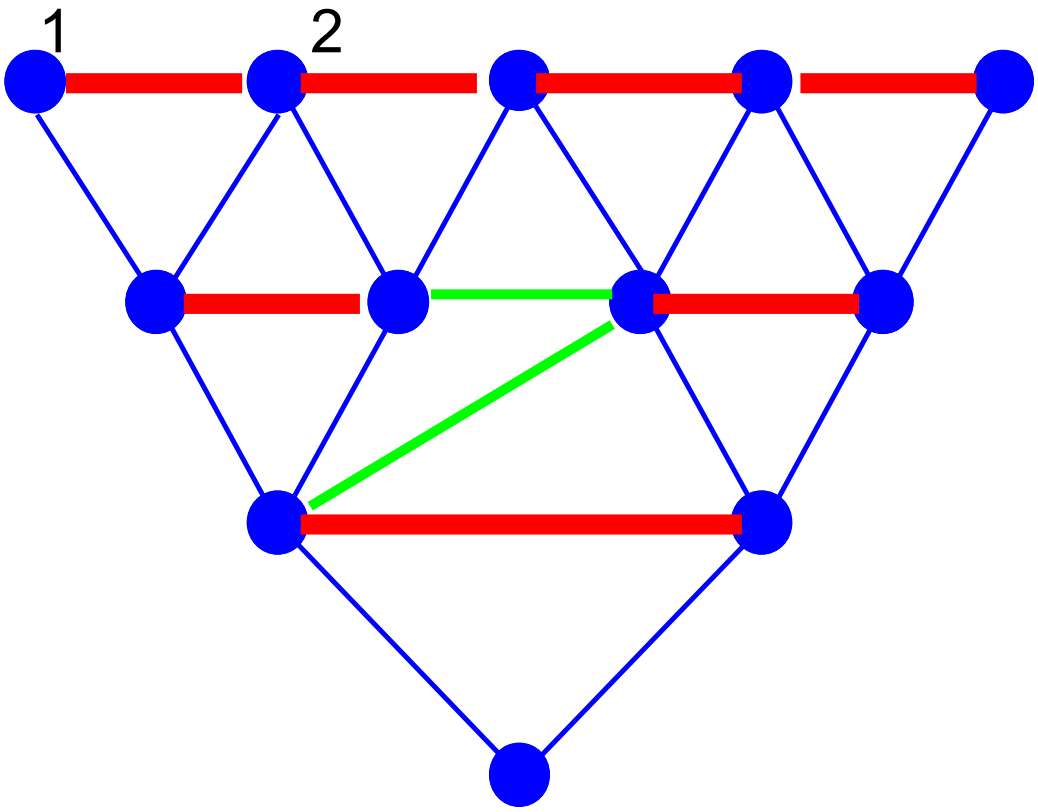
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



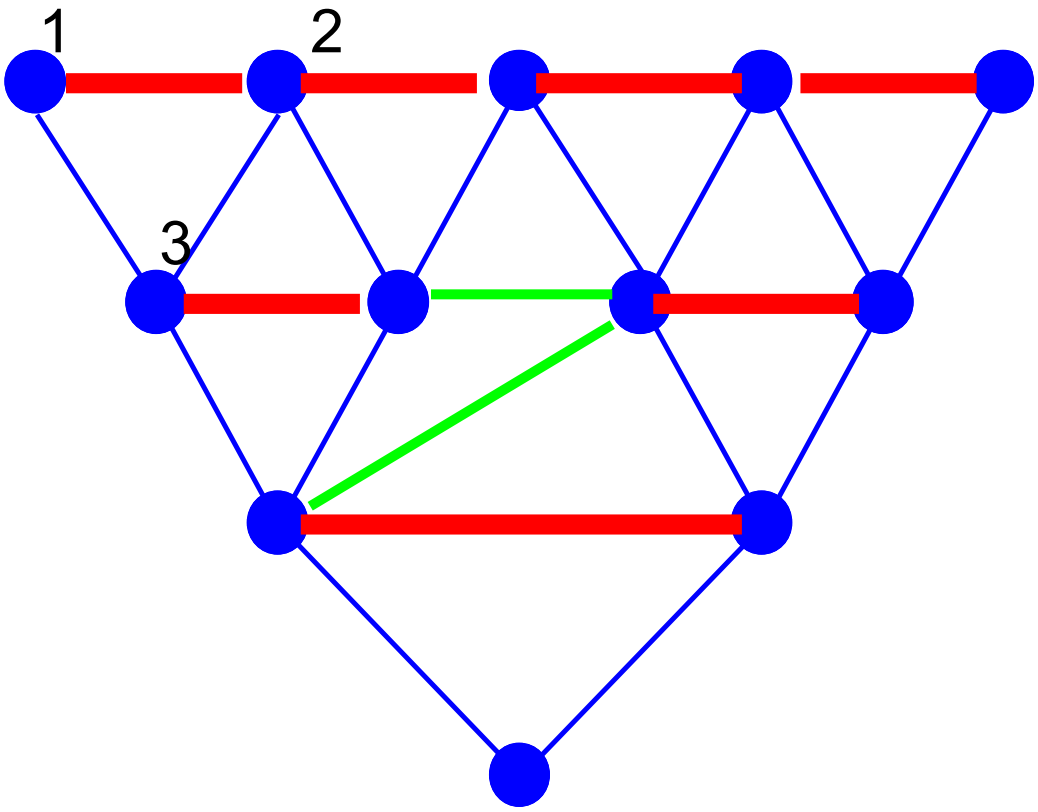
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



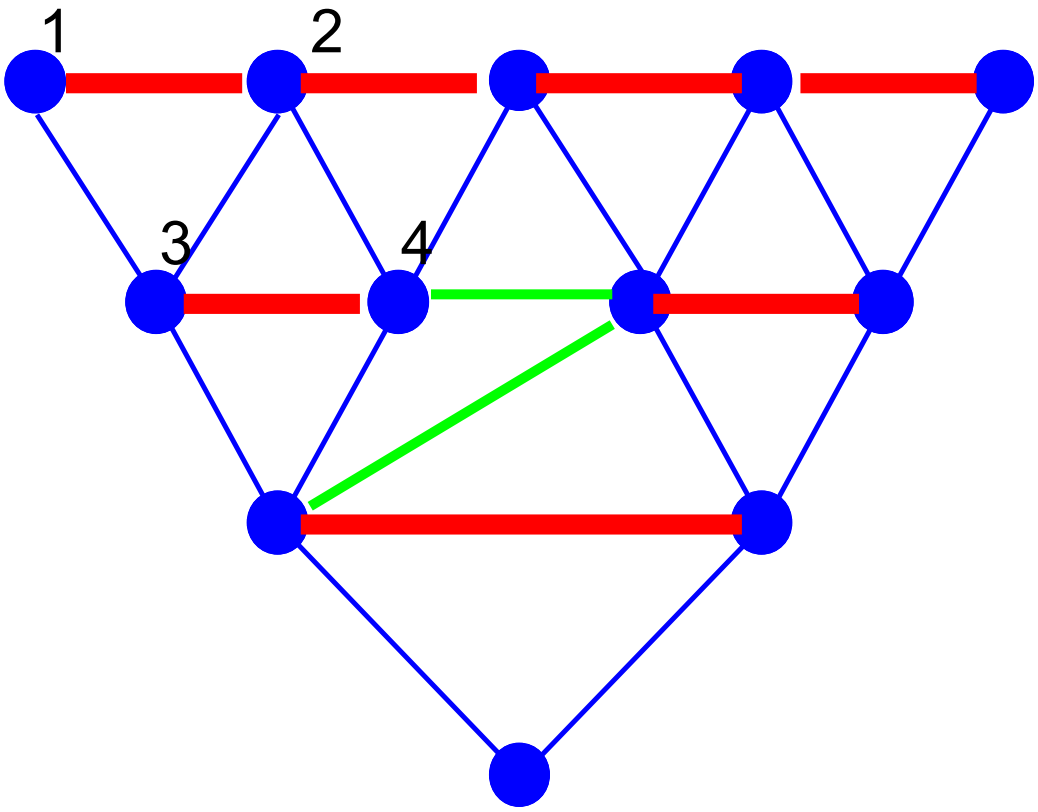
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



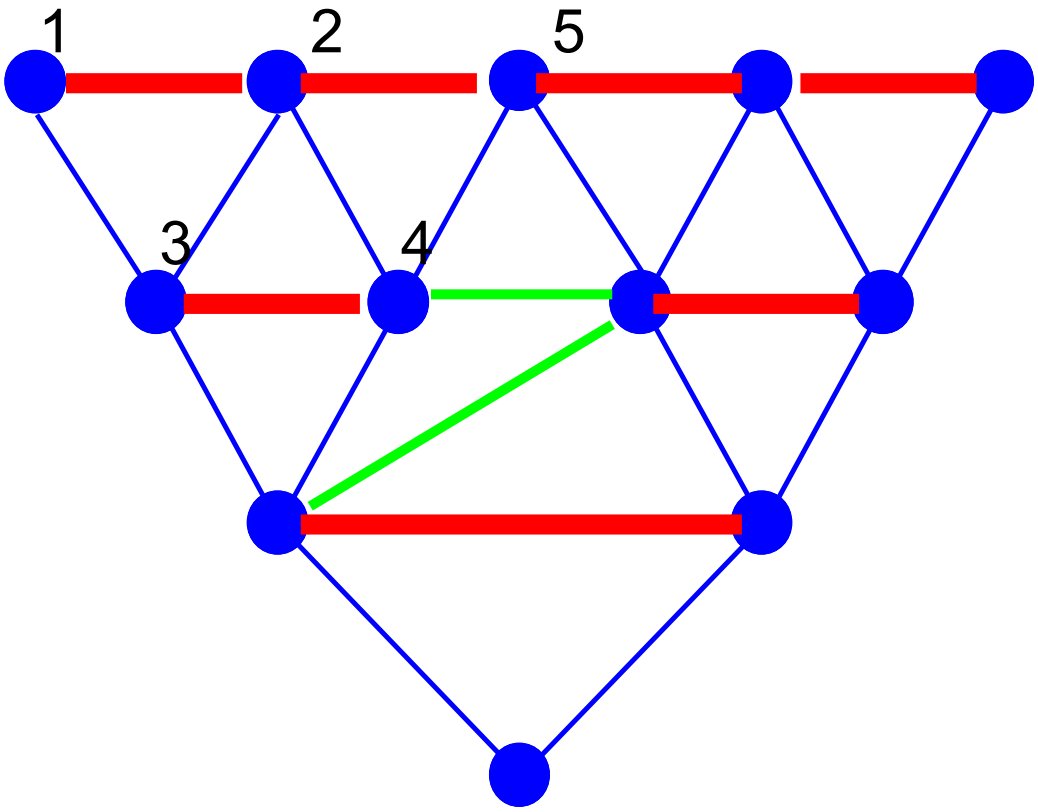
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



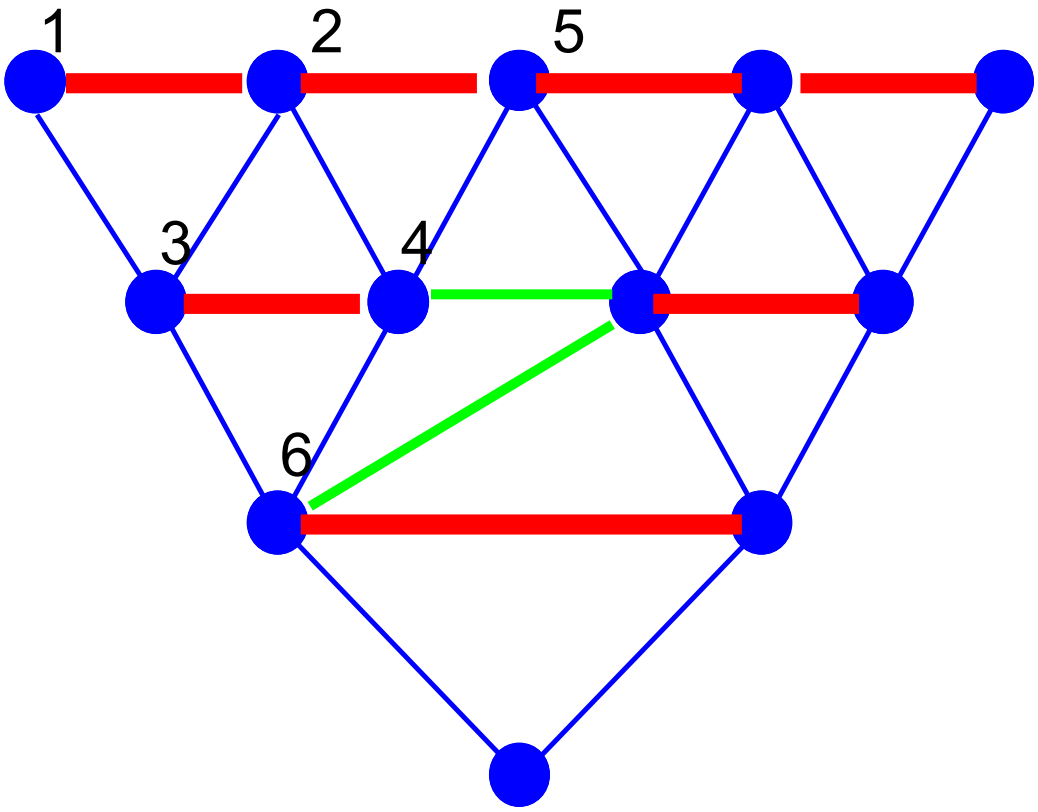
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?



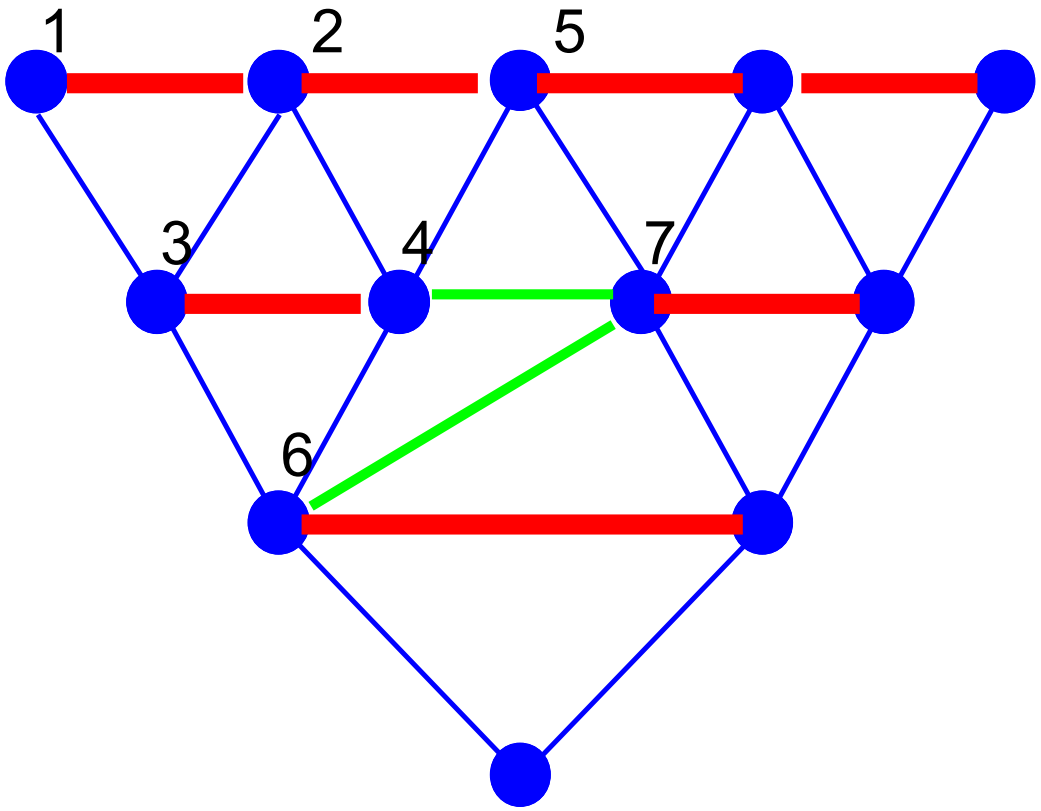
Is it Triangulated Yet?

It appears to be triangulated, but how can we be sure?

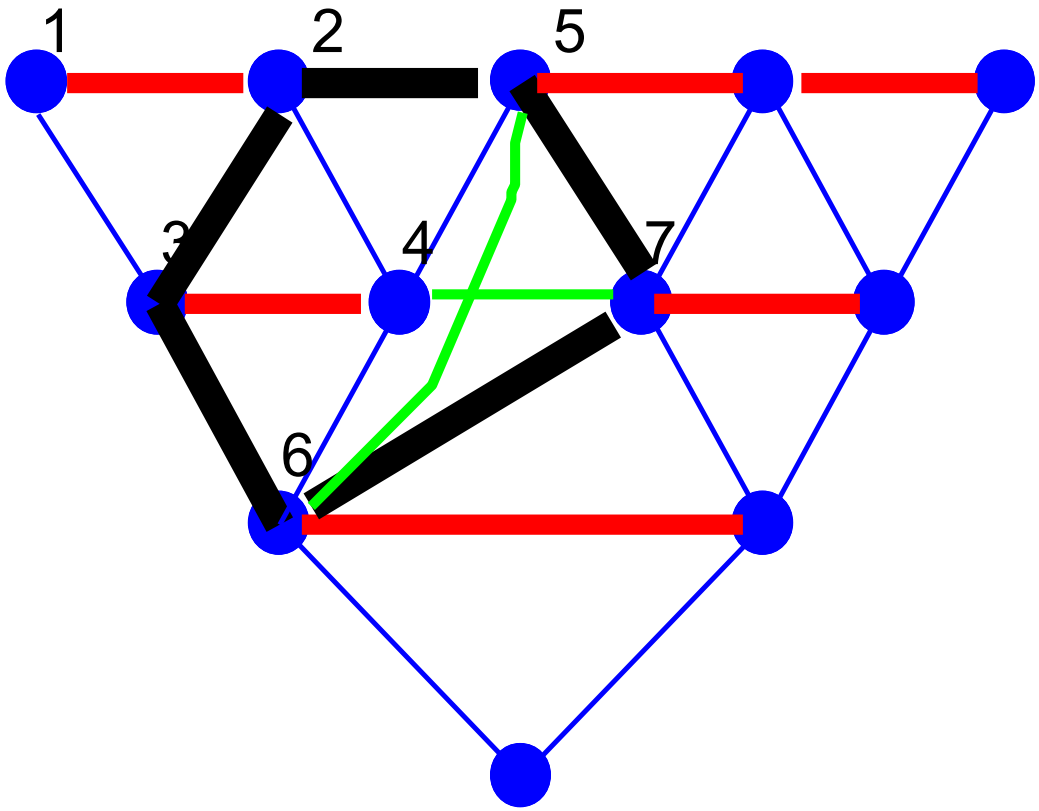


It is Not Triangulated

No edge between nodes 5 and 6, both of which are parents of 7.

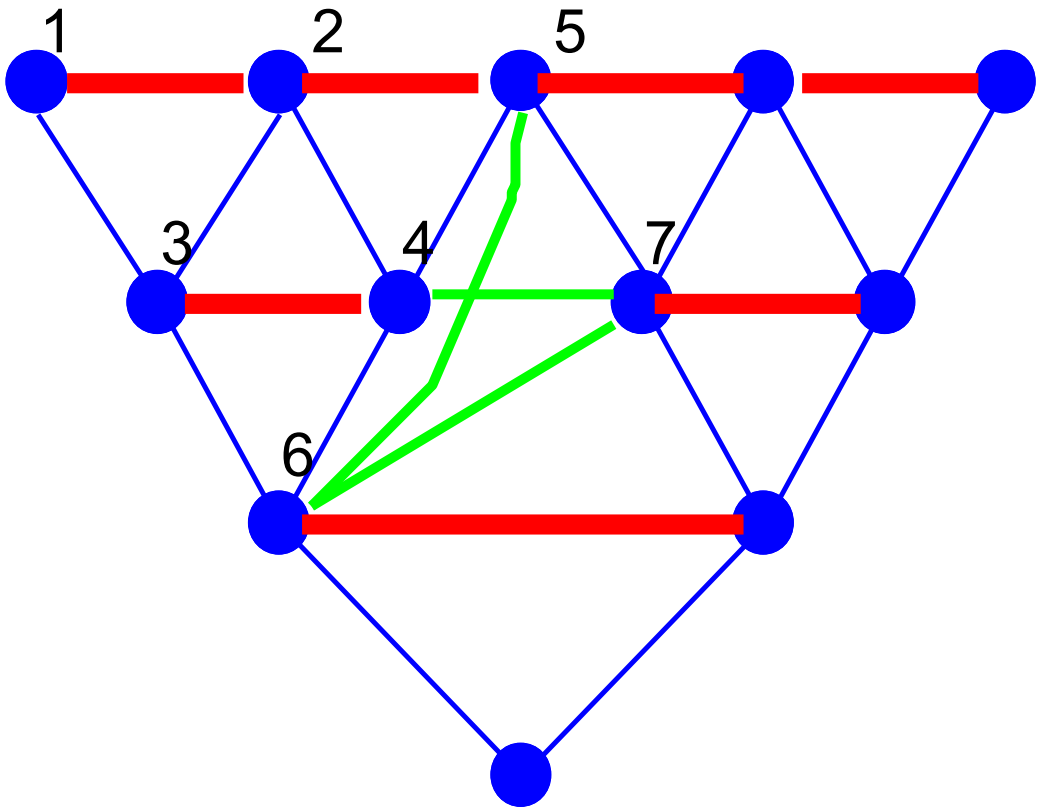


Fixing the Faulty Cycle



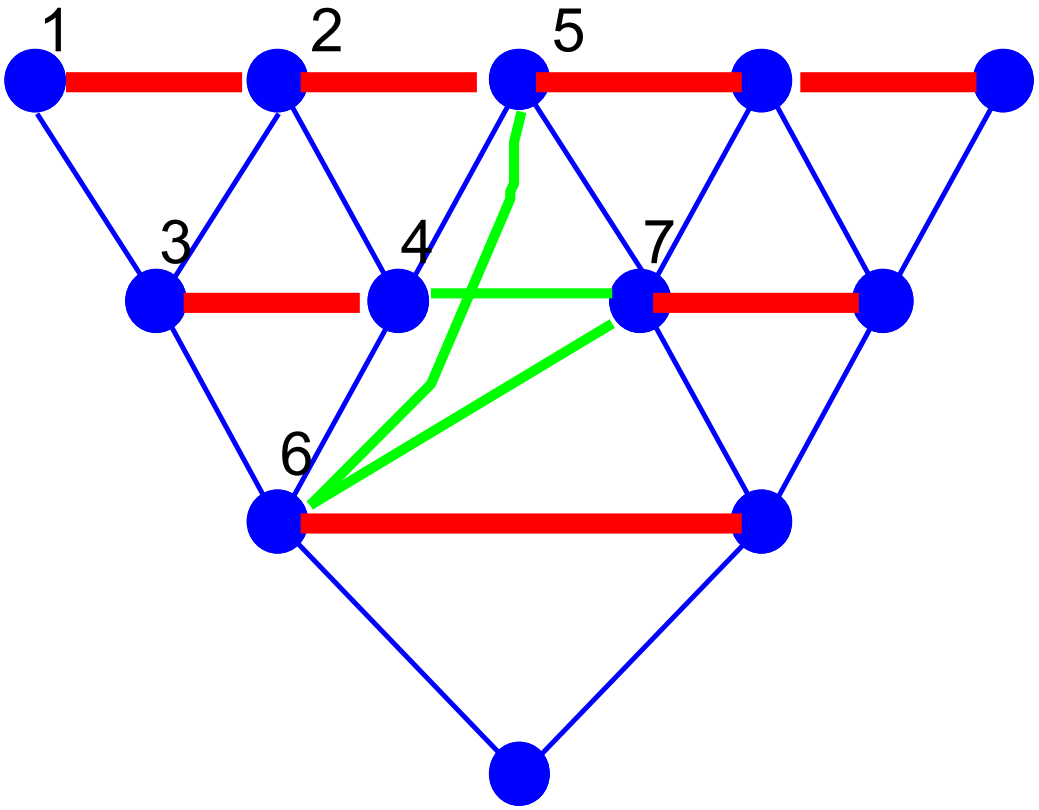
Continuing our Check...

Must restart algorithm. Ordering might change, or we might have. introduced a new cycle. Imagine repeating procedure with this graph.



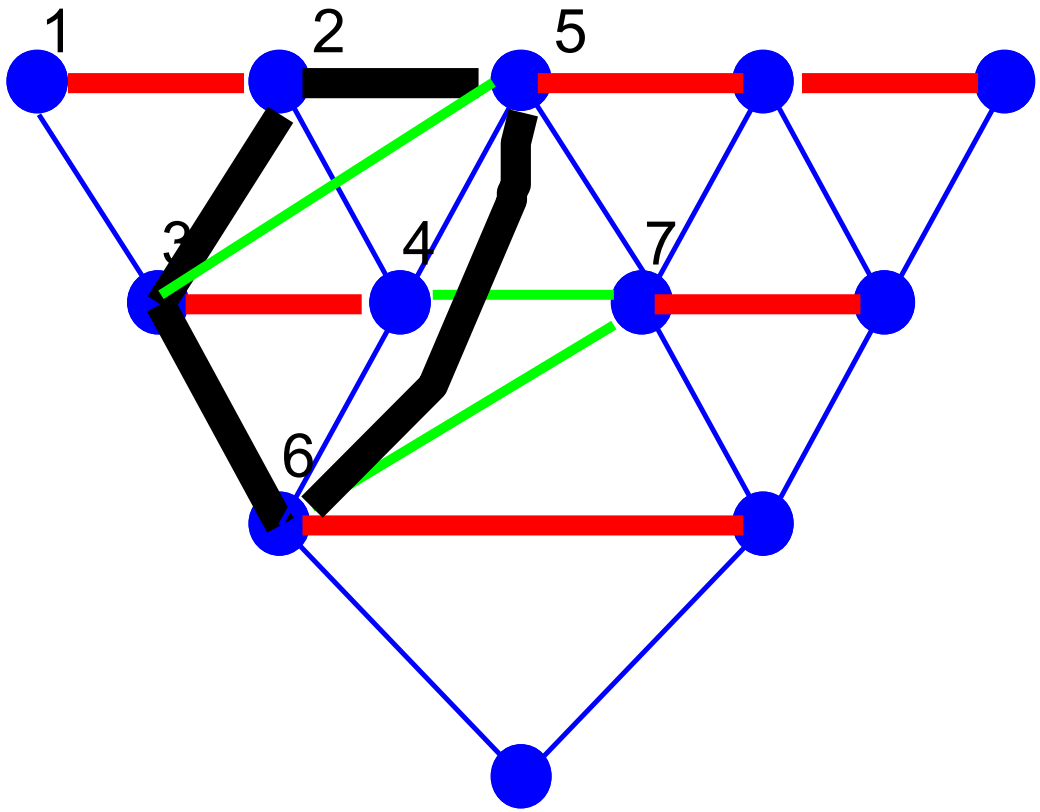
Continuing our Check...

Following our earlier change, 5 is now a neighbor of 6. When we reach
6 we see that its neighbors 5 and 3 are not adjacent -- another fix...



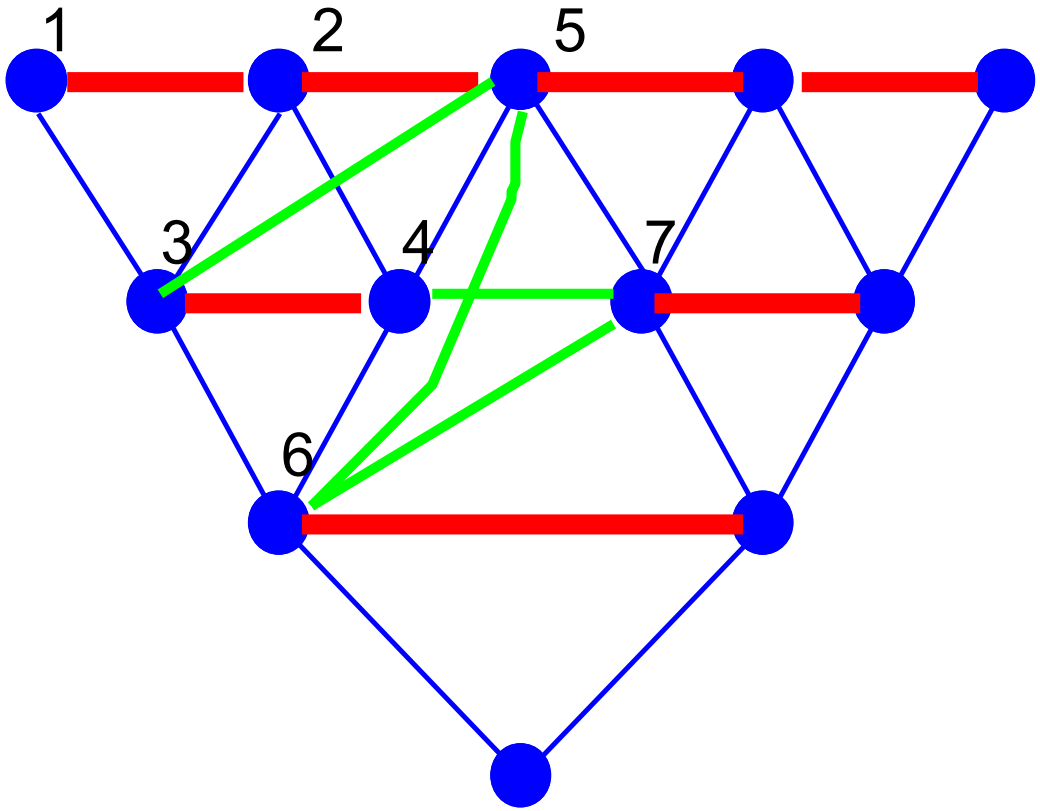
Fixing this Problem

Adding an edge between the non-adjacent parents 3 and 5 of 6 provides a missing chord as below.

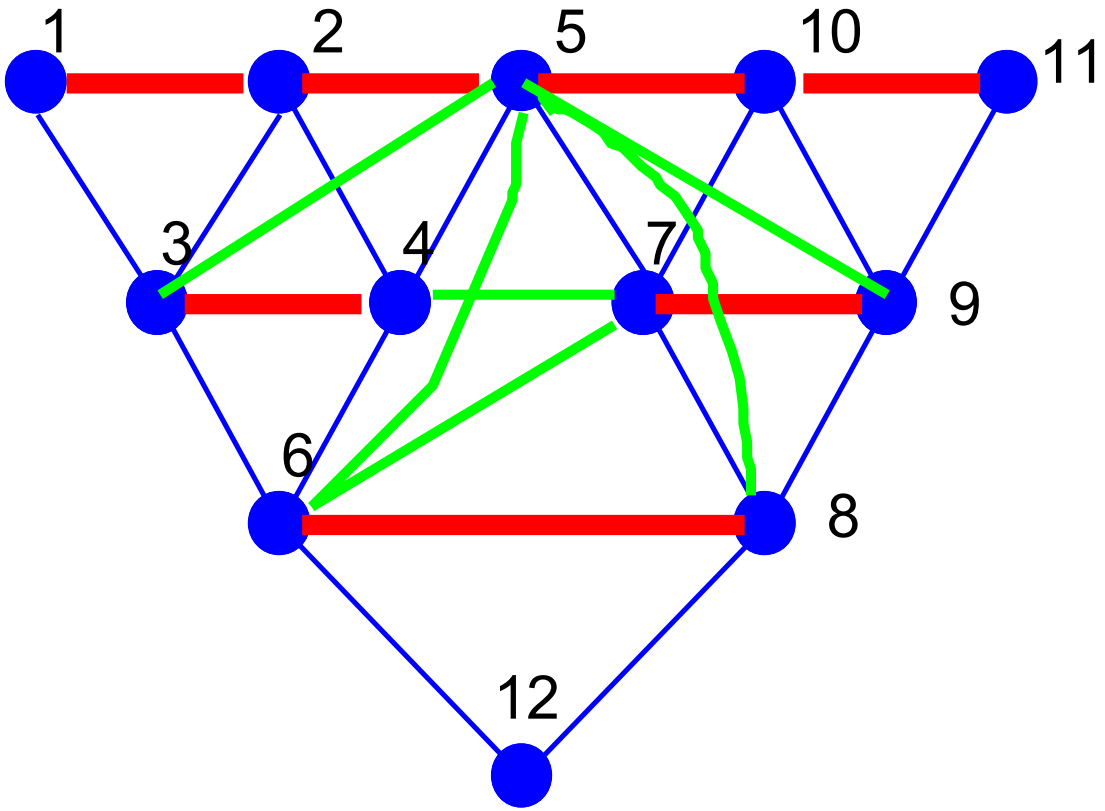


Continuing our Check...

We now have the graph below. Restarting, everything is fine through node 7 as below. As we continue, we end up adding two more edges and finally succeeding with ...



The Following is Triangulated



Triangulation: Key Points

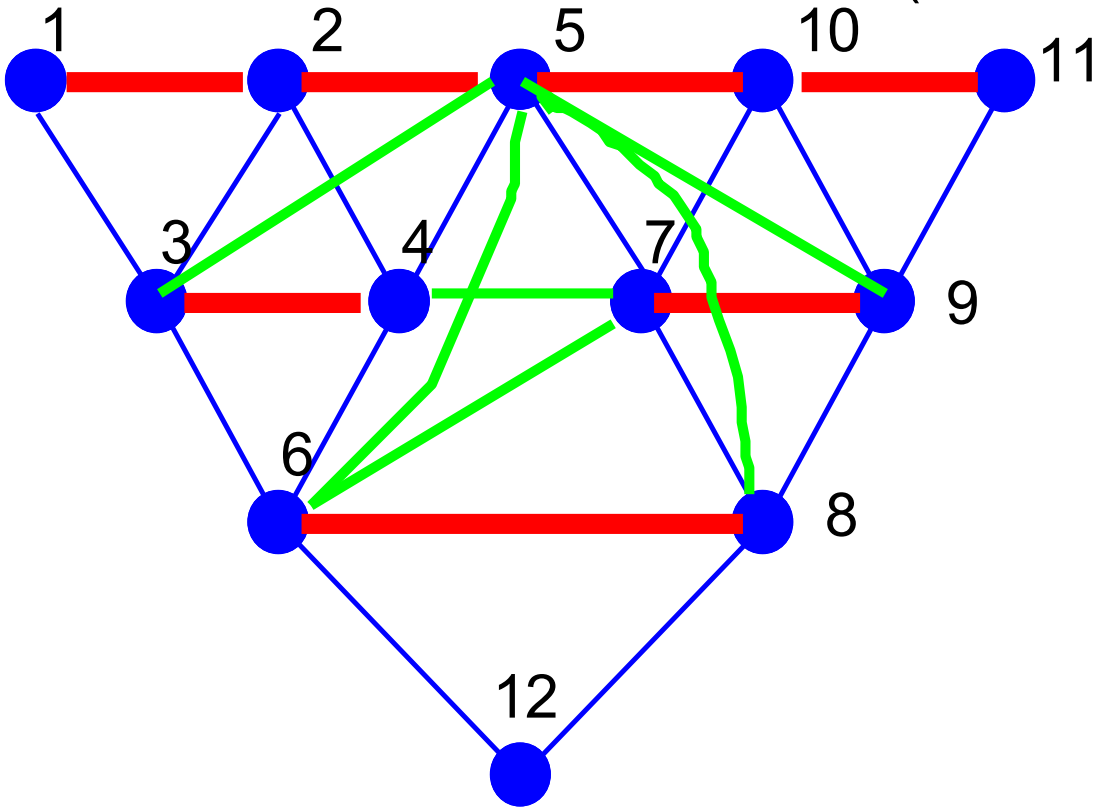
- Previous algorithm is an efficient *checker*, but not necessarily best way to triangulate.
- In general, many triangulations may exist. The only efficient algorithms are heuristic.
- Jensen and Jensen (1994) showed that any scheme for exact inference (belief updating given evidence) must perform triangulation (perhaps hidden as in Draper 1995).

Definitions

- *Complete graph or node set*: all nodes are adjacent.
- *Clique*: maximal complete subgraph.
- *Simplicial node*: node whose set of neighbors is a complete node set.

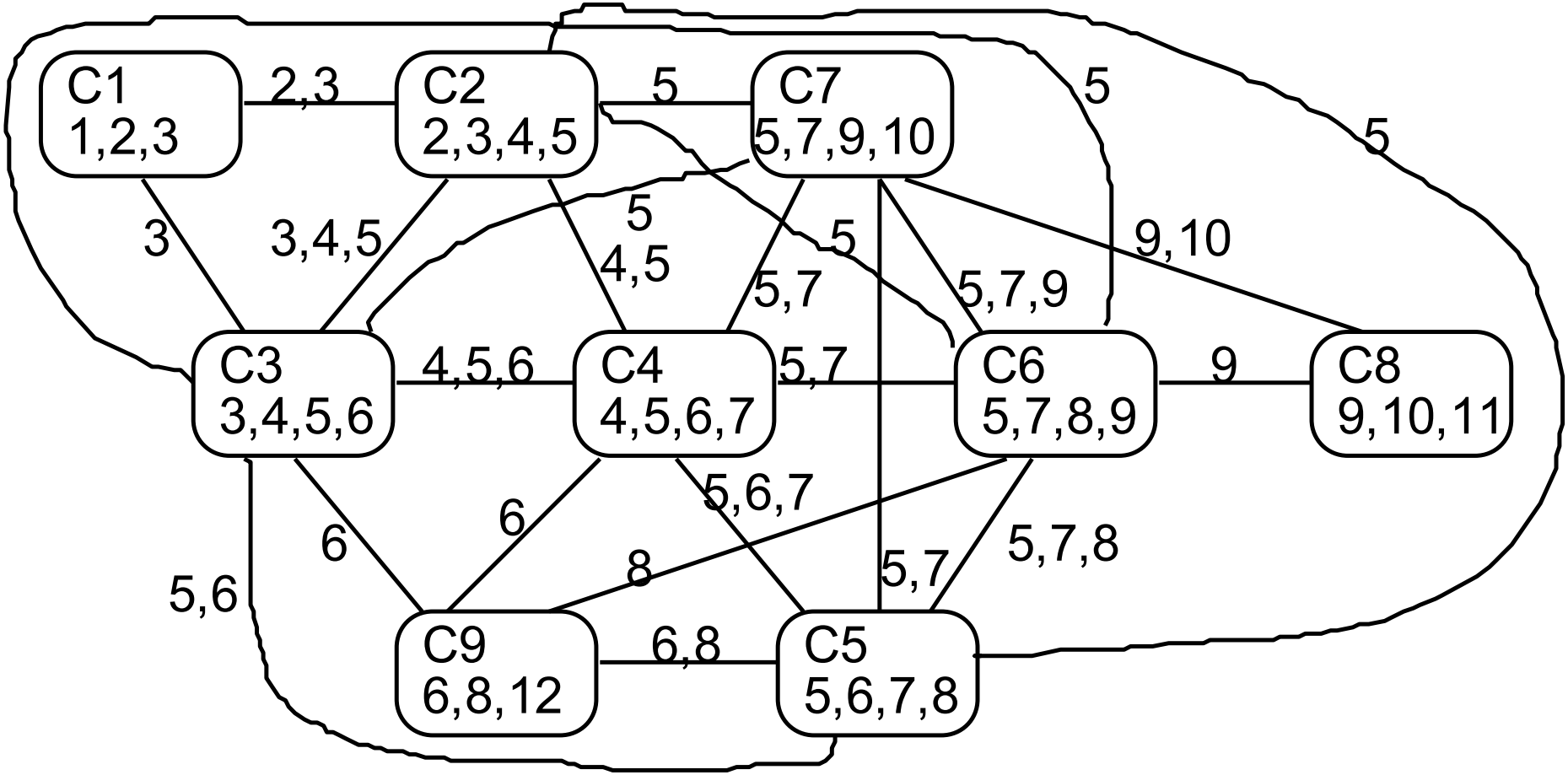
Step 4: Build Clique Graph

Find all cliques in the moralized, triangulated graph. A clique becomes a node in the clique graph. If two cliques intersect below, they are joined in the clique graph by an edge labeled with their intersection from below (shared nodes).



The Clique Graph

The label of an edge between two cliques is called the separator.



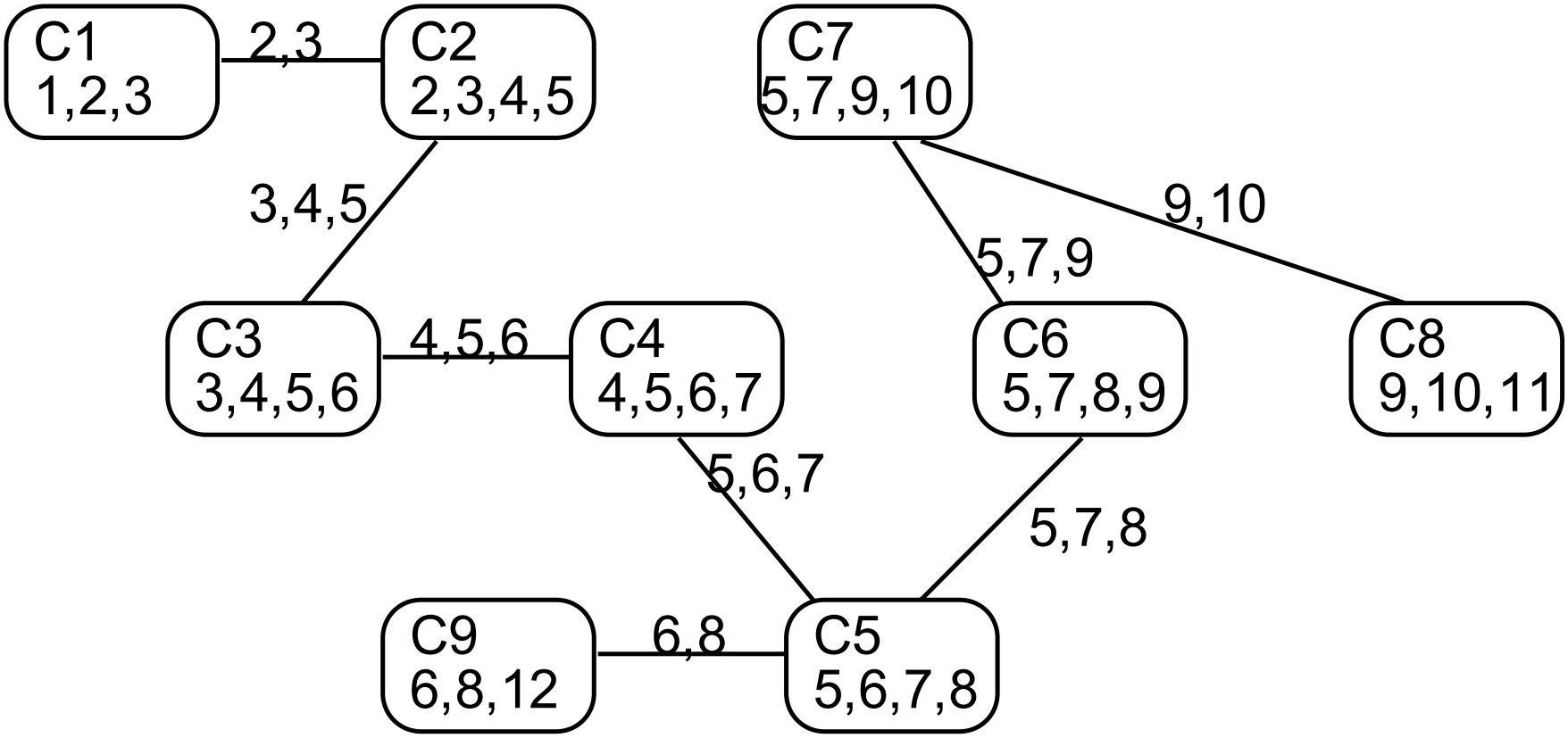
Junction Trees

- A junction tree is a subgraph of the clique graph that (1) is a tree, (2) contains all the nodes of the clique graph, and (3) 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.

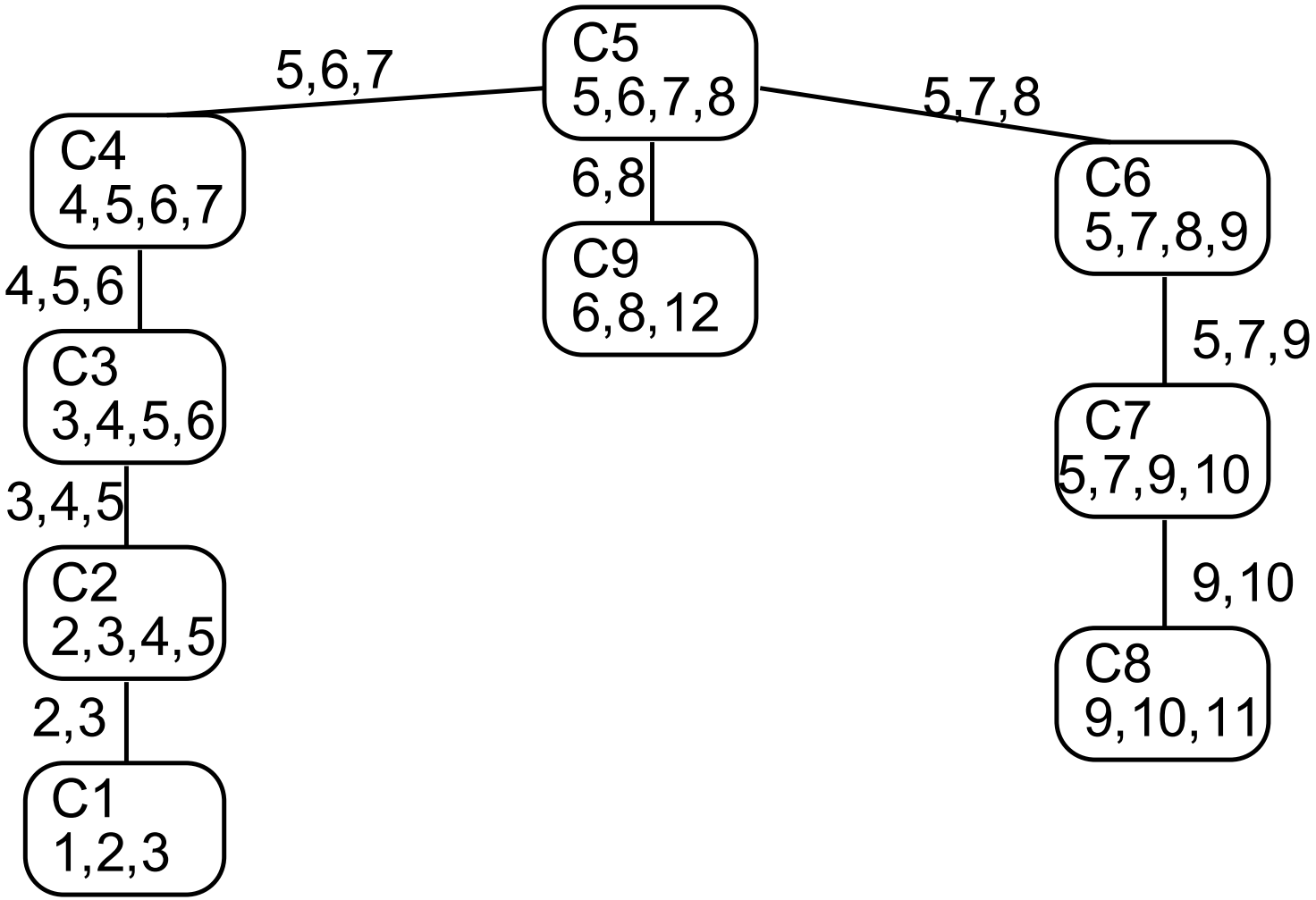
Clique Graph to Junction Tree

- We can perform exact inference efficiently on a junction tree (although CPTs may be large). But can we always build a junction tree? If so, how?
- Let the weight of an edge in the clique graph be the cardinality of the separator. Than any maximum weight spanning tree is a junction tree (Jensen & Jensen 1994).

Step 5: Build the Junction Tree



Step 6: Choose a Root



Step 7: Populate Clique Nodes

- 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).

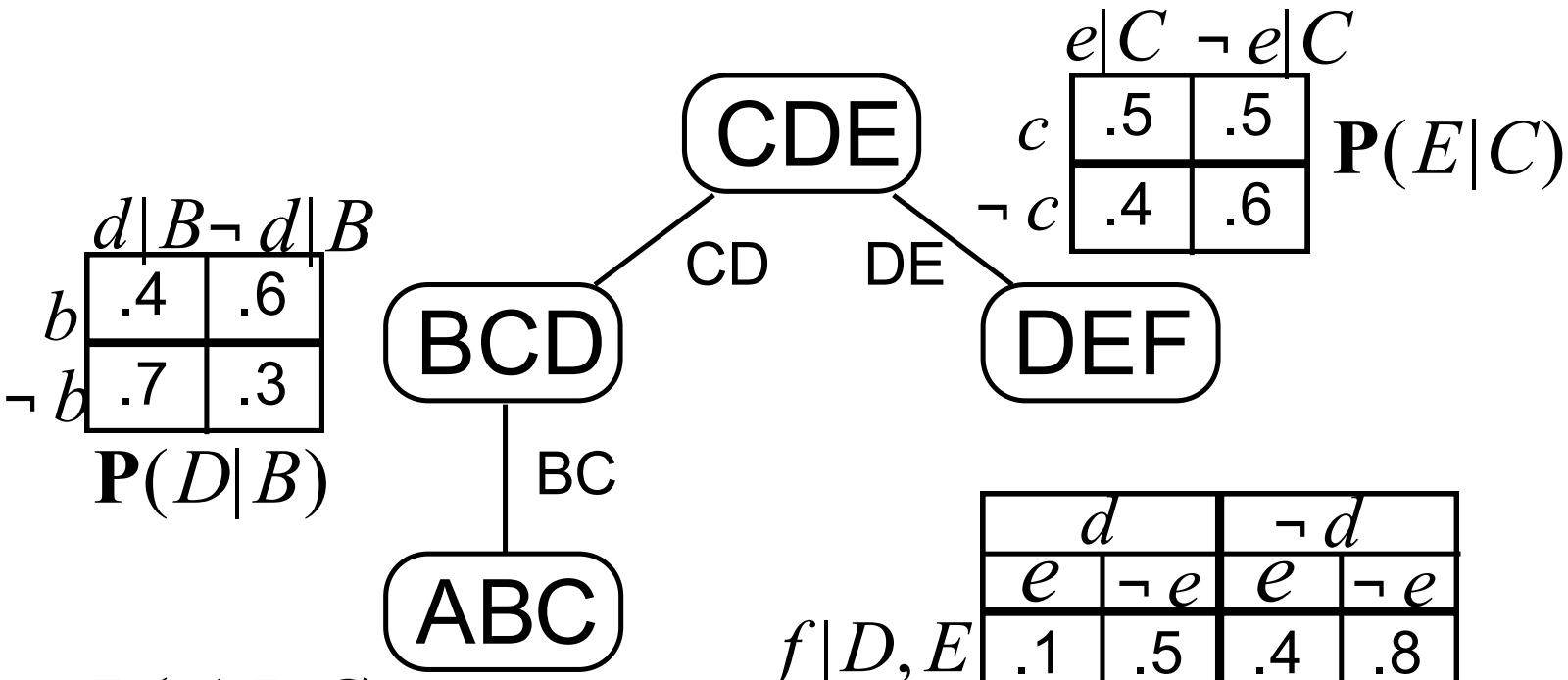
Better Triangulation Algorithm Specifically for Bayes Nets, Based on Variable Elimination

- Repeat until no nodes remain:
  - If the graph has a simplicial node, eliminate it (consider it "processed" and remove it together with all its edges).
  - Otherwise, find the node whose elimination would give the smallest potential possible. Eliminate that node, and note the need for a "fill-in" edge between any two non-adjacent nodes in the resulting potential.
- Add the "fill-in" edges to the original graph.

Find Cliques while Triangulating (or in triangulated graph)

- *While executing the previous algorithm: for each simplicial node, record that node with all its neighbors as a possible clique. (Then remove that node and its edges as before.)*
- *After recording all possible cliques, throw out any one that is a subset of another.*
- The remaining sets are the cliques in the triangulated graph.
- $O(n^3)$, guaranteed correct only if graph is triangulated.

Choose Root, Assign CPTs



	$e \mid C$	$\neg e \mid C$
c	.5	.5
$\neg c$	.4	.6

$\mathbf{P}(E \mid C)$

	$d \mid B$	$\neg d \mid B$
b	.4	.6
$\neg b$	.7	.3

$\mathbf{P}(D \mid B)$

	d		$\neg d$	
	e	$\neg e$	e	$\neg e$
$f \mid D, E$	.1	.5	.4	.8
$\neg f \mid D, E$	.9	.5	.6	.2

$\mathbf{P}(F \mid D, E)$

$\mathbf{P}(A, B, C)$

	b		$\neg b$	
	c	$\neg c$	c	$\neg c$
a	.007	.003	.063	.027
$\neg a$	.162	.648	.018	.072

Junction Tree Inference Algorithm

- ***Incorporate Evidence:*** For each evidence variable, go to *one* table that includes that variable. Set to *0* all entries in that table that disagree with the evidence.
- ***Upward Step:*** For each leaf in the junction tree, send a message to its parent. The message is the marginal of its table, ...

J.T. Inference (Continued)

- (Upward Step continued)… summing out any variable not in the separator. When a parent receives a message from a child, it multiplies its table by the message table to obtain its new table. When a parent receives messages from all its children, it repeats the process (acts as a leaf). This process continues until the root receives messages from all its children.

J.T. Inference (Continued)

- ***Downward Step:*** (Roughly reverses the upward process, starting at the root.) For each child, the root sends a message to that child. More specifically, the root divides its current table by the message received from that child, marginalizes the resulting table to the separator, and sends the result of this marginalization to the child. When a ...

J.T. Inference (Continued)

- (Downward Step continued)… child receives a message from its parent, multiplying this message by the child's current table will yield the joint distribution over the child's variables (if the child does not already have it). The process repeats (the child acts as root) and continues until all leaves receive messages from their parents.

Dividing Potentials

- Domain of divisor must be subset of domain of dividend.
- Quotient initially has domain of dividend.
- For each cell of quotient, find corresponding cell (one whose variables have same settings) in dividend and divisor. Divide value in dividend by that in divisor to get the result for quotient cell.
- For each variable that "doesn't matter," remove variable from domain of quotient. A variable doesn't matter iff: for any setting of the other variables, the cells resulting from different settings of this variable all have the same values.

Example

	b	¬b
a	X_1	X_2
¬a	X_3	X_4

(1)

×

	c	¬c
a	Y_1	Y_2
¬a	Y_3	Y_4

(2)

=

	c		¬c	
	b	¬b	b	¬b
a	$X_1 Y_1$	$X_2 Y_1$	$X_1 Y_2$	$X_2 Y_2$
¬a	$X_3 Y_3$	$X_4 Y_3$	$X_3 Y_4$	$X_4 Y_4$

(3)

(3)/(1)

	c		¬c	
	b	¬b	b	¬b
a	Y_1	Y_1	Y_2	Y_2
¬a	Y_3	Y_3	Y_4	Y_4

b doesn't matter

	c	¬c
a	Y_1	Y_2
¬a	Y_3	Y_4

(3)/(2)

	c		¬c	
	b	¬b	b	¬b
a	X_1	X_2	X_1	X_2
¬a	X_3	X_4	X_3	X_4

c doesn't matter

	b	¬b
a	X_1	X_2
¬a	X_3	X_4

Incorporating Evidence

- For any evidence $V = v$:

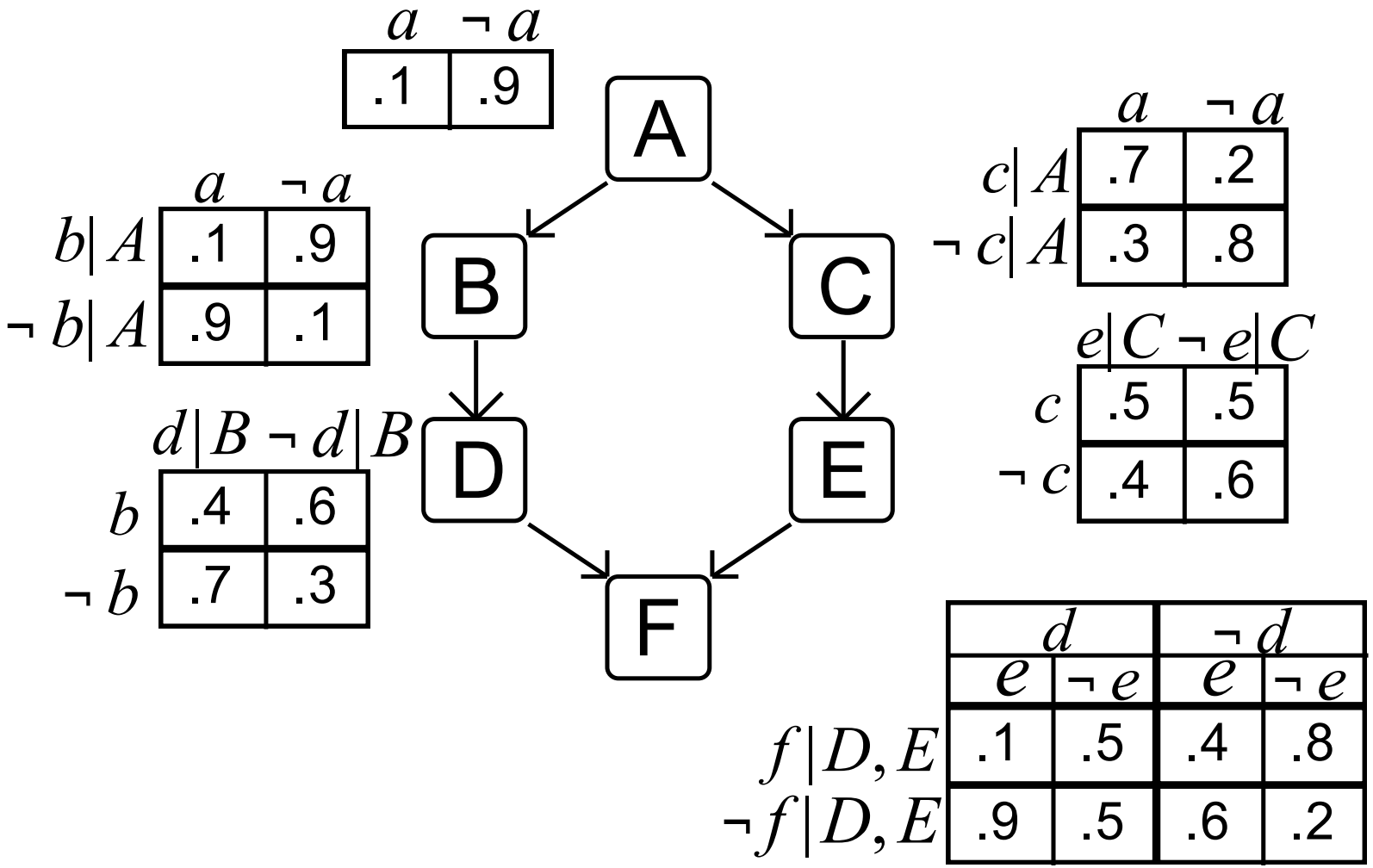
  Find <u>any</u> table in junction tree involving V and multiply it by a table with the same domain such that:
  - * every cell with $V=v$ has value 1.0
  - * every cell with $V \neq v$ has value 0.0

- After incorporating all evidence, perform upward and downward pass.

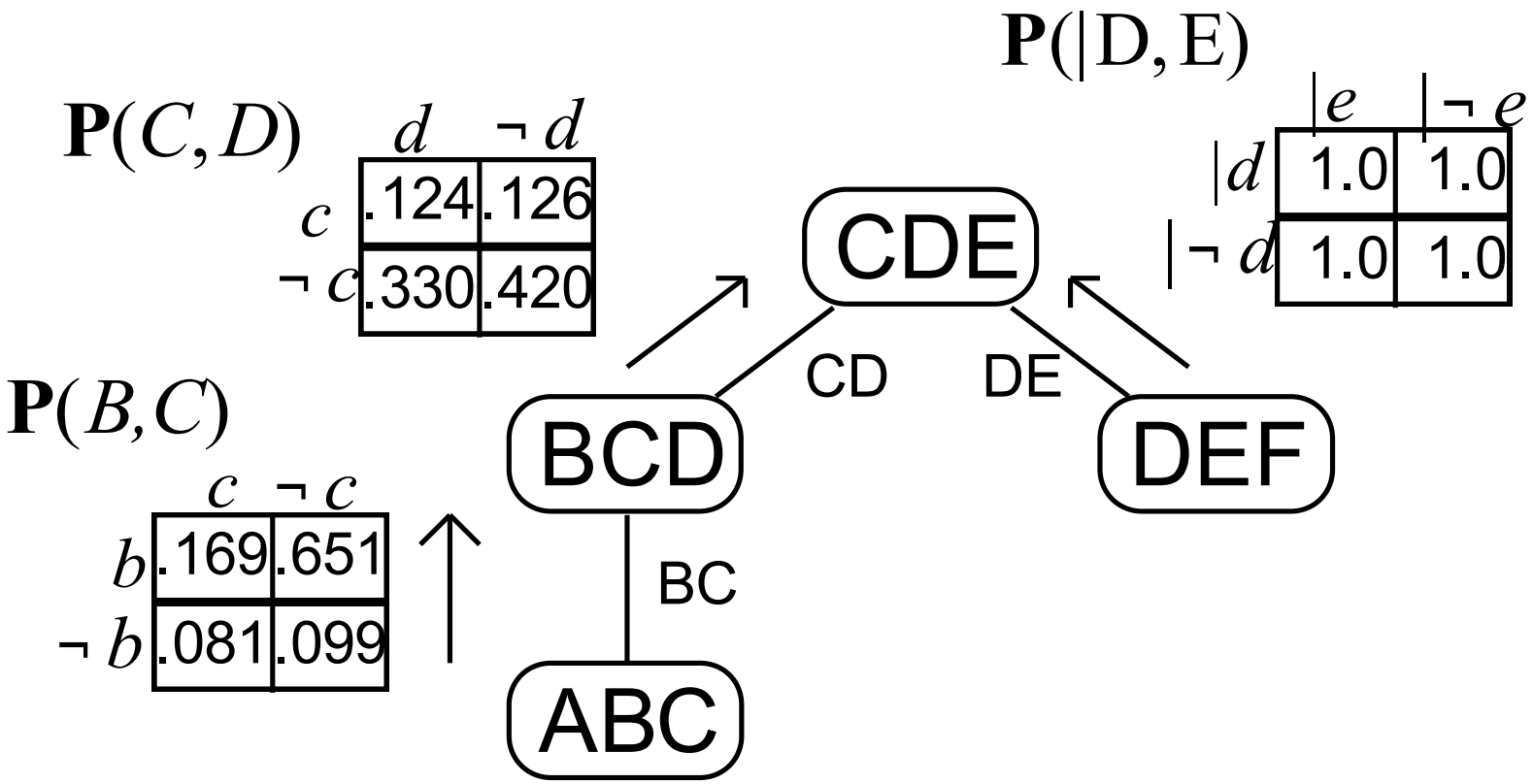
One Catch for Division

- At times we may find ourselves needing to divide by *0*.
- We can verify that whenever this occurs, we are dividing *0* by *0*.
- We simply adopt the convention that for this special case, the result will be *0* rather than *undefined*.

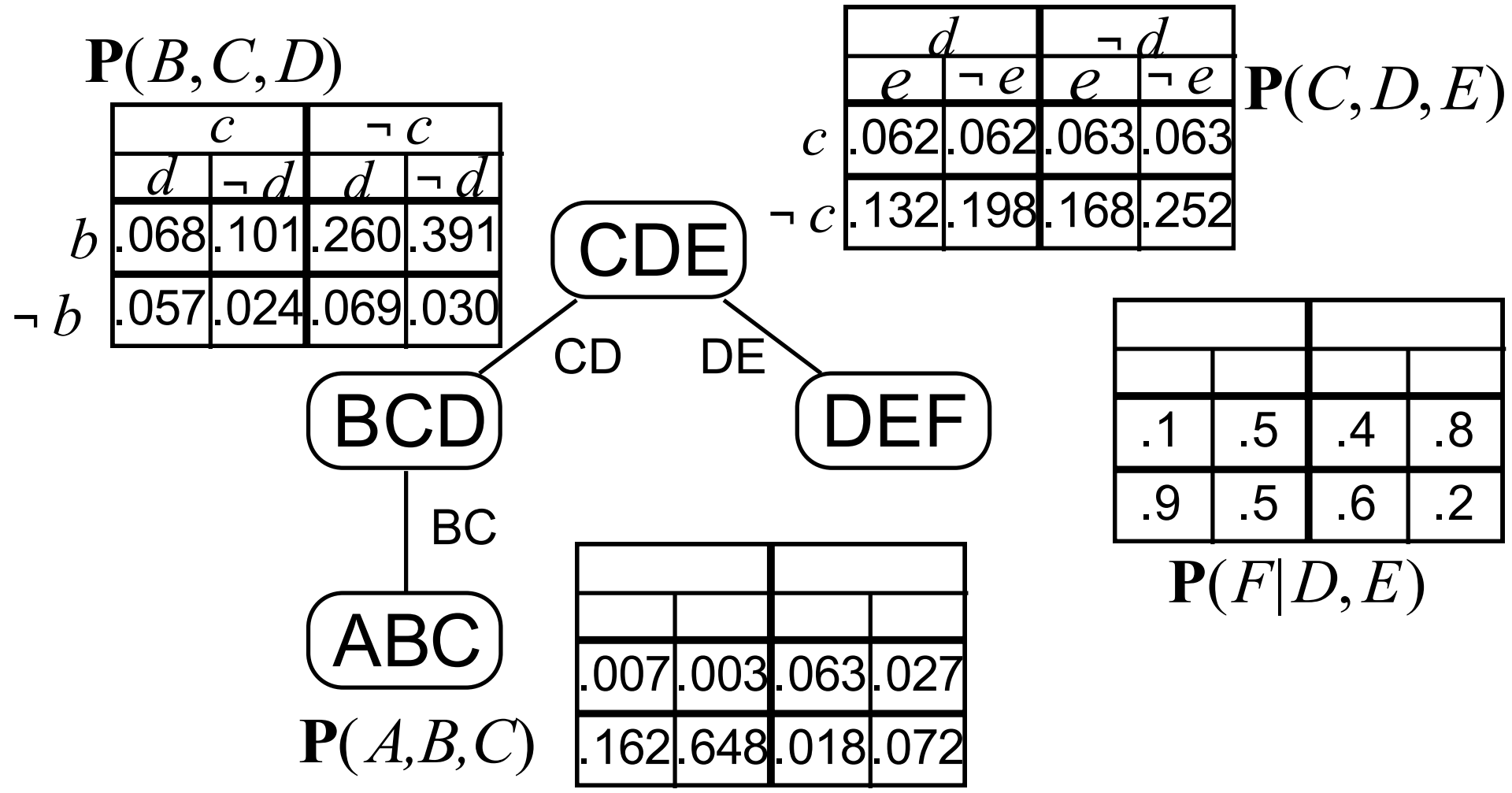
Build Junction Tree for BN Below



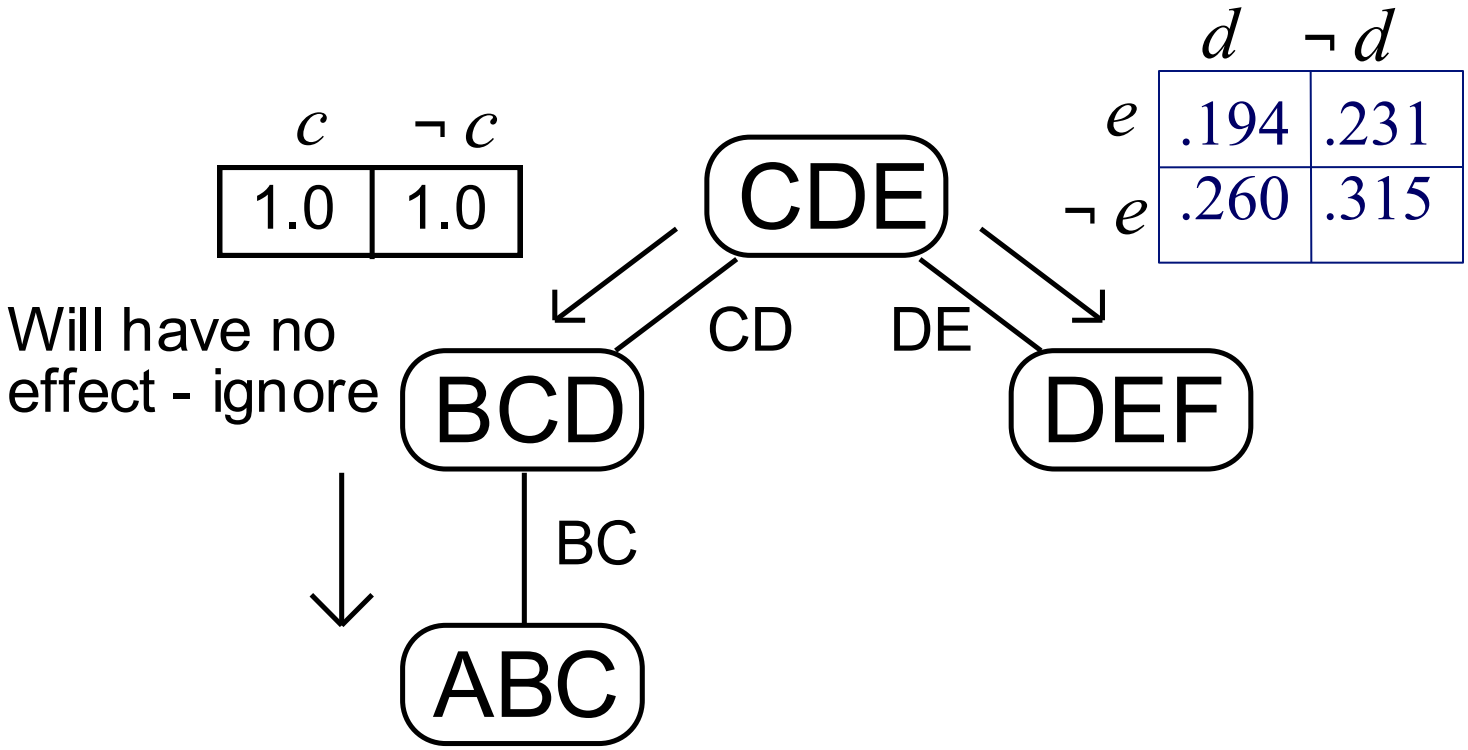
Inference Example (assume no evidence): Going Up



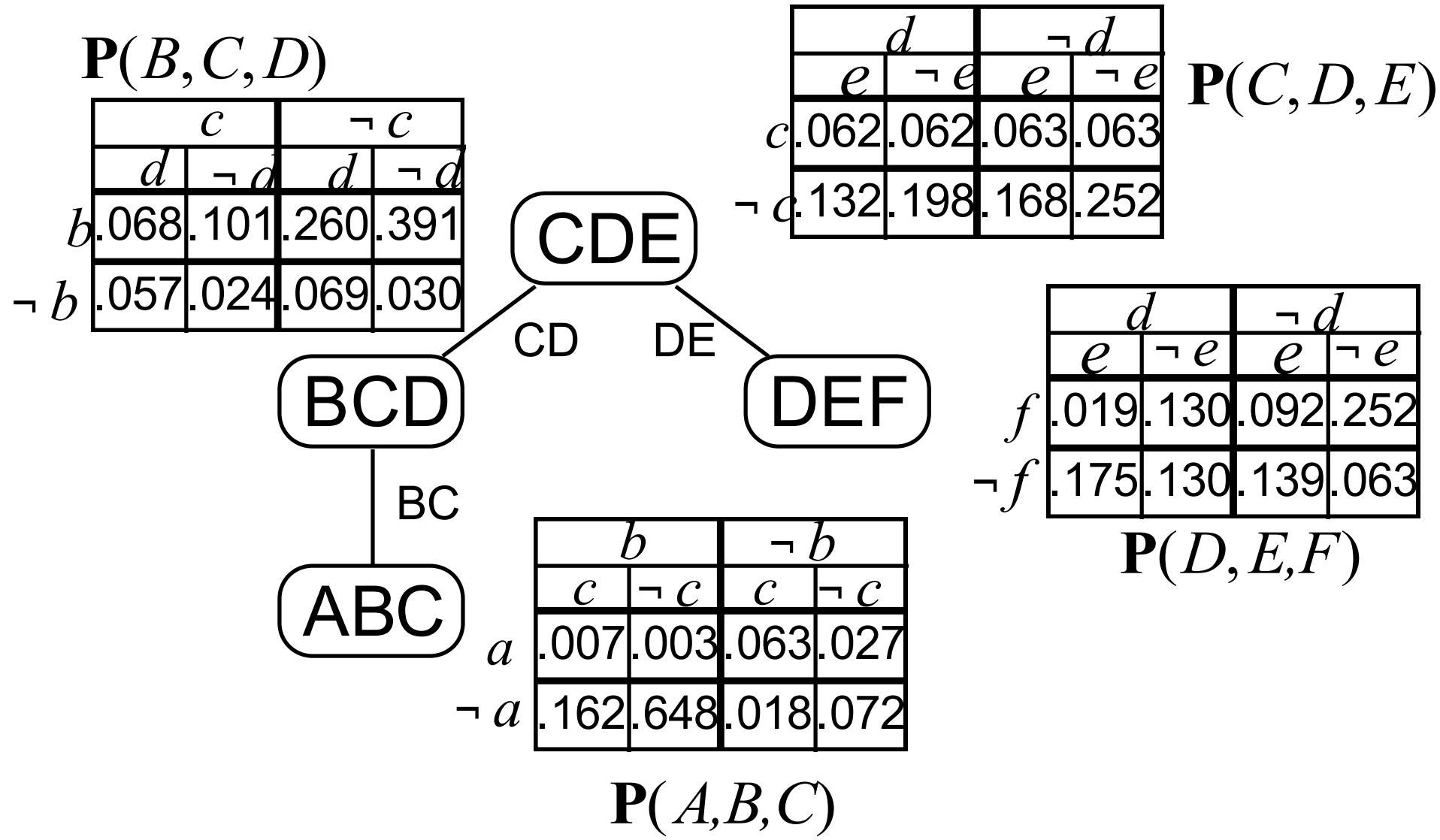
Status After Upward Pass



Going Back Down



Status After Downward Pass



Answering Queries: Final Step

- Having built the junction tree, we now can ask about any variable. We find the clique node containing that variable and sum out the other variables to obtain our answer.
- If given new evidence, we must repeat the Upward-Downward process.
- A junction tree can be thought of as storing the subjoints computed during elimination.

Junction Tree Savings

- Need to build the junction tree structure only once ever.
- Need to repeat upward and downward passes only when new evidence is received.

Significance of Junction Trees

- "…only well-understood, efficient, provably correct method for concurrently computing multiple queries (AI Mag' 99)."
- As a result, they are the most widely-used and well-known method of inference in Bayes Nets, although…
- Junction trees soon may be overtaken by approximate inference using MCMC.

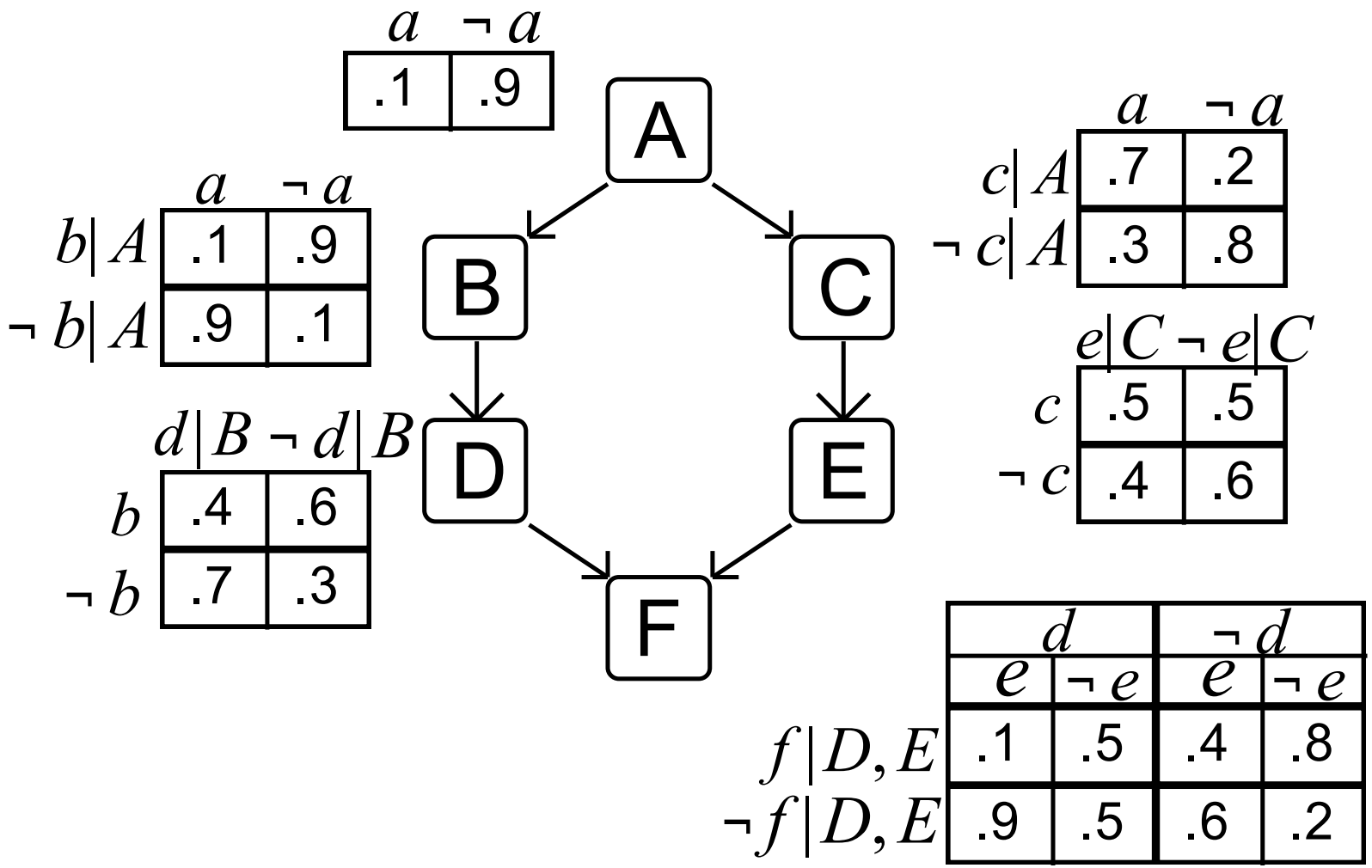
The Link Between Junction Trees and Variable Elimination

- To eliminate a variable at any step, we combine all remaining distributions (tables) indexed on (involving) that variable.
- A node in the junction tree corresponds to the variables in one of the tables created during variable elimination (the other variables required to remove a variable).
- An arc in the junction tree shows the flow of data in the elimination computation.

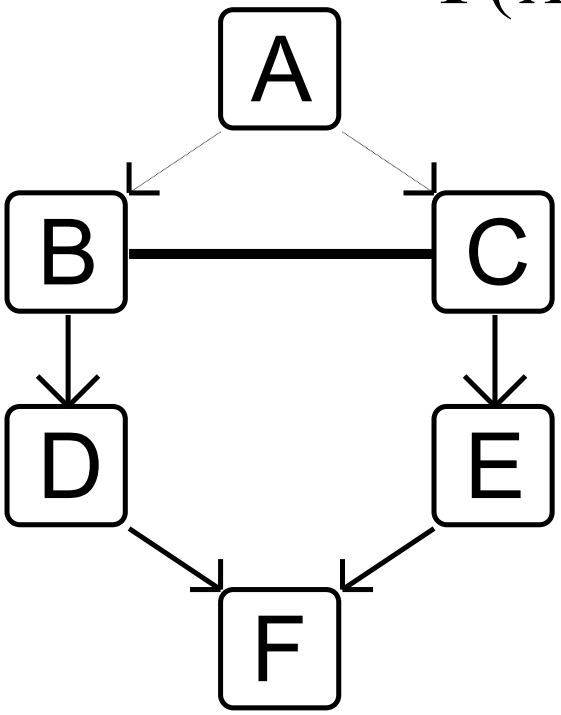
Junction Trees/Variable Elim.

- We can use different orderings in variable elimination -- affects efficiency.
- Each ordering corresponds to a junction tree.
- Just as some elimination orderings are more efficient than others, some junction trees are better than others. (Recall our mention of heuristics for triangulation.)

Recall Variable Elimination Example



First Eliminated Variable *A*



$$\mathbf{P}(A,B,C) = \mathbf{P}(B|A)\,\mathbf{P}(C|A)\,\mathbf{P}(A)$$

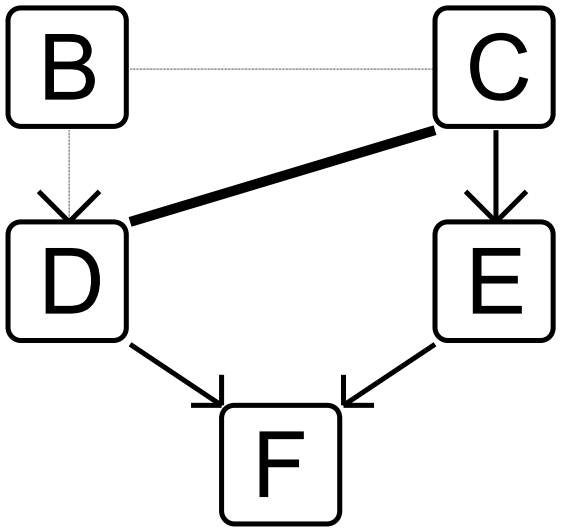
	b		$\neg b$	
	c	$\neg c$	c	$\neg c$
a	.007	.003	.063	.027
$\neg a$	.162	.648	.018	.072

$$\downarrow \sum_A$$

$\mathbf{P}(B,C)$

	c	$\neg c$
b	.169	.651
$\neg b$	.081	.099

Next Eliminated Variable B



$$\mathbf{P}(B,C,D) = \mathbf{P}(D|B)\,\mathbf{P}(B,C)$$

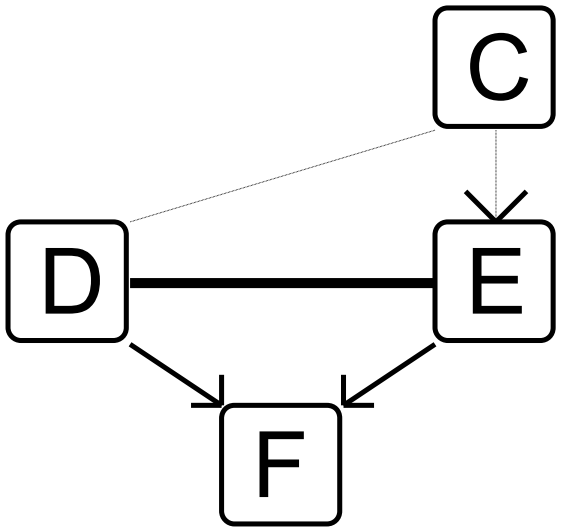
	c		$\neg c$	
	d	$\neg d$	d	$\neg d$
b	.068	.101	.260	.391
$\neg b$	.057	.024	.069	.030

$$\downarrow \sum_B$$

$\mathbf{P}(C,D)$

	d	$\neg d$
c	.124	.126
$\neg c$	.330	.420

Next Eliminated Variable C



$$\mathbf{P}(C,D,E) = \mathbf{P}(E|C)\,\mathbf{P}(C,D)$$

	d		$\neg d$	
	e	$\neg e$	e	$\neg e$
c	.062	.062	.063	.063
$\neg c$	.132	.198	.168	.252

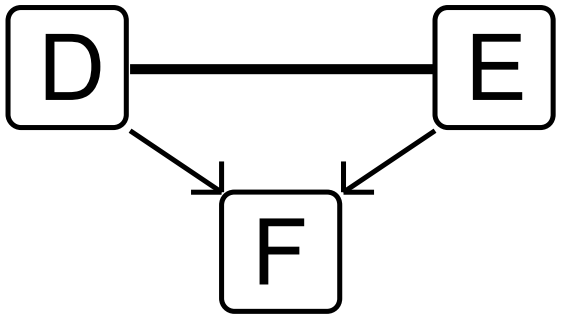
$$\downarrow \sum_C$$

$\mathbf{P}(D,E)$

	e	$\neg e$
d	.194	.260
$\neg d$	.231	.315

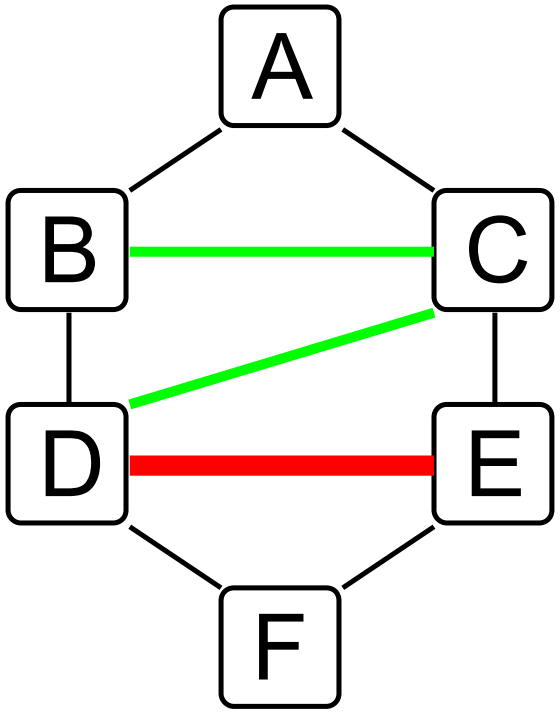
Left us with **P**(D,E,F)

$$\mathbf{P}(D,E,F) = \mathbf{P}(F|D,E)\,\mathbf{P}(D,E)$$



	d		$\neg d$	
	e	$\neg e$	e	$\neg e$
f	.019	.130	.092	.252
$\neg f$	.175	.130	.139	.063

Corresponding Moralized, Triangulated Graph...



Notice how the cliques correspond to the 3-variable joint tables we created during the variable elimination steps. We could replace the CD arc by a BE arc; we could eliminate C before B.

Approximate (Monte Carlo) Inference in Bayes Nets

- Basic idea: Let's repeatedly sample according to the distribution represented by the Bayes Net. If in 400/1000 draws, the variable *X* is *true*, then we estimate that the probability *X* is *true* is 0.4.
- To sample according to Bayes Net, just set the variables one at a time using a total ordering consistent with the partial...

Monte Carlo (continued)

- (Samping continued)… ordering represented by the underlying DAG of the Net. In this way, when we wish to draw the value for *X* we already have the values of its parents, so we can find the probabilities to use from the CPT for *X*.
- This approach is simple to implement using a pseudorandom number generator.

So it seems we're done, right?

- Wrong: what if we take into account evidence (observed values for variables)?
- If the evidence happens to be in the "top" nodes of the network (nodes with no parents), we're still fine. Otherwise...
- No efficient general method exists for sampling according to the new distribution based on the evidence. (There are inefficient ways, e.g., compute full joint.)

Rejection Sampling

- One natural option for sampling with evidence is to use our original sampling approach, and just throw out (*reject*) any setting that does not agree with the evidence. This is *rejection sampling*.
- Problem: if evidence involves many variables, most of our draws will be rejected (few will agree with the evidence).

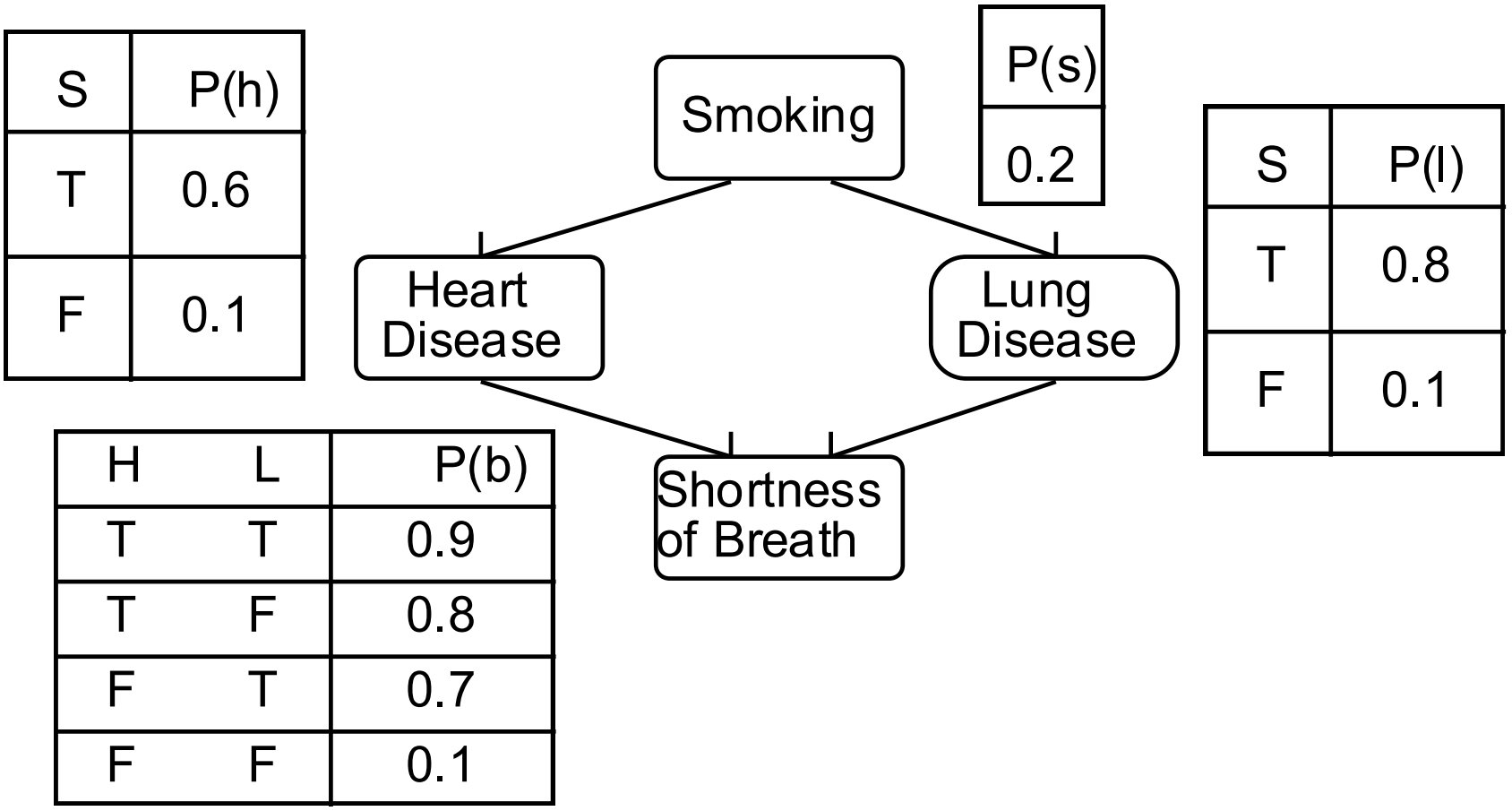
Likelihood Weighting

- Another approach is to set the evidence variables, sample the others with the original Monte Carlo approach, and then correct for improbable combinations by *weighting* each setting by its probability.
- Disadvantage: with many evidence variables, probabilities become vanishingly small. We don't sample the more probable events very thoroughly.

Markov Chain Monte Carlo

- Key idea: give up on *independence* in sampling.
- Generate next setting probabilistically based on current setting (Markov chain).
- Metropolis-Hastings Algorithm for the general case, Gibbs Sampler for Bayes Nets in particular. Key property: *detailed balance* yields *stationary distribution*.

Gibbs Sampling by Example



Gibbs Sampling Example (Continued)

- Let our query be **P**(*HeartDisease* | *smoking, shortnessOfBreath*). That is, we know we've been smoking (*Smoking=True)* and we know we're experiencing shortness of breath (*ShortnessOfBreath=True*), and we wish the know the probability that we have heart disease.
- Might as well keep a tally for *LungDisease* while we're at it.

Other Decisions

- Let's assume we use an off-the-shelf pseudorandom number generator for the range [0..1].
- We can loop through the non-evidence variables in a pre-established order or randomly, uniformly. Let's go with the latter. Tally at each step. (If the former, we could tally at each step or each iteration.)

Other Decisions (Continued)

- One chain or many: let's go with one.
- Length or burn-in: ordinarily 500-1000, but let's go with 2 (don't tally for original setting or setting after first step).
- Initial values: let's say all *True*. Note that *Smoking* and *ShortnessOfBreath* must be initialized to *True*, since this is our evidence. The initial settings for non-evidence variables are arbitrary.

Other Decisions (Continued)

- Use of random numbers in selecting a variable to draw. We have only two non-evidence variables: *HeartDisease* and *LungDisease*. Let's adopt a policy that a random number greater than 0.5 leads us to draw a value for *LungDisease*, and a random number of 0.5 or less leads us to draw for *HeartDisease*.

Other Decisions (Continued)

- Use of random numbers in selecting values of variables. Since all our variables are Boolean, our distributions will be over the values <*True*,False> and will have the form <P(*True*),1-P(*True*)>. If our random number is less than or equal to P(*True*), then we will set the variable to *True*, and otherwise we will set it to *False*.

A Final Supposition for our Example

- Having made all our choices, the only other factor that will affect the activity of the Gibbs Sampling algorithm is the sequence of random numbers that we draw.
- Let's suppose our sequence of random numbers begins 0.154, 0.383, 0.938, 0.813, 0.273, 0.739, 0.049, 0.233, 0.743, 0.932, 0.478, 0.832, …

Round 1

- Our first random number is 0.154, so we will draw a value for *HeartDisease*.
- To draw the value, we must first determine the distribution for *HeartDisease* given its Markov Blanket.
- First, we compute a value for *True*. We multiply P(*heartDisease*|*smoking*) by P(*shortnessOfBreath* | *heartDisease, lungDisease*). Notice we take *LungDisease*

Round 1 (Continued)

- (Continued)… to be *True* because that is its current setting. (We use the current settings of all variables in the Markov Blanket.) This product is (0.6)(0.9) = 0.54.
- Next we repeat the process for *HeartDisease=False*. We multiply the probability that *HeartDisease* is *False* given *smoking* by the probability of *shortnessOfBreath* given *HeartDisease* is

Round 1 (Continued)

- (Continued)… *False* and *LungDisease* is *True*. The resulting product is (0.4)(0.7) = 0.28.
- We now normalize <0.54,0.28> to get the probability distribution <0.66,0.34>. Hence we will set *HeartDisease* to *True* if and only if our random number is at most 0.66. It is 0.383, so *HeartDisease* remains *True*.

Round 2

- Our next random number is 0.938, so we next will draw a value for *LungDisease* given the current settings for the other variables.
- To obtain a value for *LungDisease*=*True*, we multiply P(*lungDisease* | *smoking*) by P(*shortnessOfBreath* | *heartDisease, lungDisease*). (Recall that *True* is our current setting for *HeartDisease* and *True*

Round 2 (Continued)

- (Continued)… is our candidate setting for *LungDisease*. This product is (0.8)(0.9) = 0.72.
- Similarly, for *LungDisease=False*, we multiply P(*LungDisease=False* | *smoking*) by P(*shortnessOfBreath* | *heartDisease, LungDisease=False*). This product is (0.2)(0.8) = 0.16.

Round 2 (Continued)

- Normalizing <0.72, 0.16> we get the distribution <0.82, 0.18>.
- Our next random number is 0.813, so we (barely) keep *LungDisease* set to *True.*
- This is the first round after our burn-in, so we record the frequencies. We now have counts of 0 for *HeartDisease* and *LungDisease* set to *False*, and counts of 1 for each of these set to *True*.

Round 3

- Our next random number is 0.273, so we draw a value for *HeartDisease* next.
- Because all the variables have the same value as the last time we drew for *HeartDisease*, the distribution is the same: <0.66, 0.34>. Our next random number is 0.739, so we set *HeartDisease* to *False.*

Round 3 (Continued)

- Updating our tallies, we have counts of: 1 for *HeartDisease=False,* 1 for *HeartDisease=True*, 0 for *LungDisease=False*, and 2 for *LungDisease=True.*

Round 4

- The next random number is 0.049. Therefore we draw a value for *HeartDisease* again. Because all other variables are unchanged, and we consider both values of *HeartDisease*, once again the distribution is <0.66, 0.34>. Our next random number is 0.233, so we reset *HeartDisease* to *True*.

Round 4 (Continued)

- Our new counts are as follows: 2 for *HeartDisease=True,* 1 for *HeartDisease=False,* 3 for *LungDisease=True*, and 0 for *LungDisease=False.*

Round 5

- Our next random number is 0.743, so we next draw a value for *LungDisease*.
- The values for all other variables are as they were the first time we drew a value for *LungDisease*, so the distribution remains <0.82,0.18>. Our next random number is 0.932, so we set *LungDisease* to *False*.

Round 5 (Continued)

- Our new tallies are as follows: 3 each for *HeartDisease=True* and *LungDisease=True*, and 1 each for *HeartDisease=False* and *LungDisease=False.*

Round 6

- The next random number is 0.478, so again we sample *HeartDisease.* But since the setting for *LungDisease* has changed, we must recompute the distribution over *HeartDisease.*
- To get a value for *HeartDisease=True*, we multiply P(*heartDisease* | *smoking*) by P(*shortnessOfBreath* | *HeartDisease=True,*

Round 6 (Continued)

- (Continued)… *LungDisease=False*). This results in the product (0.6)(0.8) = 0.48.
- For *HeartDisease=False*, we multiply P(*HeartDisease=False* | *smoking*) by P(*shortnessOfBreath* | *HeartDisease=False, LungDisease=False*). The result is (0.4)(0.1) = 0.04.
- Normalizing these values to obtain a

Round 6 (Continued)

- (Continued)… probability distribution, we get <0.92, 0.08>. Our next random number is 0.832 so we choose *HeartDisease=True.*
- Our tallies now stand at 1 for *HeartDisease=False,* 4 for *HeartDisease=True,* 2 for *LungDisease=False,* and 3 for *LungDisease=True.*

Final Results

- Of course, we have not run the Markov chain nearly long enough to expect an accurate estimate. Nevertheless, let's ask what the answer is to our query at this point.
- We assign a probability of 4/5 or 0.8 to *heartDisease.*
- We also might ask about *lungDisease*, to which we assign 3/5 or 0.6.

Markov Chain

- A Markov chain includes
  - A set of states
  - A set of associated transition probabilities
    - For every pair of states *s* and *s'* (not necessarily distinct) we have an associated transition probability T(*s*➔*s'*) of moving from state *s* to state *s'*
    - For any time *t*, T(*s*➔*s'*) is the probability of the Markov process being in state *s'* at time *t+1* given that it is in state *s* at time *t*

Some Properties of Markov Chains

- **Irreducible** chain: can get from any state to any other eventually (non-zero probability)
- **Periodic** state: state *i* is periodic with period *k* if all returns to *i* must occur in multiples of *k*
- **Ergodic** chain: irreducible and has an aperiodic state. Implies all states are aperiodic, so chain is aperiodic.
- Finite state space: can represent chain as matrix of transition probabilities… then *ergodic* = *regular*…
- **Regular** chain: some power of chain has only positive elements
- **Reversible** chain: satisfies detailed balance (**later**)

Sufficient Condition for Regularity

- A Markov chain is regular if the following properties both hold:

  1. For any pair of states *s*, *s'* that each have nonzero probability there exists some path from *s* to *s'* with nonzero probability

  2. For all *s* with nonzero probability, the "self loop" probability T(*s*➔*s*) is nonzero

- Gibbs sampling is regular if no zeroes in CPTs

Notation: Probabilities

- $\pi_t(\boldsymbol{y})$ = probability of being in state ***y*** at time *t*
- Transition function T(***y*** → ***y'***) = probability of moving from state ***y*** to state ***y'***

Bayesian Network Probabilities

- We use P to denote probabilities according to our Bayesian network, conditioned on the evidence
  - For example, $P(y_i' | \boldsymbol{u_i})$ is the probability that random variable Y_i has value y_i' given that $Y_j = y_j$ for all j not equal to i

Assumption: CPTs nonzero

- We will assume that all probabilities in all conditional probability tables are nonzero
- So, for any ***y***, $P(y) = \prod_{i=1}^{n} P(y_i \mid y_j \forall j \in Parents[i]) > 0$
- So, for any event *S*, $P(S) = \sum_{y \in S} P(y) > 0$
- So, for any events S_1 and S_2, $P(S_1 \mid S_2) = \frac{P(S_1 \cap S_2)}{P(S_2)} > 0$

How π Changes with Time in a Markov Chain

- $\pi_{t+1}(\boldsymbol{y}') = \sum_{\mathbf{y}} \pi_t(\mathbf{y}) T(\mathbf{y} \rightarrow \mathbf{y}')$
- A distribution π_t is stationary if $\pi_t = \pi_{t+1}$, that is, for all $\boldsymbol{y}$, $\pi_t(\boldsymbol{y}) = \pi_{t+1}(\boldsymbol{y})$

Detailed Balance

- A Markov chain satisfies detailed balance if there exists a unique distribution π such that for all states $\boldsymbol{y}$, $\boldsymbol{y'}$,

$$\pi(\boldsymbol{y})T(\boldsymbol{y} \rightarrow \boldsymbol{y'}) = \pi(\boldsymbol{y'})T(\boldsymbol{y'} \rightarrow \boldsymbol{y})$$

- If a regular Markov chain satisfies detailed balance with distribution π, then there exists t such that for any initial distribution π_0, $\pi_t = \pi$
- Detailed balance with regularity implies convergence to unique stationary distribution

Gibbs Sampler satisfies Detailed Balance

Claim: A Gibbs sampler Markov chain defined by a Bayesian network with all CPT entries nonzero satisfies detailed balance with probability distribution $\pi(\boldsymbol{y})=P(\boldsymbol{y})$ for all states $\boldsymbol{y}$

Proof: First we will show that $P(\boldsymbol{y})T(\boldsymbol{y}\rightarrow\boldsymbol{y}')$ = $P(\boldsymbol{y}')T(\boldsymbol{y}'\rightarrow\boldsymbol{y})$. Then we will show that no other probability distribution π satisfies $\pi(\boldsymbol{y})T(\boldsymbol{y}\rightarrow\boldsymbol{y}')$ = $\pi(\boldsymbol{y}')T(\boldsymbol{y}'\rightarrow\boldsymbol{y})$

Using Other Samplers

- The Gibbs sampler only changes one random variable at a time
  - Slow convergence
  - High-probability states may not be reached because reaching them requires going through low-probability states

Metropolis Sampler

- Propose a transition with probability $T^{Q}(\boldsymbol{y} \rightarrow \boldsymbol{y}')$
- Accept with probability $A=\min(1, P(\boldsymbol{y}')/P(\boldsymbol{y}))$
- If for all $\boldsymbol{y}$, $\boldsymbol{y}'$ $T^{Q}(\boldsymbol{y} \rightarrow \boldsymbol{y}')=T^{Q}(\boldsymbol{y}' \rightarrow \boldsymbol{y})$ then the resulting Markov chain satisfies detailed balance

Metropolis-Hastings Sampler

- Propose a transition with probability $T^Q(\boldsymbol{y} \rightarrow \boldsymbol{y}')$
- Accept with probability

  $A=\min(1, P(\boldsymbol{y}')T^Q(\boldsymbol{y}' \rightarrow \boldsymbol{y})/P(\boldsymbol{y})T^Q(\boldsymbol{y} \rightarrow \boldsymbol{y}'))$
- Detailed balance satisfied
- Acceptance probability often easy to compute even though sampling according to P difficult

Gibbs Sampler as Instance of Metropolis-Hastings

- Proposal distribution $T^Q(\boldsymbol{u_i}, y_i \rightarrow \boldsymbol{u_i}, y_i') = P(y_i' | \boldsymbol{u_i})$
- Acceptance probability:

$$A = \min(1, \frac{P(u_i, y_i')T^Q(u_i, y_i' \rightarrow u_i, y_i)}{P(u_i, y_i)T^Q(u_i, y_i \rightarrow u_i, y_i')})$$

$$= \min(1, \frac{P(y_i' | u_i)P(u_i)P(y_i | u_i)}{P(y_i | u_i)P(u_i)P(y_i' | u_i)})$$

$$= 1$$

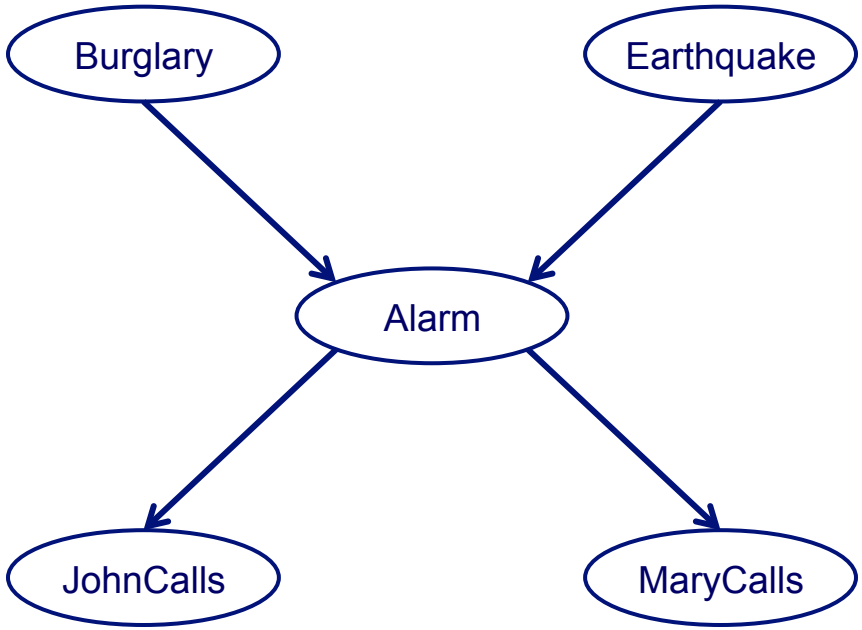
Comments on BN inference

- *inference by enumeration* is an *exact* method (i.e. it computes the exact answer to a given query)
- it requires summing over a joint distribution whose size is exponential in the number of variables
- in many cases we can do exact inference efficiently in large networks
  - variable elimination: save computation by pushing sums inward
  - junction trees: re-use work if same evidence in many queries
- in general, the Bayes net inference problem is NP-hard
- there are also methods for approximate inference – these get an answer which is “close”: belief propagation (not covered), MCMC
- in general, the approximate inference problem is NP-hard also, but approximate methods work well for many real-world problems
- Gibbs sampling and other MCMC methods have many ML applications outside of Bayes nets also

The parameter learning task

- Given: a set of training instances, the graph structure of a BN

B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	t	f	t
		…		



- Do: infer the parameters of the CPDs

The structure learning task

- Given: a set of training instances

B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	t	f	t
		…		

- Do: infer the graph structure (and perhaps the parameters of the CPDs too)

Parameter learning and maximum likelihood estimation

- *maximum likelihood estimation* (MLE)
  - given a model structure (e.g. a Bayes net graph) and a set of data D
  - set the model parameters θ to maximize $P(D \mid \theta)$

- i.e. make the data D look as likely as possible under the model $P(D \mid \theta)$

Maximum likelihood estimation

consider trying to estimate the parameter θ (probability of heads) of a biased coin from a sequence of flips

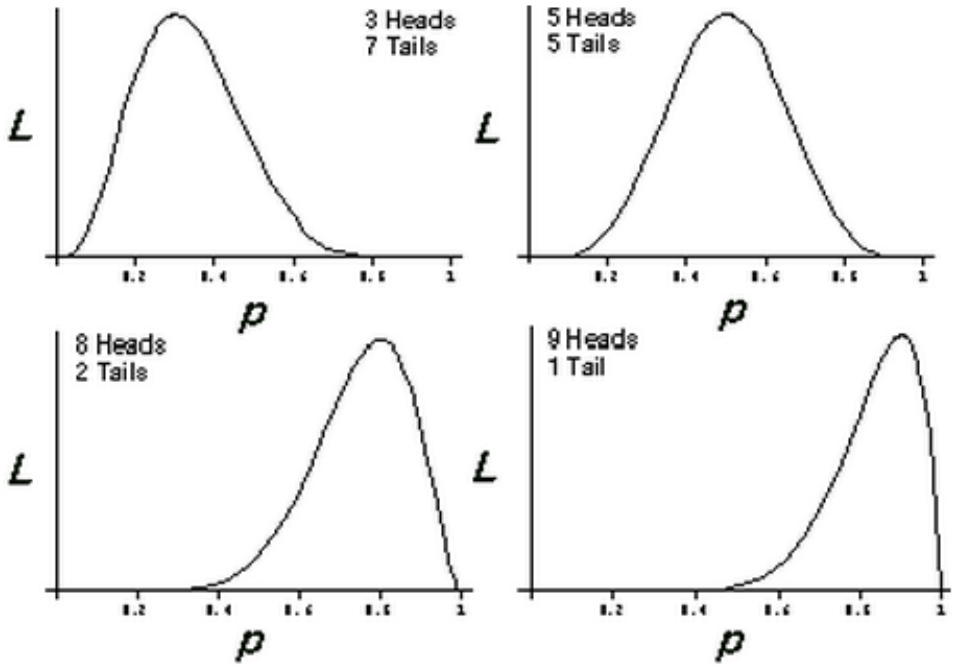
$$\boldsymbol{x} = \{1,1,1,0,1,0,0,1,0,1\}$$

the likelihood function for θ is given by:

$$L(\theta \mid x_1,\ldots,x_n) = \theta^{x_1}(1-\theta)^{1-x_1}\cdots\theta^{x_n}(1-\theta)^{1-x_n}$$

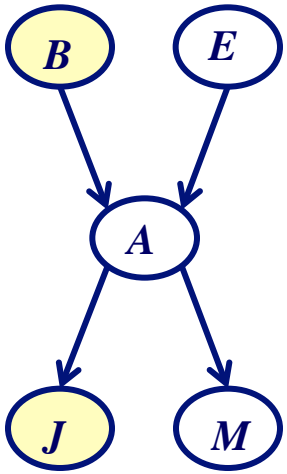
$$= \theta^{\sum x_i}(1-\theta)^{n-\sum x_i}$$

for h heads in n flips
the MLE is h/n



Maximum likelihood estimation

consider estimating the CPD parameters for B and J in the alarm network given the following data set



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
t	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{1}{8} = 0.125$$

$$P(\neg b) = \frac{7}{8} = 0.875$$

$$P(j \mid a) = \frac{3}{4} = 0.75$$

$$P(\neg j \mid a) = \frac{1}{4} = 0.25$$

$$P(j \mid \neg a) = \frac{2}{4} = 0.5$$

$$P(\neg j \mid \neg a) = \frac{2}{4} = 0.5$$

Learning structure + parameters

- number of structures is super-exponential in the number of variables
- finding optimal structure is NP-complete problem
- two common options:
  - search very restricted space of possible structures (e.g. networks with tree DAGs)
  - use heuristic search (e.g. sparse candidate)

The Chow-Liu algorithm

- learns a BN with a tree structure that maximizes the likelihood of the training data
- algorithm
  1. compute weight $I(X_i, X_j)$ of each possible edge (X_i, X_j)
  2. find maximum weight spanning tree (MST)
  3. assign edge directions in MST

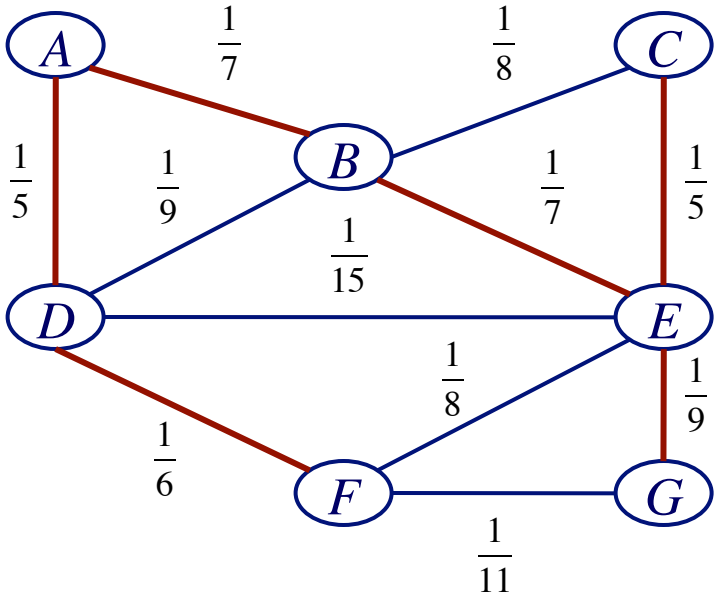
The Chow-Liu algorithm

1. use mutual information to calculate edge weights

$$I(X,Y) = \sum_{x \in \text{values}(X)} \sum_{y \in \text{values}(Y)} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

The Chow-Liu algorithm

2. find maximum weight spanning tree: a maximal-weight tree that connects all vertices in a graph



Prim's algorithm for finding an MST

given: graph with vertices V and edges E

$V_{new} \leftarrow \{ v \}$ where v is an arbitrary vertex from V
$E_{new} \leftarrow \{ \}$
repeat until $V_{new} = V$
{
    choose an edge (u, v) in E with max weight where u is in V_{new} and v is not
    add v to V_{new} and (u, v) to E_{new}
}
return V_{new} and E_{new} which represent an MST

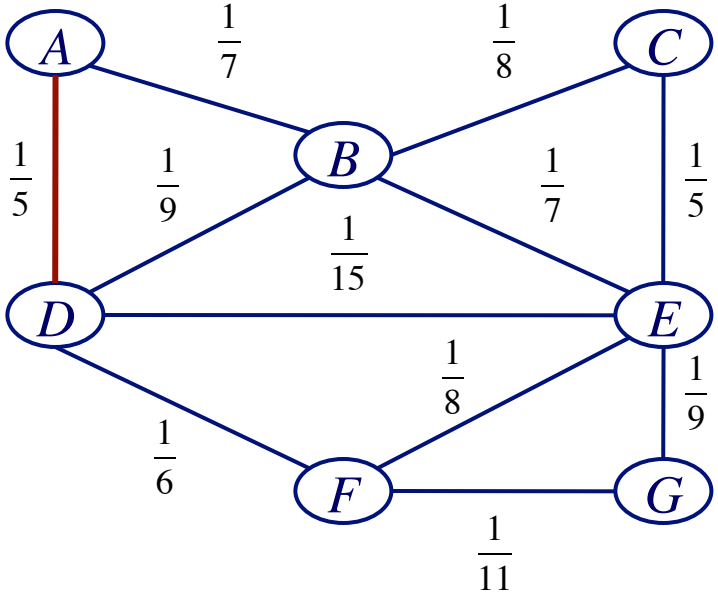
Kruskal's algorithm for finding an MST

given: graph with vertices V and edges E

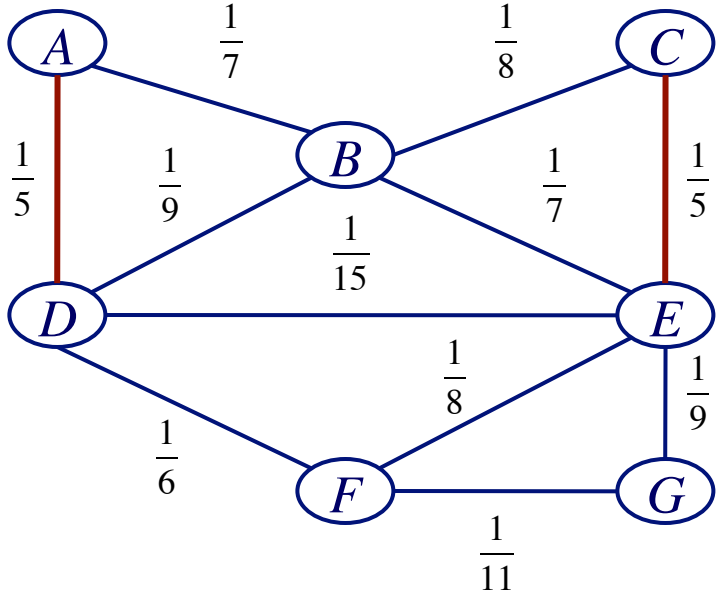
$E_{new} \leftarrow \{\ \}$
for each (u, v) in E ordered by weight (from high to low)
{
    remove (u, v) from E
    if adding (u, v) to E_{new} does not create a cycle
        add (u, v) to E_{new}
}
return V and E_{new} which represent an MST

Finding MST in Chow-Liu

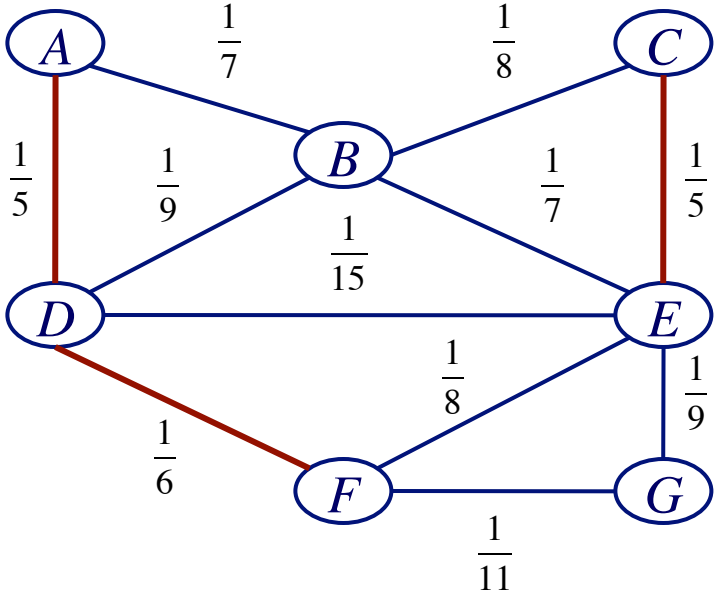
i.



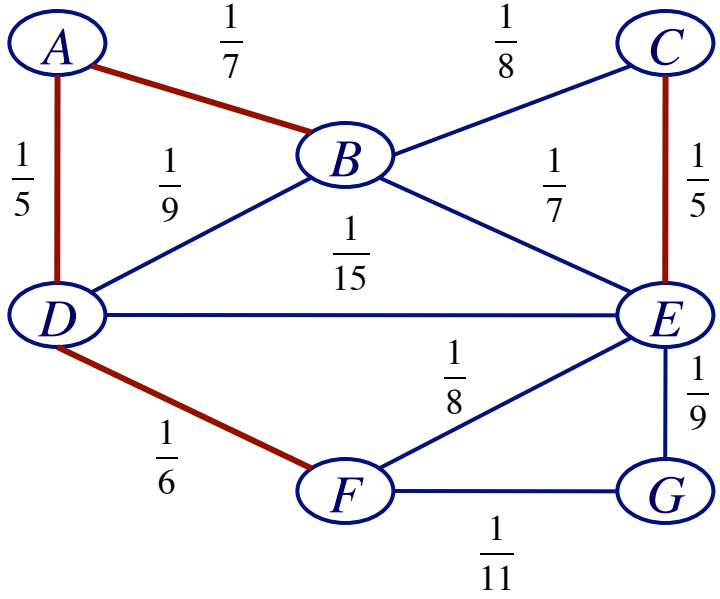
ii.



iii.

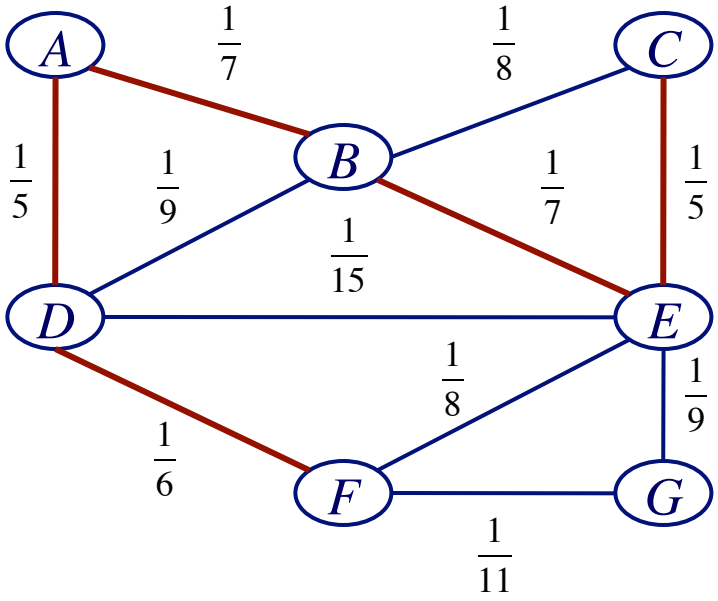


iv.

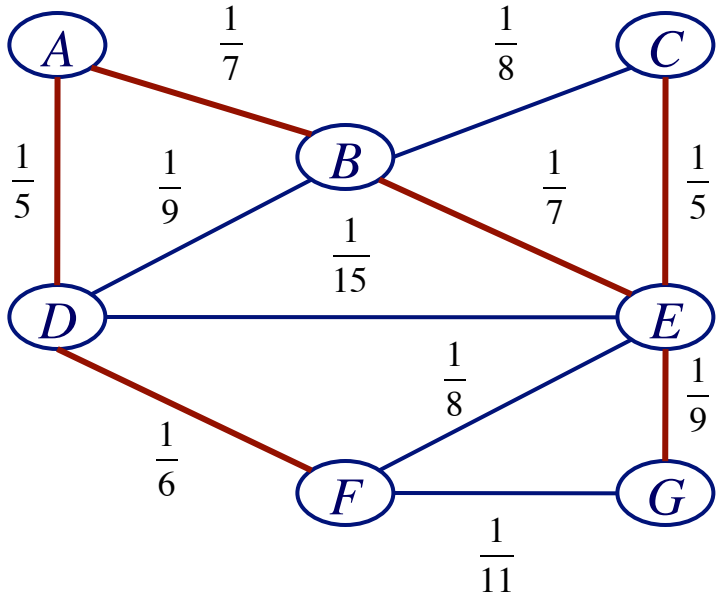


Finding MST in Chow-Liu

v.

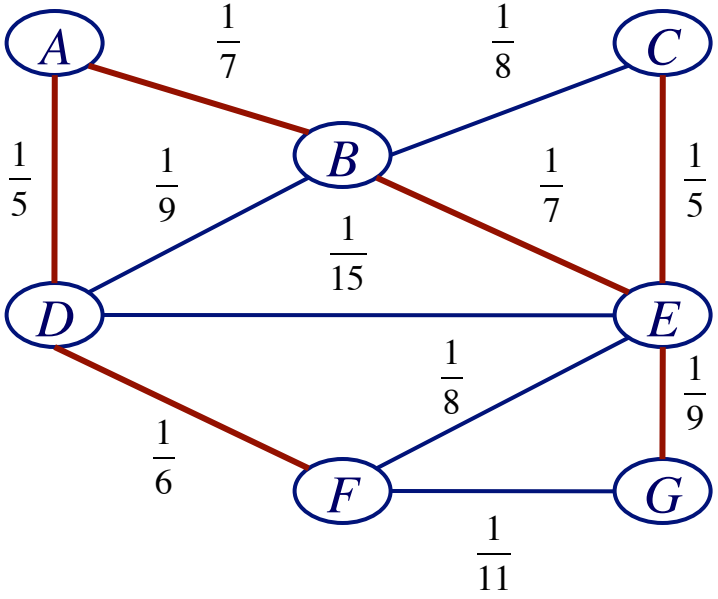


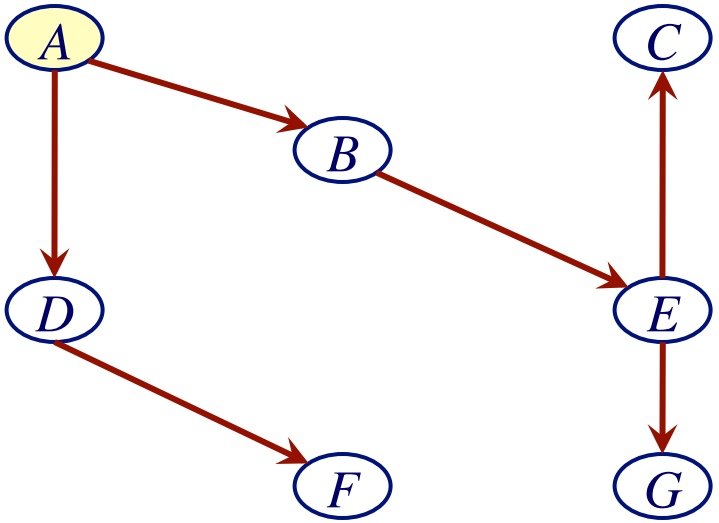
vi.



Returning directed graph in Chow-Liu

3. pick a node for the root, and assign edge directions





The Chow-Liu algorithm

- How do we know that Chow-Liu will find a tree that maximizes the data likelihood?
- Two key questions:
  - Why can we represent data likelihood as sum of $I(X;Y)$ over edges?
  - Why can we pick any direction for edges in the tree?

Why Chow-Liu maximizes likelihood (for a tree)

data likelihood given directed edges of G, best fit parameters θ_G

$$\log P(D \mid G, \theta_G) = \sum_{d \in D} \sum_i \log_2 P(x_i^{(d)} \mid Pa(X_i))$$

$$= |D| \sum_i \sum_{values(X_i, Pa(X_i))} P(X_i, Pa(X_i)) \log_2 \left(P(X_i \mid Pa(X_i)) \right)$$

(since summing over all examples is equivalent to computing average over all examples and then multiplying by total number of examples $|D|$)

$$= |D| \sum_i \sum_{values(X_i, Pa(X_i))} P(X_i, Pa(X_i)) \log_2 \left(P(X_i, Pa(X_i)) / Pa(X_i) \right)$$

$$= |D| \sum_i \sum_{values(X_i, Pa(X_i))} P(X_i, Pa(X_i)) \log_2 \left(P(X_i, Pa(X_i)) / P(X_i)(Pa(X_i)) \right) +$$
$$P(X_i, Pa(X_i)) \log_2 P(X_i)$$

$$= |D| \sum_i \left(I(X_i, Pa(X_i)) - H(X_i) \right)$$

Why Chow-Liu maximizes likelihood (for a tree)

data likelihood given directed edges

$$\log P(D \mid G, \theta_G) = \sum_{d \in D} \sum_i \log_2 P(x_i^{(d)} \mid Parents(X_i))$$

$$= |D| \sum_i \left(I(X_i, Parents(X_i)) - H(X_i) \right)$$

we're interested in finding the graph G that maximizes this

$$\arg\max_G \log P(D \mid G, \theta_G) = \arg\max_G \sum_i I(X_i, Parents(X_i))$$

if we assume a tree, one node has no parents, all others have exactly one

$$\arg\max_G \log P(D \mid G, \theta_G) = \arg\max_G \sum_{(X_i, X_j) \in \text{edges}} I(X_i, X_j)$$

edge directions don't matter for likelihood, because MI is symmetric

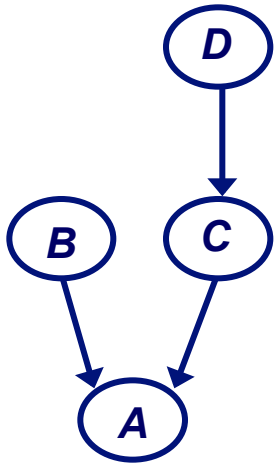
$$I(X_i, X_j) = I(X_j, X_i)$$

Heuristic search for structure learning

- each state in the search space represents a DAG Bayes net structure
- to instantiate a search approach, we need to specify
  - state transition operators
  - scoring function for states
  - search algorithm (how to move through state space)

The typical structure search operators

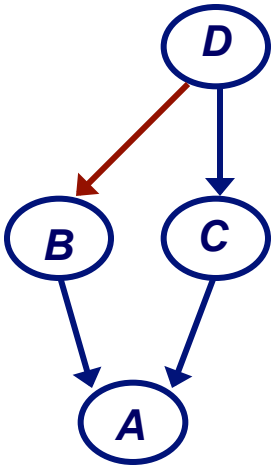
given the current network
at some stage of the search,
we can…

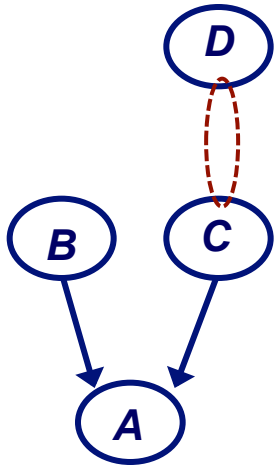


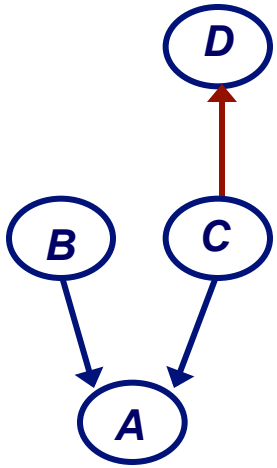
add an edge

delete an edge

reverse an edge







Scoring function decomposability

- If score is *likelihood*, and all instances in D are complete, then score can be decomposed as follows (and so can some other scores we'll see later)

$$\text{score}(G, D) = \sum_i \text{score}(X_i, Parents(X_i) : D)$$

- thus we can
  - score a network by summing terms over the nodes in the network
  - efficiently score changes in a *local* search procedure

Bayesian network search: *hill-climbing*

given: data set D, initial network B_0

$i = 0$
$\mathrm{B}_{best} \leftarrow B_0$
while stopping criteria not met
{
    for each possible operator application a
    {
        $B_{new} \leftarrow \text{apply}(a, B_i)$
        if score(B_{new}) > score(B_{best})
            $\mathrm{B}_{best} \leftarrow B_{new}$
    }
    $++i$
    $B_i \leftarrow B_{best}$
}
return B_i

Bayesian network search: the *Sparse Candidate* algorithm

[Friedman et al., *UAI* 1999]

given: data set D, initial network B_0, parameter k

$i = 0$

repeat

{

$++i$

// restrict step

select for each variable X_j a set C_j^i ($|C_j^i| \leq k$) of candidate parents

// maximize step

find network B_i maximizing score among networks where $\forall X_j$, Parents(X_j) $\subseteq C_j^i$

} until convergence

return B_i

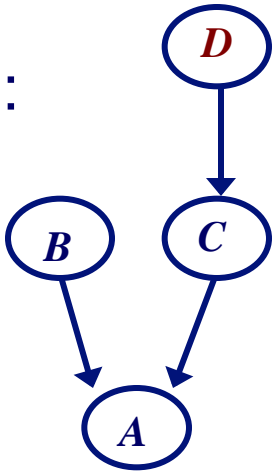
The *restrict* step in Sparse Candidate

- to identify candidate parents in the first iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

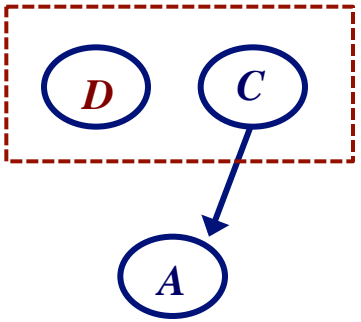
The *restrict* step in Sparse Candidate

- suppose true network structure is:



- we're selecting two candidate parents for A, and $I(A, C) > I(A, D) > I(A, B)$

- the candidate parents for A would then be C and D ; the maximize step would select C



- how could we get B as a candidate parent on the next iteration?

The *restrict* step in Sparse Candidate

- to identify candidate parents in the first iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

The *restrict* step in Sparse Candidate

- to identify candidate parents in the first iteration, can compute the *mutual information* between pairs of variables

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

- subsequent iterations keep current parents in candidate set and condition on parents with *conditional mutual information:*

$$I(X,Y \mid Z) = \sum_{x,y} P(x,y,z) \log \frac{P(x,y \mid z)}{P(x \mid z)P(y \mid z)}$$

Another view of the *restrict* step

- *Kullback-Leibler* (KL) *divergence* provides a distance measure between two distributions, P and Q

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

- mutual information can be thought of as the KL divergence between the distributions

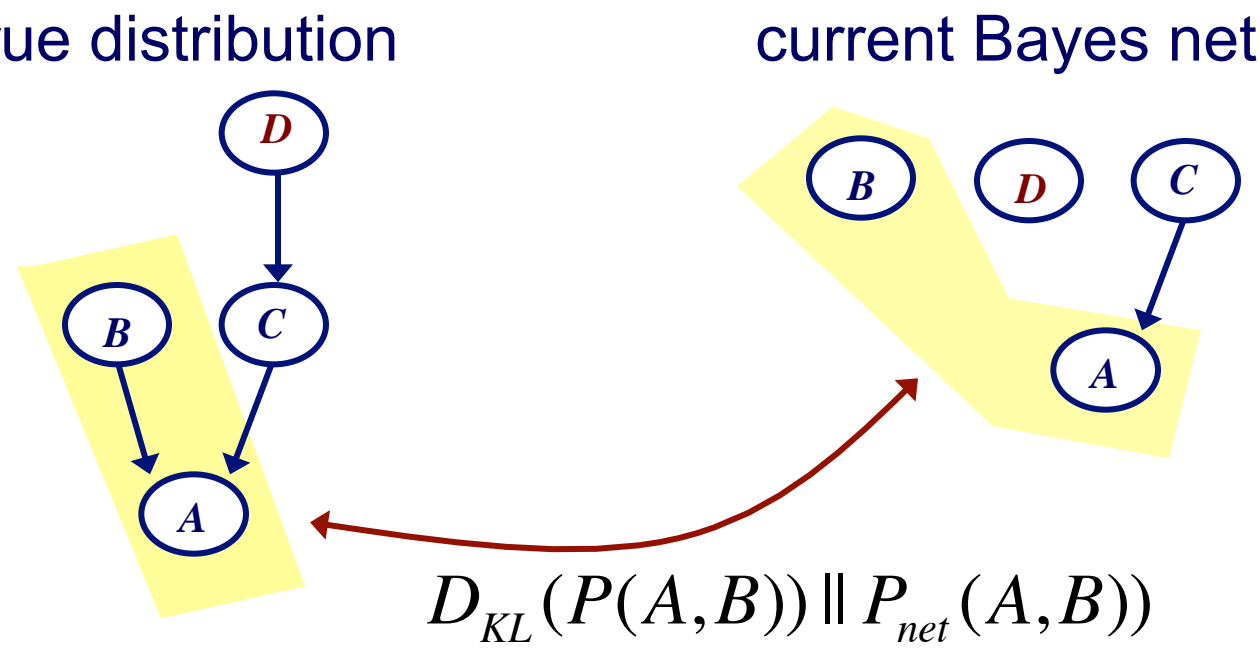
$$P(X,Y)$$

$$P(X)P(Y)$$ (assumes X and Y are independent)

The *restrict* step in Sparse Candidate

- we can use KL to assess the discrepancy between the network's estimate $P_{net}(X, Y)$ and the empirical estimate

$$M(X,Y) = D_{KL}(P(X,Y)) \| P_{net}(X,Y))$$



- Can estimate $P_{net}(X, Y)$ by sampling from the network (i.e. using it to generate instances).

The *restrict* step in Sparse Candidate

given: data set D, current network B_i, parameter k

for each variable X_j
{
  calculate CMI(X_j , X_l |Parents(X_j) for all $X_j \neq X_l$ such that
    $X_l \notin$ Parents(X_j)

  choose highest ranking $X_1 \ldots X_{k-s}$ where $s=$ | Parents(X_j) |

  // include current parents in candidate set to ensure monotonic
  // improvement in scoring function
  $C_j^i =$Parents(X_j) $\cup$ $X_1 \ldots X_{k-s}$
}
return { C_j^i } for all X_j

The *maximize* step in Sparse Candidate

- hill-climbing search with *add-edge*, *delete-edge*, *reverse-edge* operators
- test to ensure that cycles aren't introduced into the graph

Scoring functions for structure learning

- Can we find a good structure just by trying to maximize the likelihood of the data?

$$\arg\max_{G,\,\theta_G} \log P(D \mid G, \theta_G)$$

- If we have a strong restriction on the the structures allowed (e.g. a tree), then maybe.

- Otherwise, no! Adding an edge will never decrease likelihood. Overfitting likely.

Scoring functions for structure learning

- there are many different scoring functions for BN structure search
- one general approach where *n* is number of data points

$$\arg\min_{G,\,\theta_G} f(n)\left|\theta_G\right| - \log P(D \mid G, \theta_G)$$

complexity penalty

Akaike Information Criterion (AIC): $f(n) = 1$

Bayesian Information Criterion (BIC): $f(n) = \frac{1}{2}\log(n)$

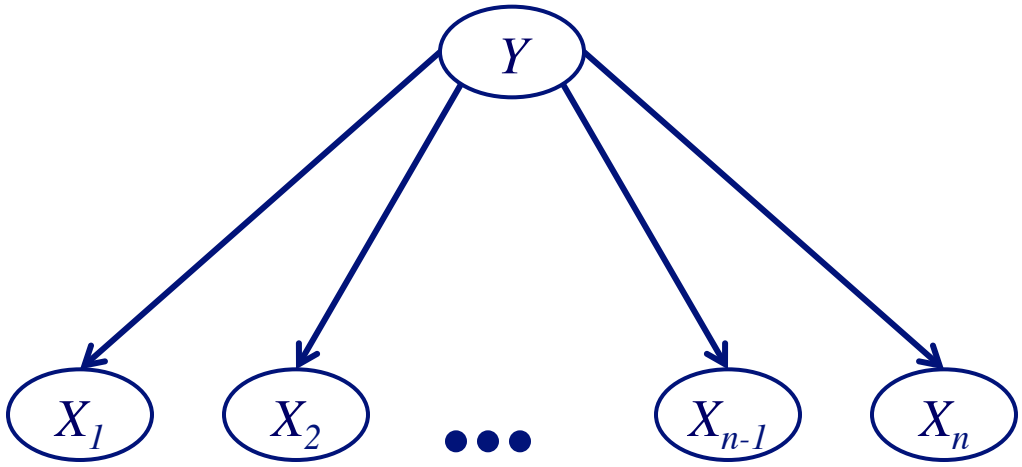
Bayes nets for classification

- the learning methods for BNs we've discussed so far can be thought of as being unsupervised
  - the learned models are not constructed to predict the value of a special class variable
  - instead, they can predict values for arbitrarily selected query variables

- now let's consider BN learning for a standard supervised task (learn a model to predict Y given $X_1 \ldots X_n$)

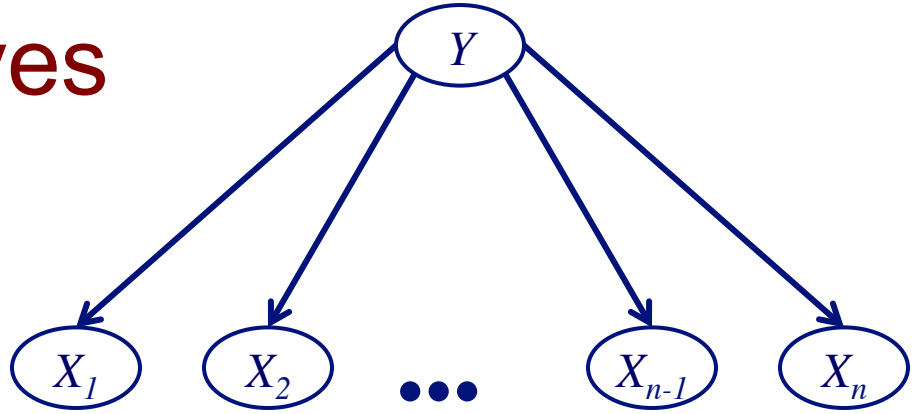
Naïve Bayes

- one very simple BN approach for supervised tasks is *naïve Bayes*
- in naïve Bayes, we assume that all features X_i are conditionally independent given the class Y



$$P(X_1, \ldots, X_n, Y) = P(Y)\prod_{i=1}^{n} P(X_i \mid Y)$$

Naïve Bayes



Learning

- estimate $P(Y = y)$ for each value of the class variable Y
- *estimate* $P(X_i = x \mid Y = y)$ *for each* X_i

Classification: use Bayes' Rule

$$P(Y = y \mid \boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x} \mid y)}{\sum_{y' \in \text{values}(Y)} P(y')P(\boldsymbol{x} \mid y')} = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{\sum_{y' \in \text{values}(Y)} \left(P(y')\prod_{i=1}^{n} P(x_i \mid y') \right)}$$

Comments about Naïve Bayes

- Can extend to numeric features
  - Assume P(X|Y) is normal with unknown mean and variance
  - Estimate these by sample mean and variance
  - During inference, just plug X into pdf of normal
- Given n features, just n-1 free parameters in discrete case
- Very robust against overfitting, but sensitive to duplicate features (strong correlations)

Naïve Bayes vs. BNs learned with an unsupervised structure search

test-set error on 25 data sets from the UC-Irvine Repository

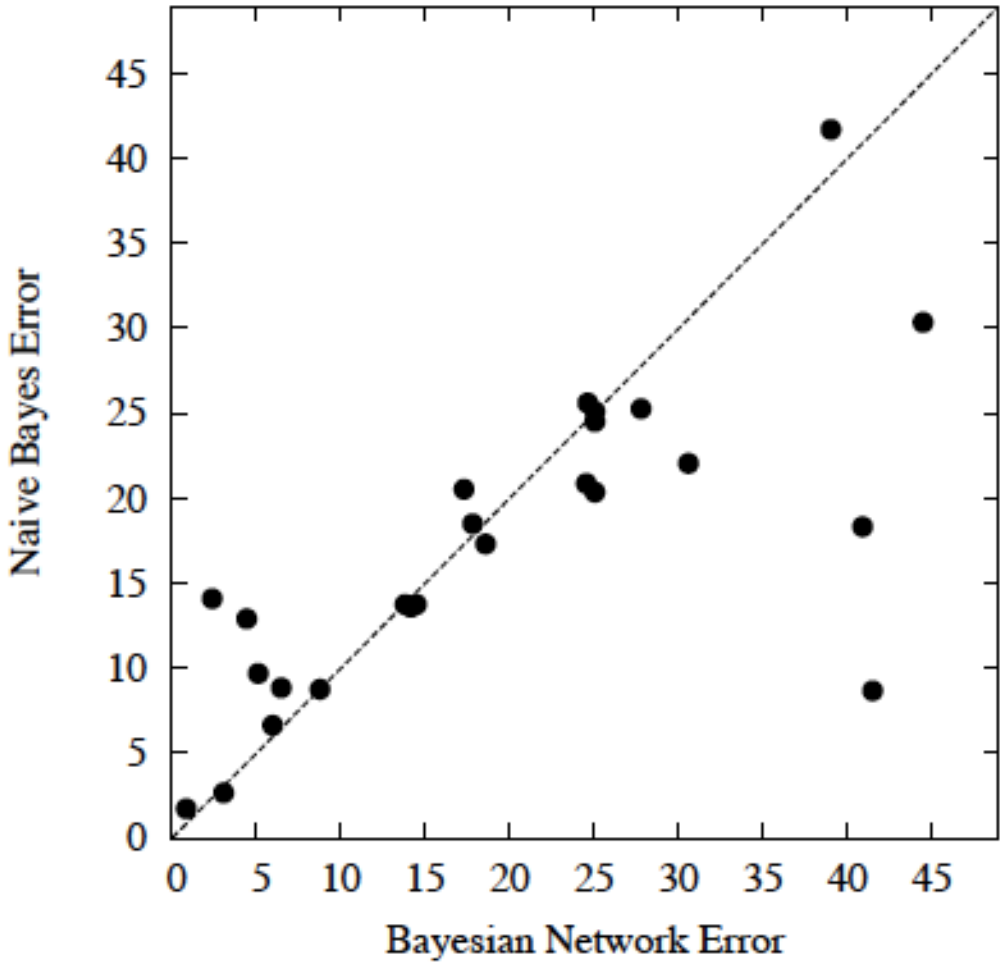


Figure from Friedman et al., *Machine Learning* 1997

The Tree Augmented Network (TAN) algorithm

[Friedman et al., *Machine Learning* 1997]

- learns a tree structure to augment the edges of a naïve Bayes network

- algorithm
  1. compute weight $I(X_i, X_j \mid Y)$ for each possible edge (X_i, X_j) between features
  2. find maximum weight spanning tree (MST) for graph over $X_1 \ldots X_n$
  3. assign edge directions in MST
  4. construct a TAN model by adding node for Y and an edge from Y to each X_i

Conditional mutual information in the TAN algorithm

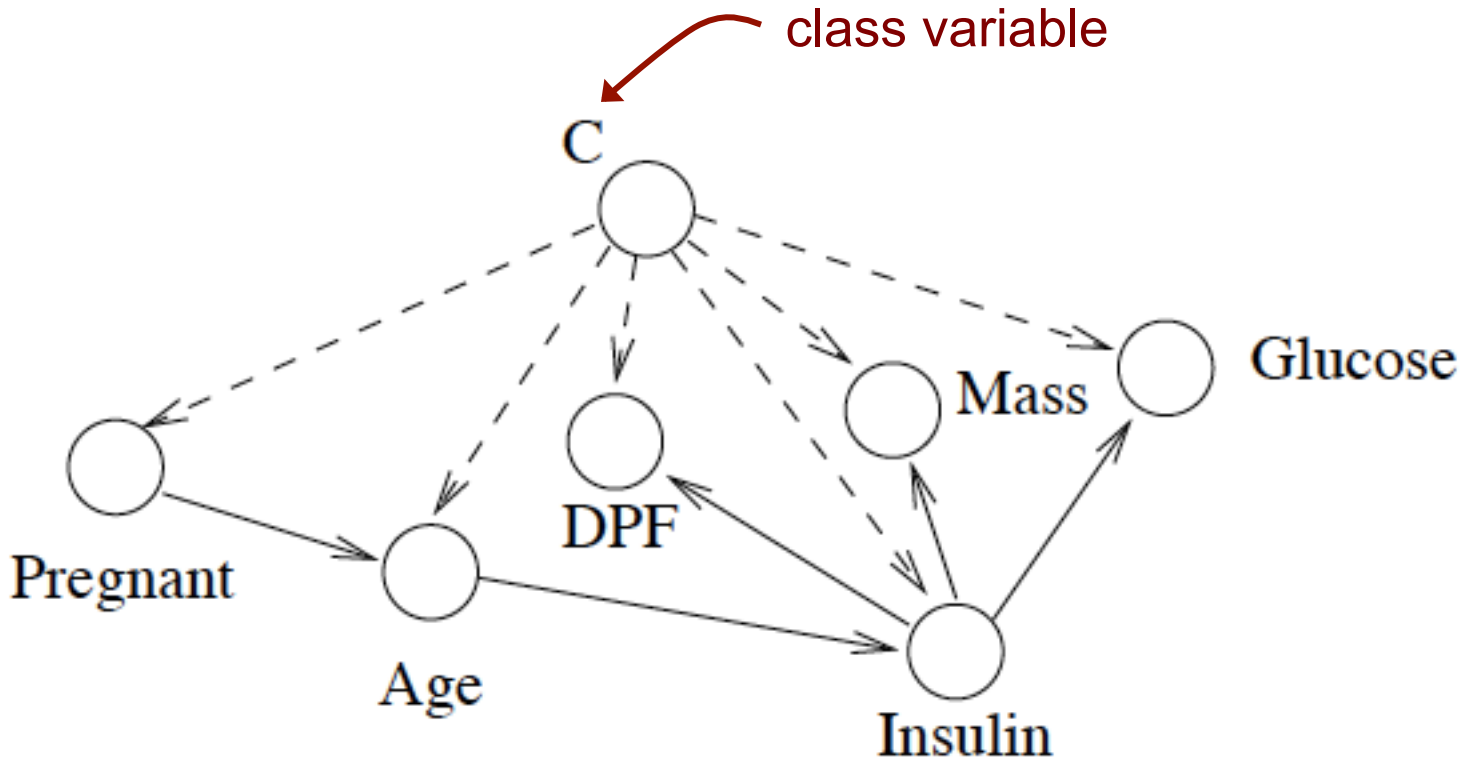
conditional mutual information is used to calculate edge weights

$$I(X_i, X_j \mid Y) =$$

$$\sum_{x_i \in values(X_i)} \sum_{x_j \in values(X_j)} \sum_{y \in values(Y)} P(x_i, x_j, y) \log_2 \frac{P(x_i, x_j \mid y)}{P(x_i \mid y) P(x_j \mid y)}$$

"how much information X_i provides about X_j when the value of Y is known"

Example TAN network



naïve Bayes edges ----->

tree augmented edges ——>

TAN vs. Chow-Liu

- TAN is mostly* focused on learning a Bayes net specifically for classification problems
- the MST includes only the feature variables (the class variable is used only for calculating edge weights)
- conditional mutual information is used instead of mutual information in determining edge weights in the undirected graph
- the directed graph determined from the MST is added to the $Y \rightarrow X_i$ edges that are in a naïve Bayes network

*TAN is still generative in maximizing P(**x**,*y*), not P(*y*|**x**)

TAN vs. Naïve Bayes

test-set error on 25 data sets from the UC-Irvine Repository

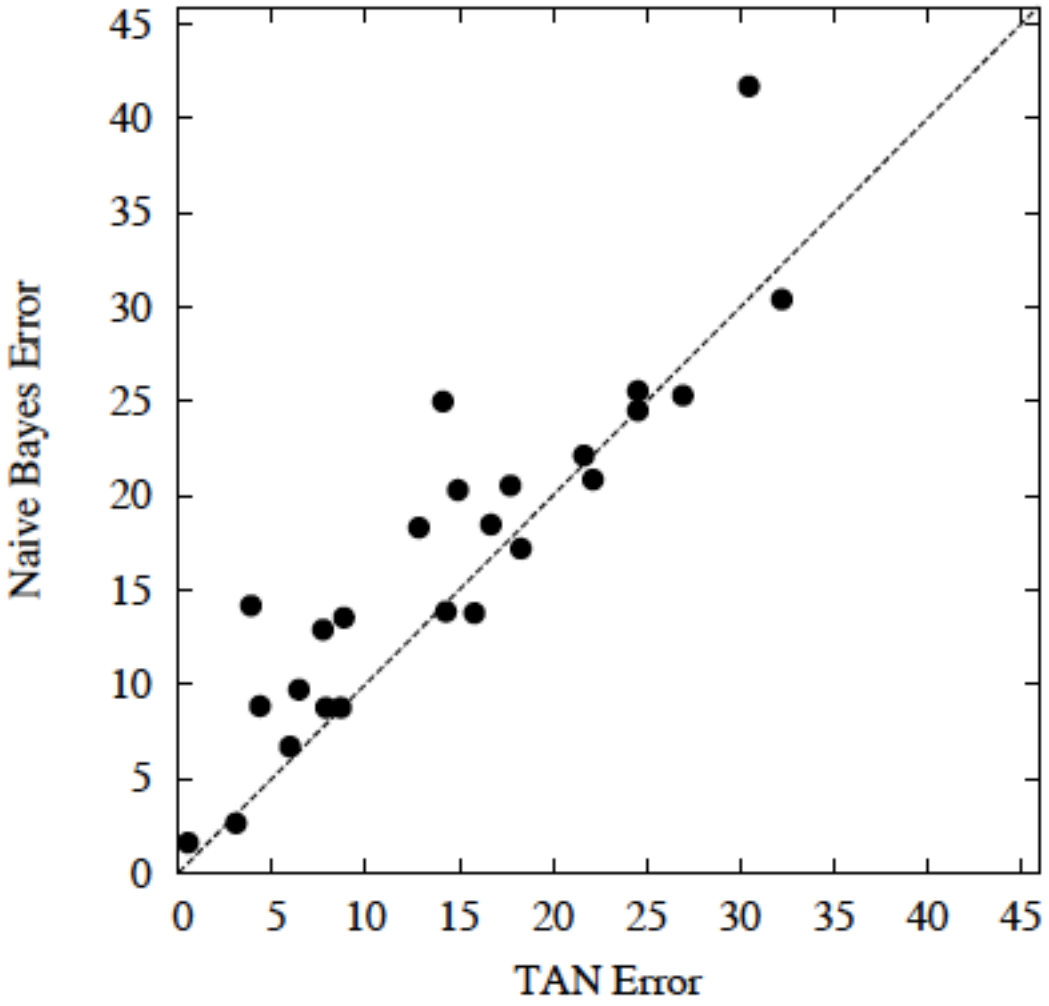
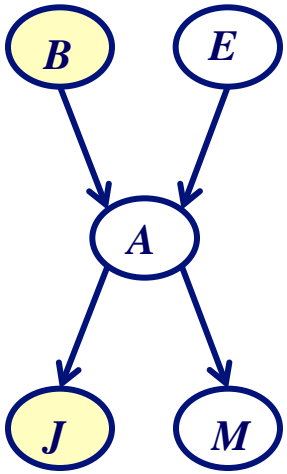


Figure from Friedman et al., *Machine Learning* 1997

Back to maximum likelihood estimation

consider estimating the CPD parameters for B and J in the alarm network given the following data set



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
t	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{1}{8} = 0.125$$

$$P(\neg b) = \frac{7}{8} = 0.875$$

$$P(j \mid a) = \frac{3}{4} = 0.75$$

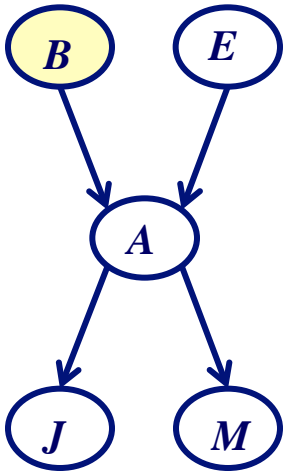
$$P(\neg j \mid a) = \frac{1}{4} = 0.25$$

$$P(j \mid \neg a) = \frac{2}{4} = 0.5$$

$$P(\neg j \mid \neg a) = \frac{2}{4} = 0.5$$

Maximum likelihood estimation

suppose instead, our data set was this…



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
f	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{0}{8} = 0$$

$$P(\neg b) = \frac{8}{8} = 1$$

do we really want to set this to 0?

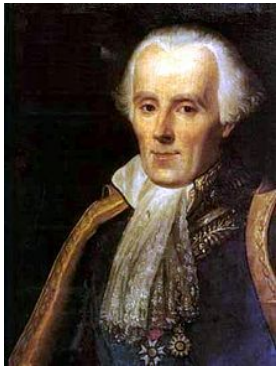
Maximum a posteriori (MAP) estimation

- instead of estimating parameters strictly from the data, we could start with some prior belief for each

- for example, we could use *Laplace estimates*

$$P(X = x) = \frac{n_x + 1}{\sum_{v \in \text{Values}(X)} (n_v + 1)}$$

pseudocounts



- where n_v represents the number of occurrences of value v

Maximum a posteriori estimation

a more general form: *m-estimates*

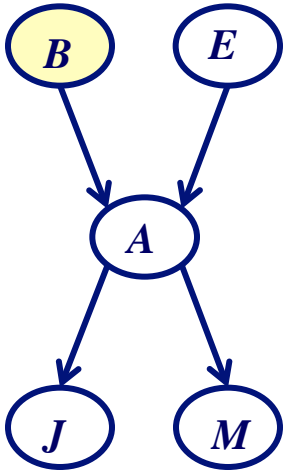
$$P(X = x) = \frac{n_x + p_x m}{\left(\sum_{v \in \text{Values}(X)} n_v \right) + m}$$

prior probability of value x

number of "virtual" instances

M-estimates example

now let's estimate parameters for B using $m=4$ and $p_b=0.25$



B	E	A	J	M
f	f	f	t	f
f	t	f	f	f
f	f	f	t	t
f	f	f	f	t
f	f	t	t	f
f	f	t	f	t
f	f	t	t	t
f	f	t	t	t

$$P(b) = \frac{0 + 0.25 \times 4}{8 + 4} = \frac{1}{12} = 0.08 \qquad P(\neg b) = \frac{8 + 0.75 \times 4}{8 + 4} = \frac{11}{12} = 0.92$$

Pseudocounts are really parameters of a Beta distribution

$$\underset{a,b}{\beta}(x) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1},$$

$$0 \leq x \leq 1 \text{ and } a, b > 0$$

$$\text{Here } \Gamma(y) = \int_0^\infty x^{y-1} e^{-x} dx$$

Any intuition for this?

- For any positive integer *y*, $\Gamma(y) = (y-1)!$.
- Suppose we use this, and we also replace
  - *x* with *p*
  - *a* with *x*
  - *a+b* with *n*
- Then we get: $\frac{(n+1)!}{x!(n-x)!} p^x (1-p)^{n-x}$
- The beta(a,b) is just the binomial(n,p) where n=a+p, and p becomes the variable. With change of variable, we need a different normalizing constant so the sum (integral) is one. Hence (n+1)! replaces n!.

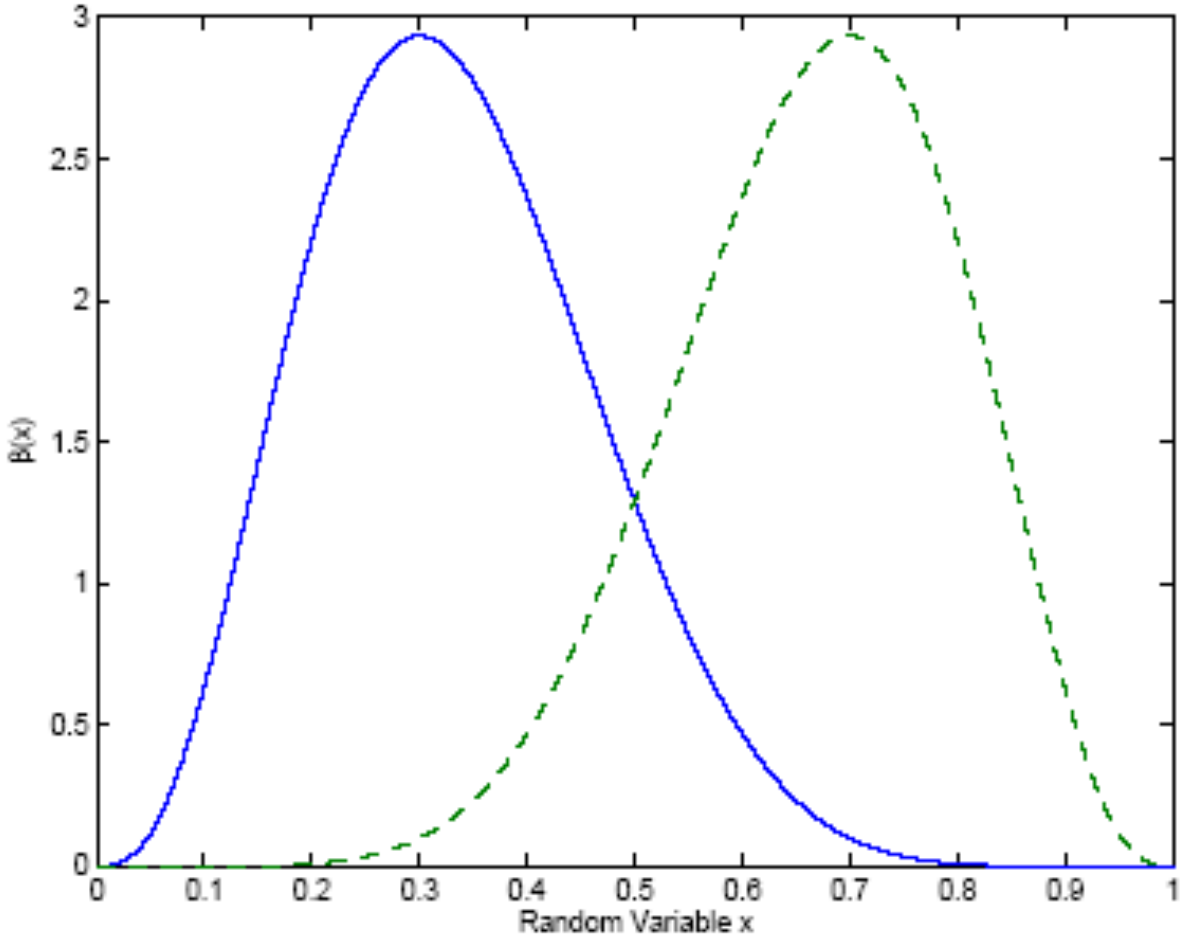


Figure 3. Beta distributions with $a = 4$ and $b = 8$ (solid line) and with $a = 8$ and $b = 4$ (dashed line). To get a higher peak (and stronger skew), use a and b that sum to a higher value.

Incorporating a Prior

- We assume a *beta* distribution as our prior distribution over the parameter *p*.
- Nice properties: unimodal, we can choose the mode to reflect the most probable value, we can choose the variance to reflect our confidence in this value.
- Best property: a *beta* distribution is parameterized by two positive numbers: *a* ...

Beta Distribution (Continued)

- (Continued)… and *b*. Higher values of *a* relative to *b* cause the mode of the distribution to be more to the left, and higher values of both *a* and *b* cause the distribution to be more peaked (lower variance). We might for example take *a* to be the number of *heads*, and *b* to be the number of *tails*. At any time, the mode of

Beta Distribution (Continued)

- (Continued)… the beta distribution (the expectation for *p*) is *a/(a+b)*, and as we get more data, the distribution becomes more peaked reflecting higher confidence in our expectation. So we can specify our prior belief for *p* by choosing initial values for *a* and *b* such that *a/(a+b)=p*, **and** we can specify confidence in this belief with high

Beta Distribution (Continued)

- (Continued)… initial values for *a* and *b*. Updating our prior belief based on data to obtain a posterior belief simply requires incrementing *a* for every *heads* outcome and incrementing *b* for every tails outcome.
- So after *h* heads out of *n* flips, our posterior distribution says P(*heads*)=(*a*+*h*)/(*a*+*b*+*n*).

Dirichlet Distributions

- What if our variable is not Boolean but can take on more values? (Let's still assume our variables are discrete.)
- *Dirichlet* distributions are an extension of beta distributions for the multi-valued case (corresponding to the extension from binomial to multinomial distributions).
- A Dirichlet distribution over a variable with *n* values has *n* parameters rather than 2.

Missing data

- Commonly in machine learning tasks, some feature values are missing

- some variables may not be observable (i.e. *hidden*) even for training instances

- values for some variables may be *missing at random*: what caused the data to be missing does not depend on the missing data itself
  - e.g. someone accidentally skips a question on an questionnaire
  - e.g. a sensor fails to record a value due to a power blip

- values for some variables may be *missing systematically*: the probability of value being missing depends on the value
  - e.g. a medical test result is missing because a doctor was fairly sure of a diagnosis given earlier test results
  - e.g. the graded exams that go missing on the way home from school are those with poor scores

Missing data

- hidden variables; values *missing at random*
  - these are the cases we'll focus on
  - one solution: try impute the values

- values *missing systematically*
  - may be sensible to represent "*missing*" as an explicit feature value
  - might build predictive models for features and use them to impute

Imputing missing data with EM

Given:

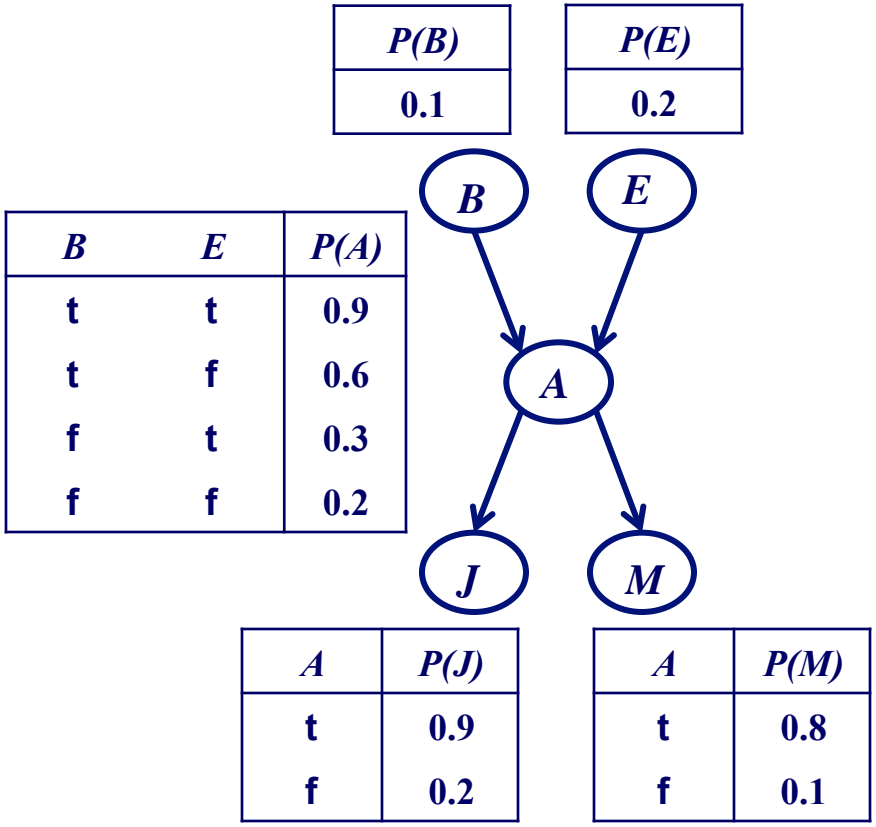
- data set with some missing values
- model structure, initial model parameters

Repeat until convergence

- *Expectation* (E) step: using current model, compute expectation over missing values
- *Maximization* (M) step: given the expectations, compute maximum likelihood (MLE) or maximum posterior probability (MAP, maximum a posteriori) parameters

example: EM for parameter learning

suppose we're given the following initial BN and training set

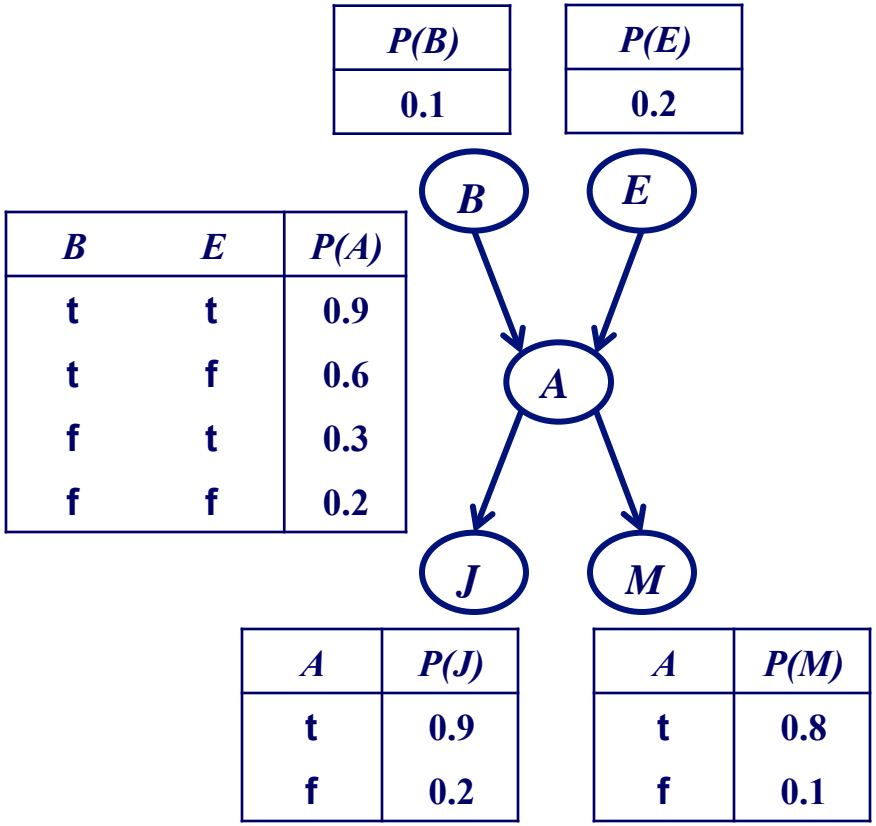


B	*E*	*A*	*J*	*M*
f	f	?	f	f
f	f	?	t	f
t	f	?	t	t
f	f	?	f	t
f	t	?	t	f
f	f	?	f	t
t	t	?	t	t
f	f	?	f	f
f	f	?	t	f
f	f	?	f	t

example: E-step

$P(a \mid \neg b, \neg e, \neg j, \neg m)$

$P(\neg a \mid \neg b, \neg e, \neg j, \neg m)$



B	E	A	J	M
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f:0.8	t	f
t	f	t:0.98 f: 0.02	t	t
f	f	t: 0.2 f: 0.8	f	t
f	t	t: 0.3 f: 0.7	t	f
f	f	t:0.2 f: 0.8	f	t
t	t	t: 0.997 f: 0.003	t	t
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f: 0.8	t	f
f	f	t: 0.2 f: 0.8	f	t

example: E-step

$$P(a \mid \neg b, \neg e, \neg j, \neg m) = \frac{P(\neg b, \neg e, a, \neg j, \neg m)}{P(\neg b, \neg e, a, \neg j, \neg m) + P(\neg b, \neg e, \neg a, \neg j, \neg m)}$$

$$= \frac{0.9 \times 0.8 \times 0.2 \times 0.1 \times 0.2}{0.9 \times 0.8 \times 0.2 \times 0.1 \times 0.2 \;+\; 0.9 \times 0.8 \times 0.8 \times 0.8 \times 0.9}$$

$$= \frac{0.00288}{.4176} = 0.0069$$

$$P(a \mid \neg b, \neg e, j, \neg m) = \frac{P(\neg b, \neg e, a, j, \neg m)}{P(\neg b, \neg e, a, j, \neg m) + P(\neg b, \neg e, \neg a, j, \neg m)}$$

$$= \frac{0.9 \times 0.8 \times 0.2 \times 0.9 \times 0.2}{0.9 \times 0.8 \times 0.2 \times 0.9 \times 0.2 \;+\; 0.9 \times 0.8 \times 0.8 \times 0.2 \times 0.9}$$

$$= \frac{0.02592}{.1296} = 0.2$$

$$P(a \mid b, \neg e, j, m) = \frac{P(b, \neg e, a, j, m)}{P(b, \neg e, a, j, m) + P(b, \neg e, \neg a, j, m)}$$

$$= \frac{0.1 \times 0.8 \times 0.6 \times 0.9 \times 0.8}{0.1 \times 0.8 \times 0.6 \times 0.9 \times 0.8 \;+\; 0.1 \times 0.8 \times 0.4 \times 0.2 \times 0.1}$$

$$= \frac{0.03456}{.0352} = 0.98$$

example: M-step

re-estimate probabilities using expected counts

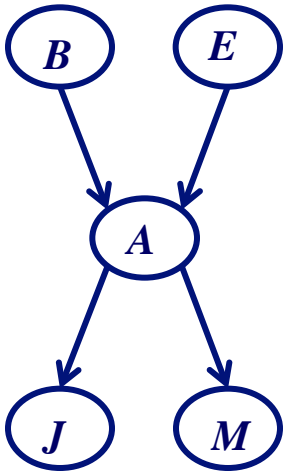
$$P(a \mid b,e) = \frac{E\#(a \wedge b \wedge e)}{E\#(b \wedge e)}$$

$$P(a \mid b,e) = \frac{0.997}{1}$$

$$P(a \mid b,\neg e) = \frac{0.98}{1}$$

$$P(a \mid \neg b,e) = \frac{0.3}{1}$$

$$P(a \mid \neg b,\neg e) = \frac{0.0069 + 0.2 + 0.2 + 0.2 + 0.0069 + 0.2 + 0.2}{7}$$



B	*E*	*P(A)*
t	t	0.997
t	f	0.98
f	t	0.3
f	f	0.145

re-estimate probabilities for $P(J \mid A)$ and $P(M \mid A)$ in same way

B	*E*	*A*	*J*	*M*
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f:0.8	t	f
t	f	t:0.98 f: 0.02	t	t
f	f	t: 0.2 f: 0.8	f	t
f	t	t: 0.3 f: 0.7	t	f
f	f	t:0.2 f: 0.8	f	t
t	t	t: 0.997 f: 0.003	t	t
f	f	t: 0.0069 f: 0.9931	f	f
f	f	t:0.2 f: 0.8	t	f
f	f	t: 0.2 f: 0.8	f	t

Convergence of EM

- E and M steps are iterated until probabilities converge
- will converge to a maximum in the data likelihood (MLE or MAP)
- the maximum may be a local optimum, however
- the optimum found depends on starting conditions (initial estimated probability parameters)

(“Soft”) EM vs. “Hard” EM

- Standard (soft) EM: expectation is a probability distribution.
- Hard EM: expectation is “all or nothing”… most likely/probable value.
- Advantage of hard EM is computational efficiency when expectation is over state consisting of values for multiple variables

EM for Parameter Learning: E Step

- For each data point with missing values, compute the probability of each possible completion of that data point. Replace the original data point with all these completions, weighted by probabilities.
- Computing the probability of each completion (expectation) is just answering query over missing variables given others.

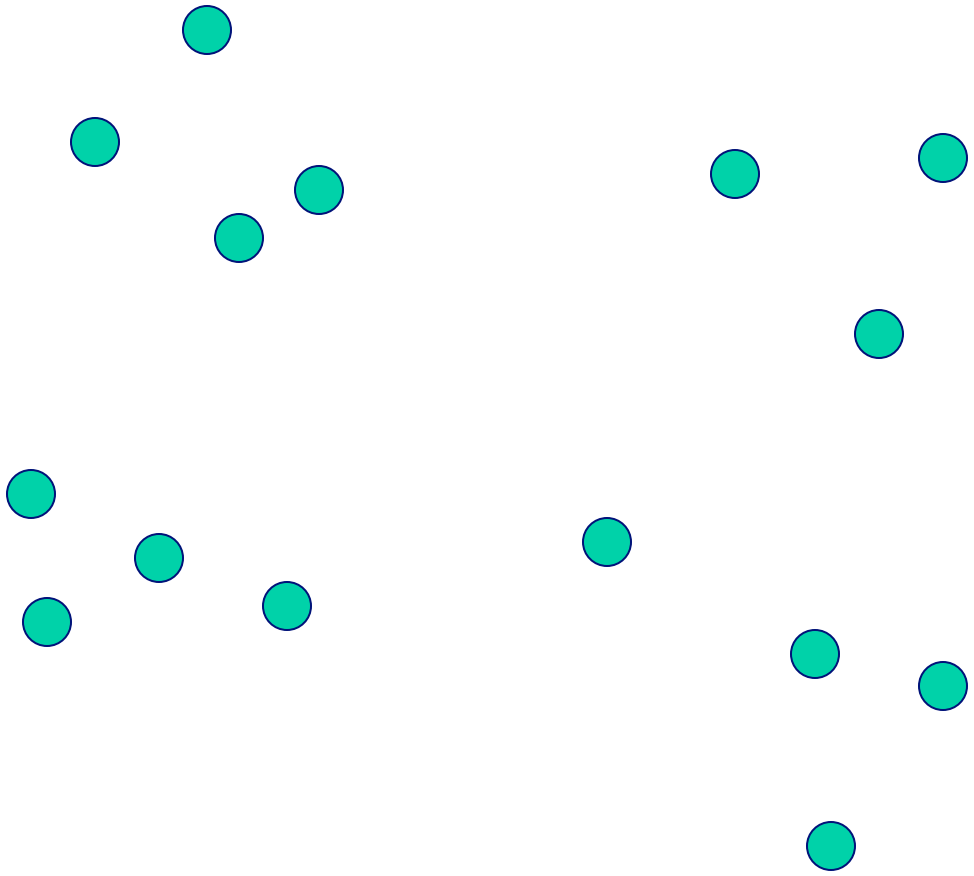
EM for Parameter Learning: M Step

- Use the completed data set to update our Dirichlet distributions as we would use any complete data set, except that our counts (tallies) may be fractional now.
- Update CPTs based on new Dirichlet distributions, as we would with any complete data set.

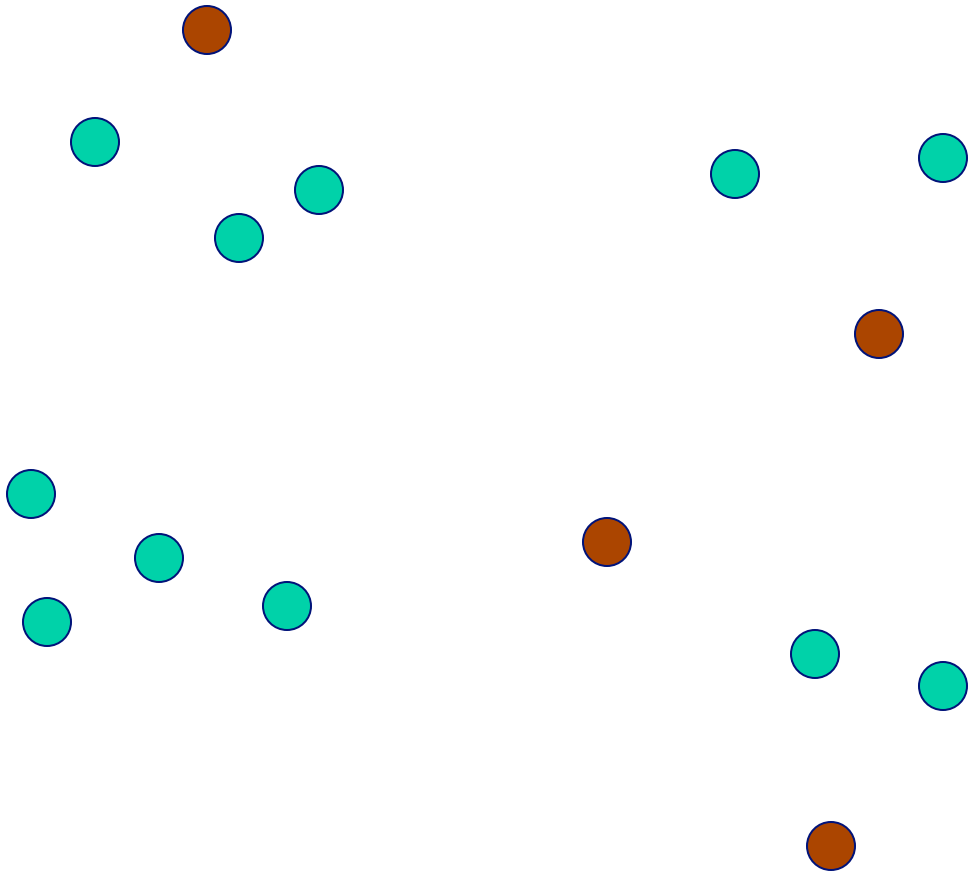
EM for Parameter Learning

- Iterate E and M steps until no changes occur. We will not necessarily get the global MAP (or ML given uniform priors) setting of all the CPT entries, but under a natural set of conditions we are guaranteed convergence to a local MAP solution.
- EM algorithm is used for a wide variety of tasks outside of BN learning as well.

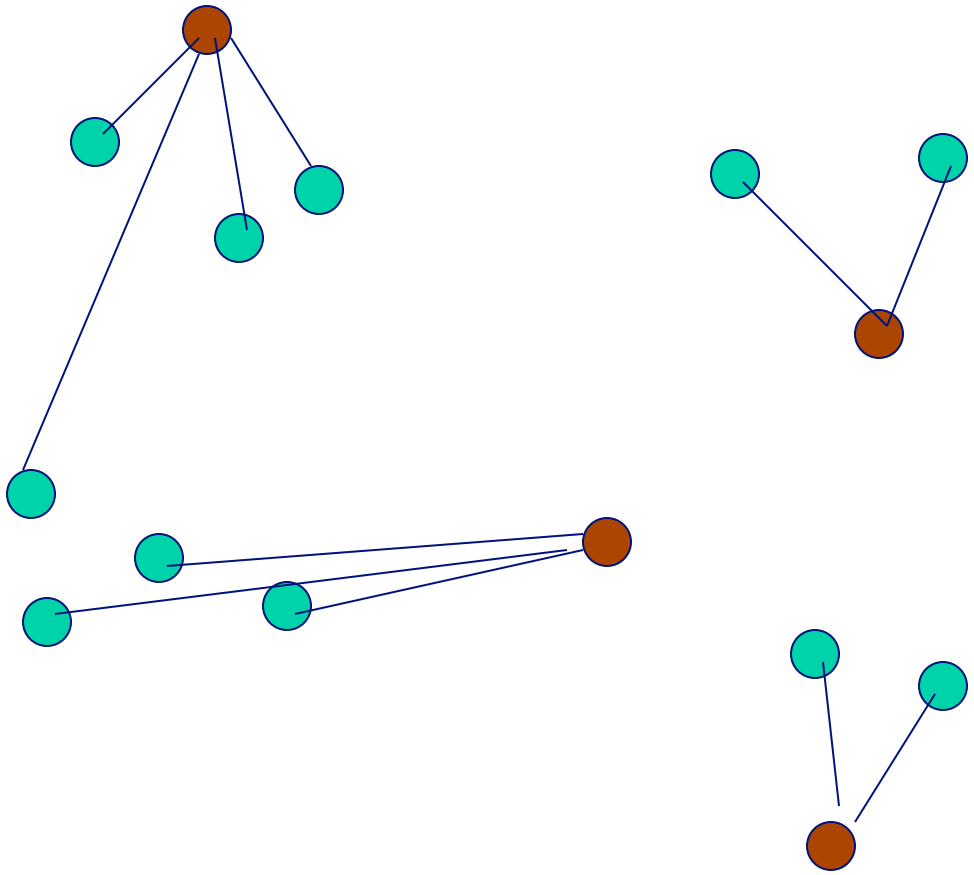
K-Means as EM



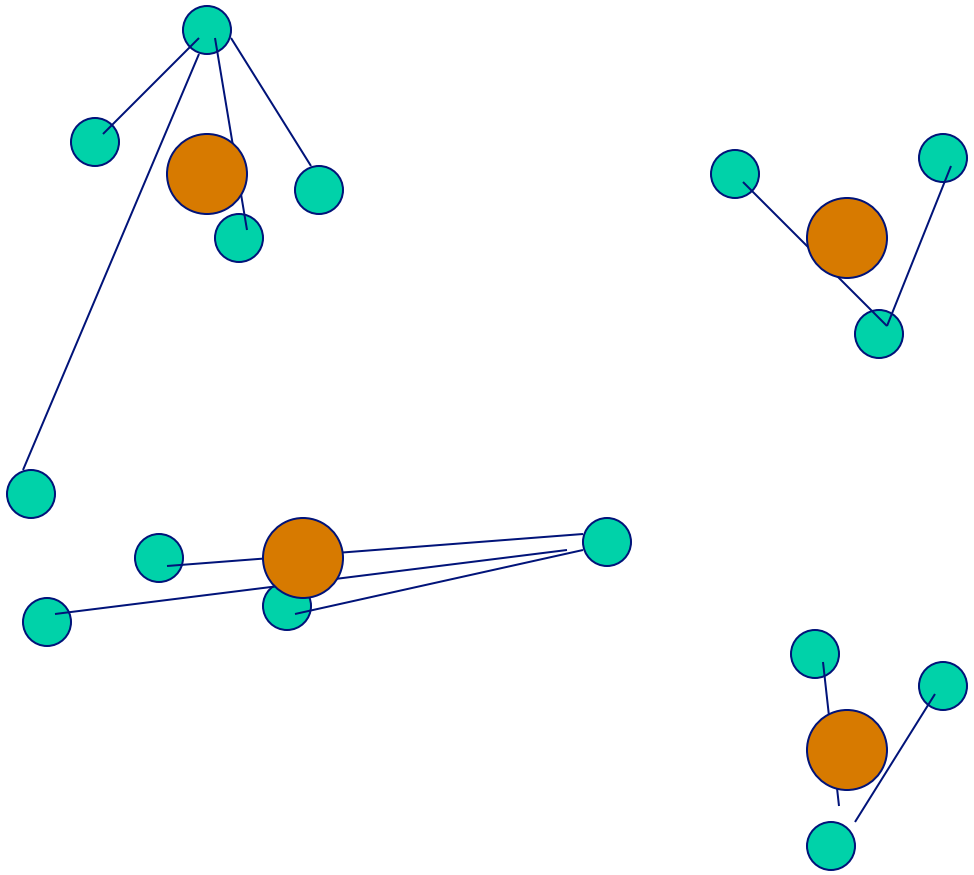
K-Means as EM



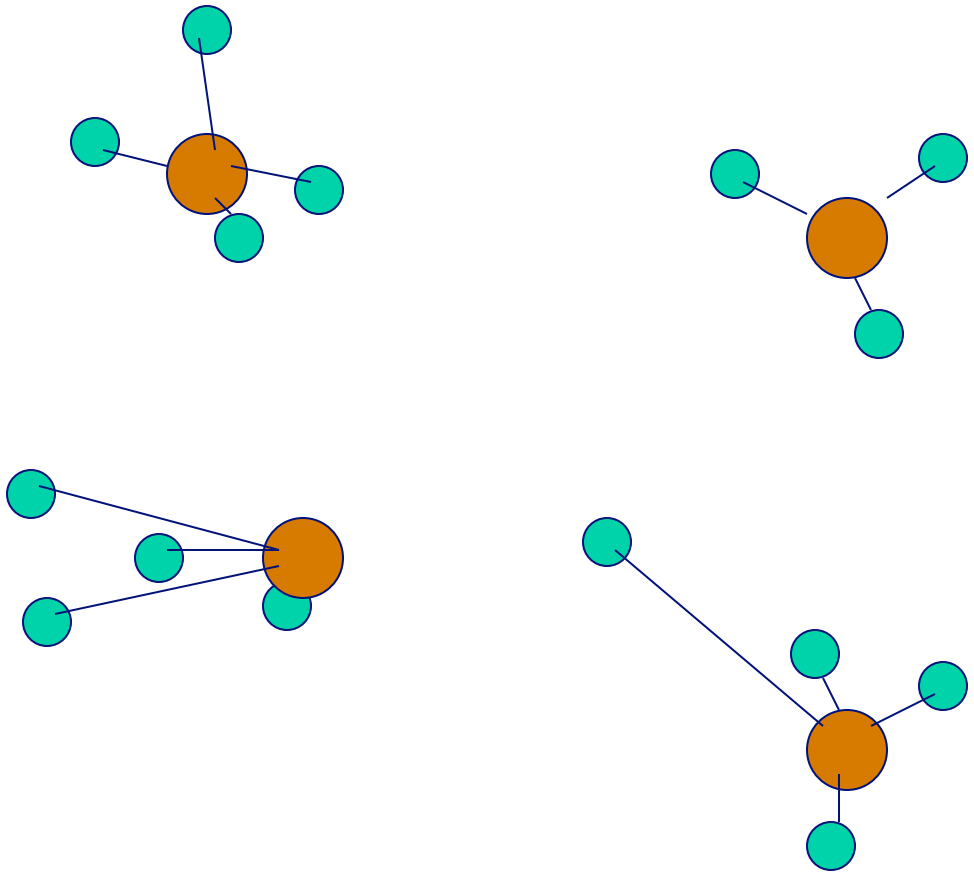
K-Means as EM



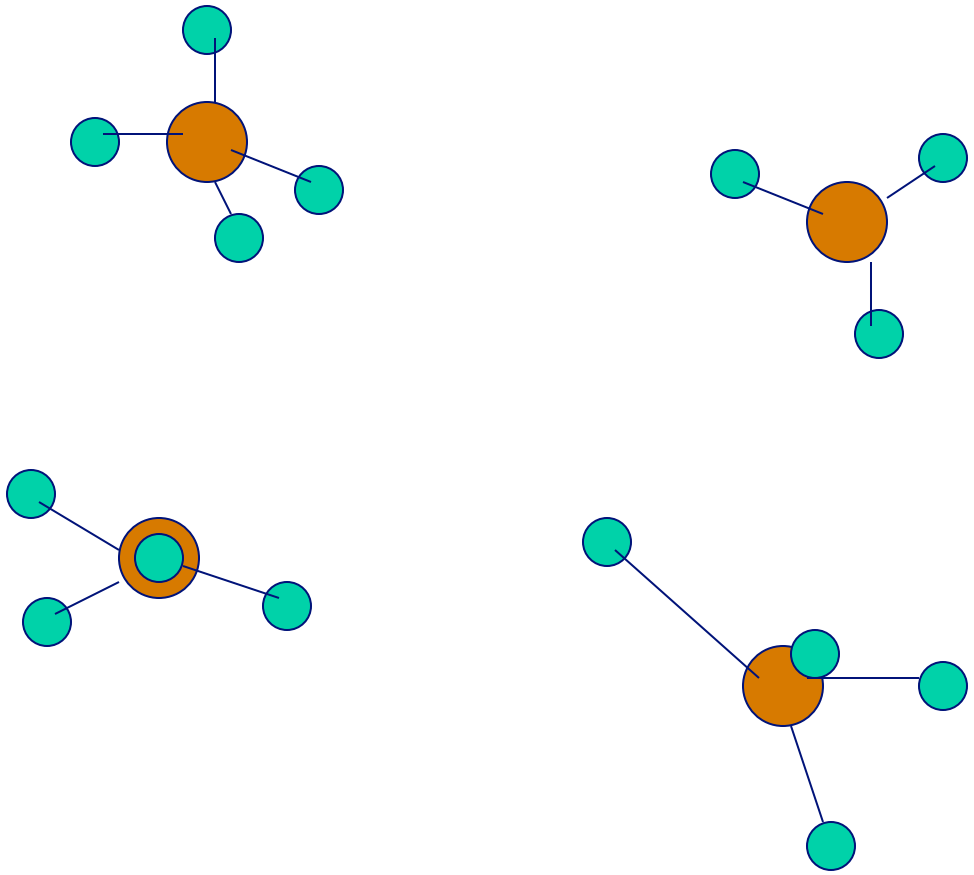
K-Means as EM



K-Means as EM



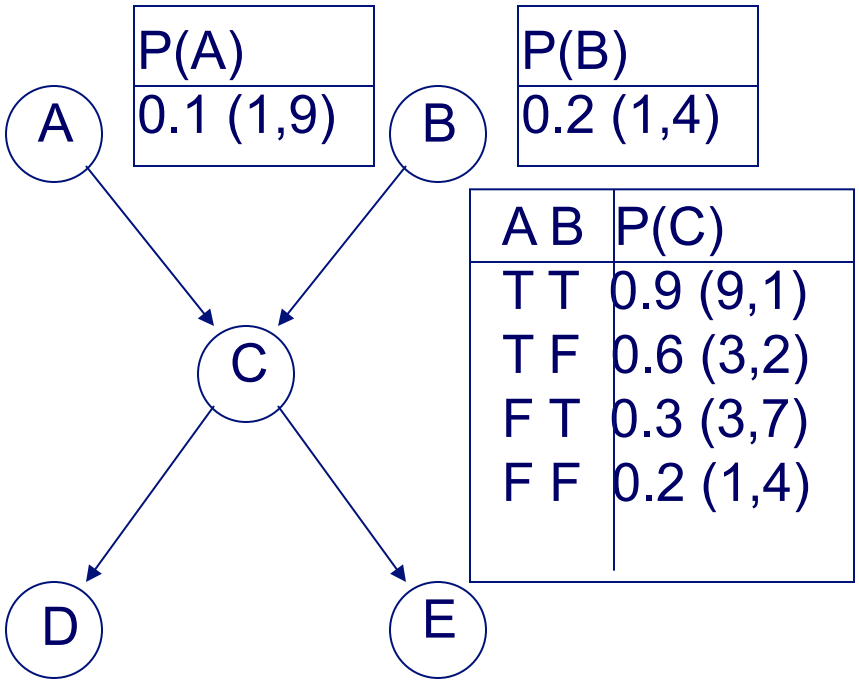
K-Means as EM



Subtlety for BN Parameter Learning via EM

- Overcounting based on number of interations required to converge to settings for the missing values.
- After each repetition of E step, reset all Dirichlet distributions before repeating M step.

EM for Parameter Learning



P(A)
0.1 (1,9)

P(B)
0.2 (1,4)

A B	P(C)
T T	0.9 (9,1)
T F	0.6 (3,2)
F T	0.3 (3,7)
F F	0.2 (1,4)

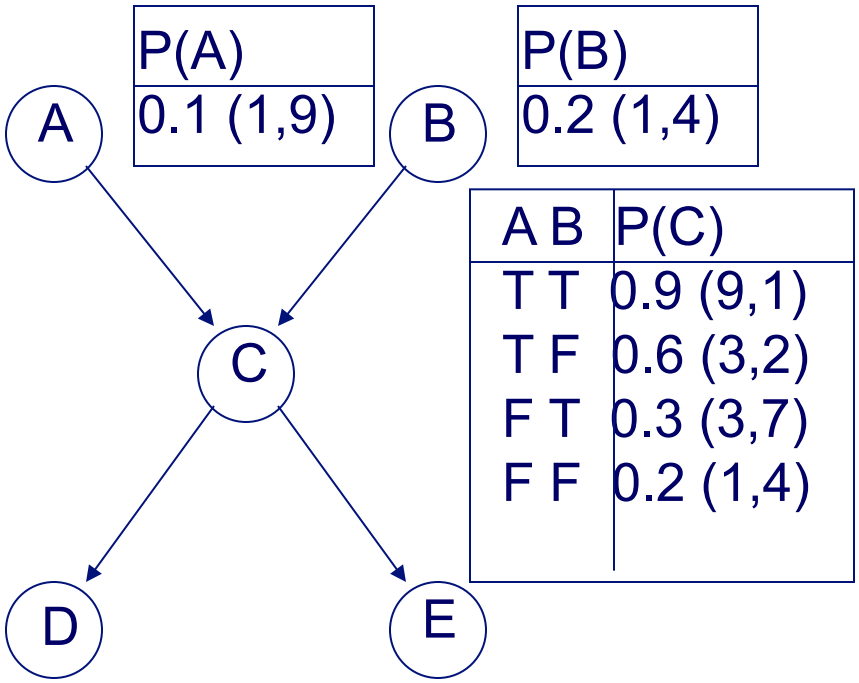
C	P(D)
T	0.9 (9,1)
F	0.2 (1,4)

C	P(E)
T	0.8 (4,1)
F	0.1 (1,9)

Data

A	B	C	D	E
0	0	?	0	0
0	0	?	1	0
1	0	?	1	1
0	0	?	0	1
0	1	?	1	0
0	0	?	0	1
1	1	?	1	1
0	0	?	0	0
0	0	?	1	0
0	0	?	0	1

EM for Parameter Learning



P(A)
0.1 (1,9)

P(B)
0.2 (1,4)

A B	P(C)
T T	0.9 (9,1)
T F	0.6 (3,2)
F T	0.3 (3,7)
F F	0.2 (1,4)

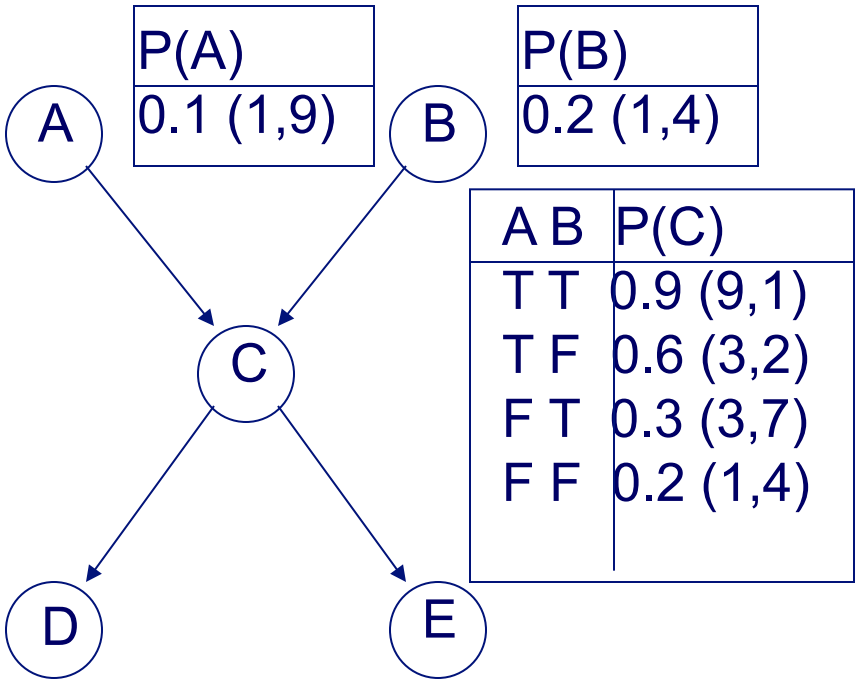
C	P(D)
T	0.9 (9,1)
F	0.2 (1,4)

C	P(E)
T	0.8 (4,1)
F	0.1 (1,9)

Data

A	B	C	D	E
0	0	0: 0.99 1: 0.01	0	0
0	0	0: 0.80 1: 0.20	1	0
1	0	0: 0.02 1: 0.98	1	1
0	0	0: 0.80 1: 0.20	0	1
0	1	0: 0.70 1: 0.30	1	0
0	0	0: 0.80 1: 0.20	0	1
1	1	0: 0.003 1: 0.997	1	1
0	0	0: 0.99 1: 0.01	0	0
0	0	0: 0.80 1: 0.20	1	0
0	0	0: 0.80 1: 0.20	0	1

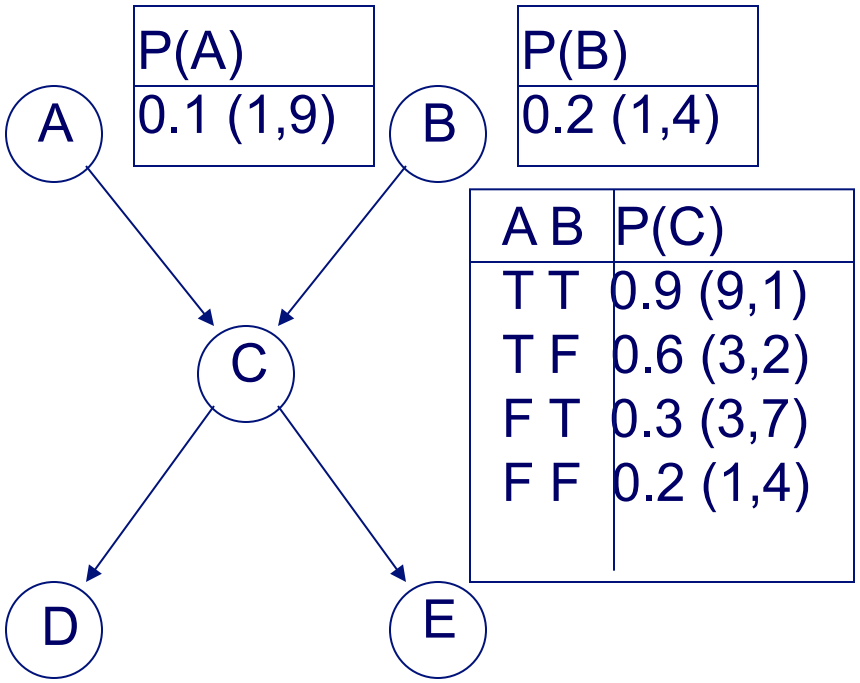
Multiple Missing Values



Data

A	B	C	D	E
?	0	?	0	1

Multiple Missing Values



P(A)
0.1 (1,9)

P(B)
0.2 (1,4)

A B	P(C)
T T	0.9 (9,1)
T F	0.6 (3,2)
F T	0.3 (3,7)
F F	0.2 (1,4)

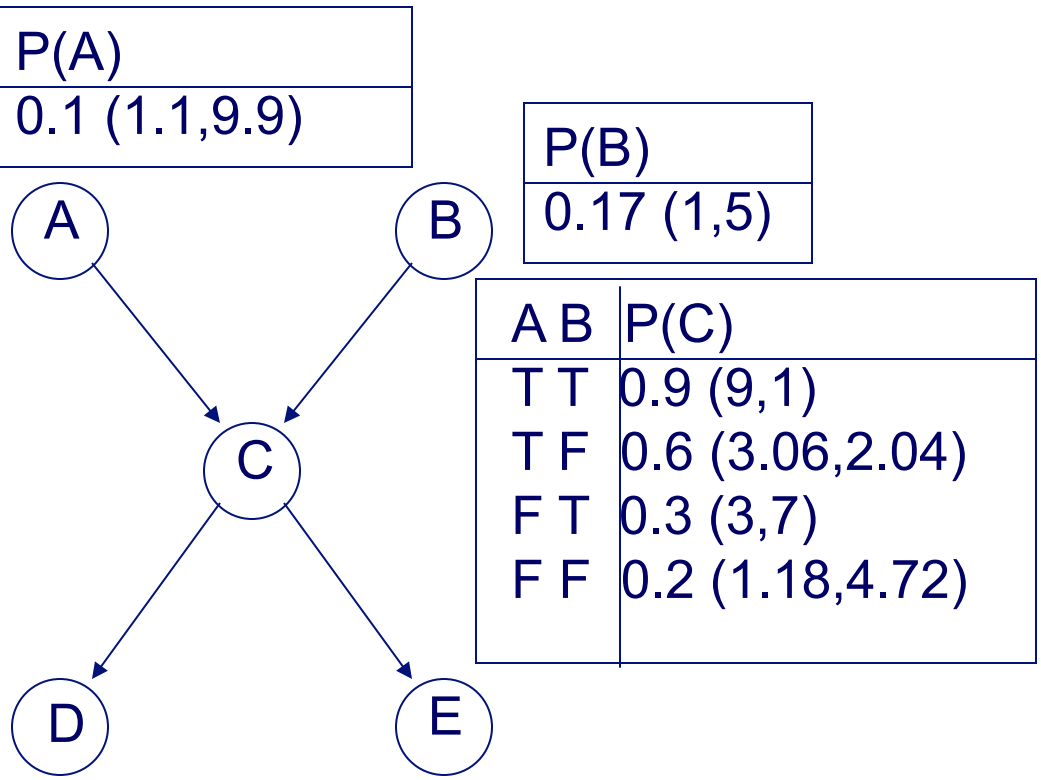
C	P(D)
T	0.9 (9,1)
F	0.2 (1,4)

C	P(E)
T	0.8 (4,1)
F	0.1 (1,9)

Data

A	B	C	D	E	
0	0	0	0	1	0.72
0	0	1	0	1	0.18
1	0	0	0	1	0.04
1	0	1	0	1	0.06

Multiple Missing Values



P(A)
0.1 (1.1,9.9)

P(B)
0.17 (1,5)

A B	P(C)
T T	0.9 (9,1)
T F	0.6 (3.06,2.04)
F T	0.3 (3,7)
F F	0.2 (1.18,4.72)

C	P(D)
T	0.88 (9,1.24)
F	0.17 (1,4.76)

C	P(E)
T	0.81 (4.24,1)
F	0.16 (1.76,9)

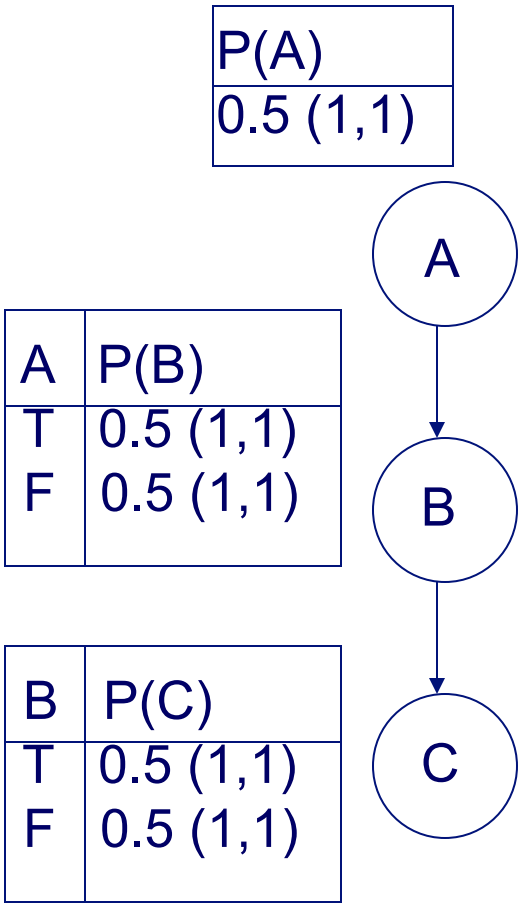
Data

A	B	C	D	E	
0	0	0	0	1	0.72
0	0	1	0	1	0.18
1	0	0	0	1	0.04
1	0	1	0	1	0.06

Problems with EM

- Only local optimum (not much way around that, though).
- Deterministic … if priors are uniform, may be impossible to make any progress…
- … next figure illustrates the need for some randomization to move us off an uninformative prior…

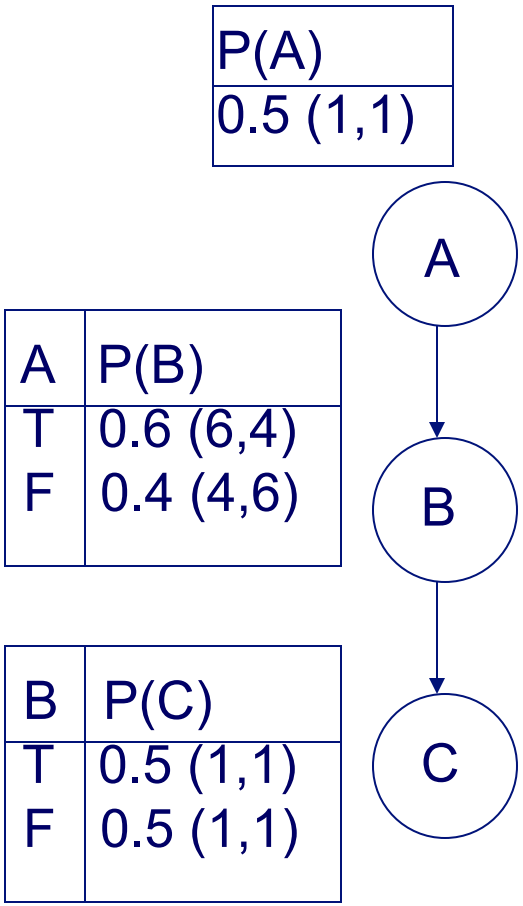
What will EM do here?



Data

A	B	C
0	?	0
1	?	1
0	?	0
1	?	1
0	?	0
1	?	1

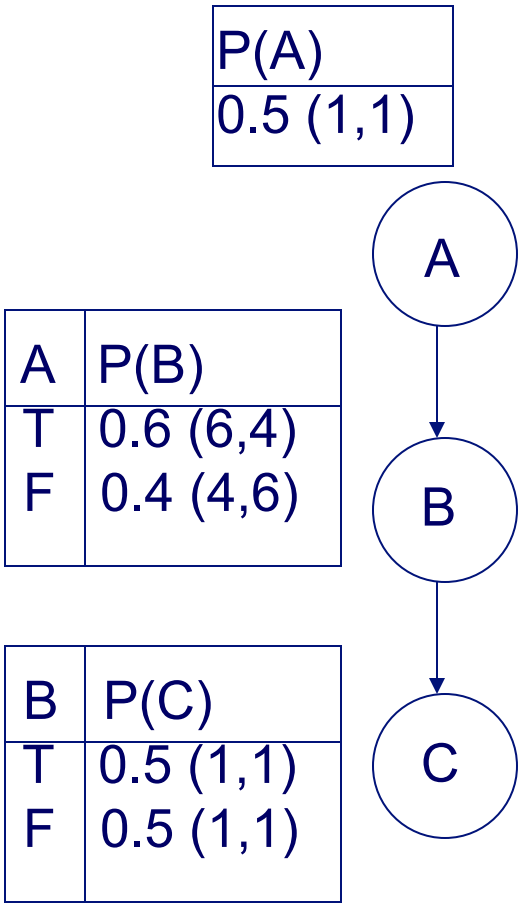
EM Dependent on Initial Beliefs



Data

A	B	C
0	?	0
1	?	1
0	?	0
1	?	1
0	?	0
1	?	1

EM Dependent on Initial Beliefs



Data

A	B	C
0	?	0
1	?	1
0	?	0
1	?	1
0	?	0
1	?	1

B is more likely *T* than *F* when A is *T*. Filling this in makes C more likely *T* than *F* when B is *T*. This makes B still more likely *T* than *F* when A is *T*. Etc. Small change in CPT for B (swap 0.6 and 0.4) would have opposite effect.

Comments on Bayesian networks

- the BN representation has many advantages
  - easy to encode domain knowledge (direct dependencies, causality)
  - can represent uncertainty
  - principled methods for dealing with missing values
  - can answer arbitrary queries (in theory; in practice may be intractable)
- for supervised tasks, it may be advantageous to use a learning approach (e.g. TAN) that focuses on the dependencies that are most important
- although very simplistic, naïve Bayes often learns highly accurate models
- BNs are one instance of a more general class of *probabilistic graphical models*