Exact Inference Algorithms for Probabilistic Reasoning; BTE and CTE

COMPSCI 276, Spring 2011

Set 6: Rina Dechter

(Reading: Primary: Class Notes (5,6)
Secondary: , Darwiche chapters 7,8)

Probabilistic Inference Tasks

- Belief updating:

$$\mathbf{BEL}(X_i) = P(X_i = x_i \mid \text{evidence})$$

- Finding most probable explanation (MPE)

$$\bar{x}^* = \arg\max_{\bar{x}} P(\bar{x}, e)$$

- Finding maximum a-posteriory hypothesis

$$(a_1^*, \ldots, a_k^*) = \arg\max_{\bar{a}} \sum_{X/A} P(\bar{x}, e)$$

$A \subseteq X$: **hypothesis variables**

- Finding maximum-expected-utility (MEU) decision

$$(d_1^*, \ldots, d_k^*) = \arg\max_{\bar{d}} \sum_{X/D} P(\bar{x}, e) U(\bar{x})$$

$D \subseteq X$: **decision variables**

$U(\bar{x})$: **utility function**

Finding $MPE = \max_{\bar{x}} P(\bar{x})$

Algorithm *elim-mpe* (Dechter 1996)

$\sum$ is replaced by ***max*** :

$$MPE = \max_{a,e,d,c,b} P(a)P(c\,|\,a)P(b\,|\,a)P(d\,|\,a,b)P(e\,|\,b,c)$$

$\max_b \prod$ ← Elimination operator

bucket B: P(b|a) P(d|b,a) P(e|b,c)

bucket C: P(c|a)

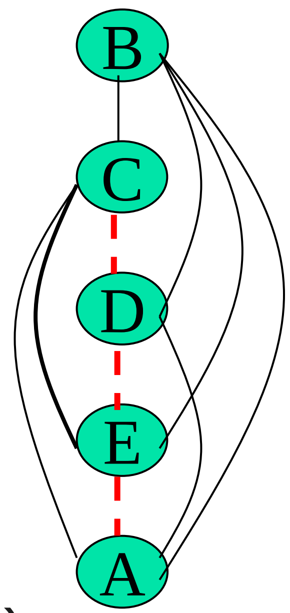
bucket D: $h^C(a,d,e)$

bucket E: e=0 $h^D(a,e)$

bucket A: P(a) $h^E(a)$

MPE

W*=4
"induced width"
(max clique size)



Generating the MPE-tuple

5. $b' = \arg\max_b P(b \mid a') \times P(d' \mid b, a') \times P(e' \mid b, c')$

4. $c' = \arg\max_c P(c \mid a') \times h^B(a', d', c, e')$

3. $d' = \arg\max_d h^C(a', d, e')$

2. $e' = 0$

1. $a' = \arg\max_a P(a) \cdot h^E(a)$

B: P(b|a) P(d|b,a) P(e|b,c)

C: P(c|a) $h^B(a,d,c,e)$

D: $h^C(a,d,e)$

E: e=0 $h^D(a,e)$

A: P(a) $h^E(a)$

Return (a', b', c', d', e')

Complexity of Bucket-elimination

- **Theorem:**

BE is $O(n \exp(w^*+1))$ time and $O(n \exp(w^*))$ space, when w^* is the induced-width of the moral graph along d when evidence nodes are processed (edges from evidence nodes to earlier variables are removed.)

More accurately: $O(r \exp(w^*(d))$ where r is the number of cpts.
For Bayesian networks r=n. For Markov networks?

Finding small induced-width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality
  - Fill-in (thought as the best)
  - See anytime min-width (Gogate and Dechter)

Min-width ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$

output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.

1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in G with smallest degree.
3. put r in position j and $G \leftarrow G - r$.
   (Delete from V node r and from E all its adjacent edges)
4. **endfor**

Proposition: algorithm min-width finds a min-width ordering of a graph
Complexity:?
O(e)

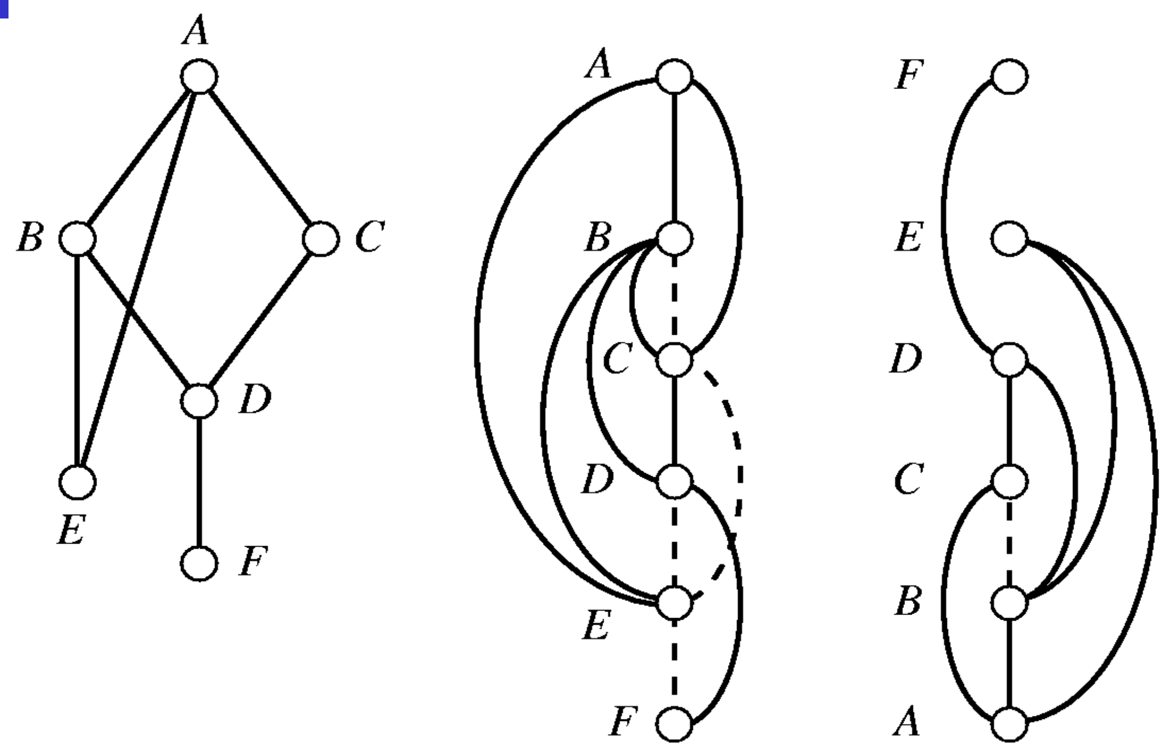
Greedy orderings heuristics

min-induced-width (miw)
input: a graph *G = (V;E), V = {v1; :::; vn}*
output: A miw ordering of the nodes *d = (v1; :::; vn).*
1. for *j = n to 1 by -1 do*
2. *r ← a node in V with smallest degree.*
3. put *r in position j.*
4. connect *r's neighbors: E ← E union {(vi; vj)| (vi; r) in E; (vj ; r) 2 in E},*
5. remove *r from the resulting graph: V ←V - {r}.*

Theorem: A graph is a tree iff it has both width and induced-width of 1.

min-fill (min-fill)
input: a graph *G = (V;E), V = {v1; :::; vn}*
output: An ordering of the nodes *d = (v1; :::; vn).*
1. for *j = n to 1 by -1 do*
2. *r ←a node in V with smallest fill edges for his parents.*
3. put *r in position j.*
4. connect *r's neighbors: E ←E union {(vi; vj)| (vi; r) 2 E; (vj ; r) in E},*
5. remove *r from the resulting graph: V ←V –{r}.*

Induced-width



(a) (b) (c) (d)

© could have been generated by min-fill

Min-induced-width

MIN-INDUCED-WIDTH (MIW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
1. **for** $j = n$ to 1 by -1 do
2. $r \leftarrow$ a node in V with smallest degree.
3. put r in position j.
4. connect r's neighbors: $E \leftarrow E \cup \{(v_i, v_j) | (v_i, r) \in E, (v_j, r) \in E\}$,
5. remove r from the resulting graph: $V \leftarrow V - \{r\}$.

Figure 4.3: The min-induced-width (MIW) procedure

Induced-width for chordal graphs

- **Definition:** A graph is chordal if every cycle of length at least 4 has a chord
- Finding w* over chordal graph is easy using the **max-cardinality ordering:** order vertices from 1 to n, always assigning the next number to the node connected to a largest set of previously numbered nodes. Lets d be such an ordering
- A graph along max-cardinality order has no fill-in edges iff it is chordal.
- On chordal graphs width=induced-width.

Max-cardinality ordering

MAX-CARDINALITY (MC)

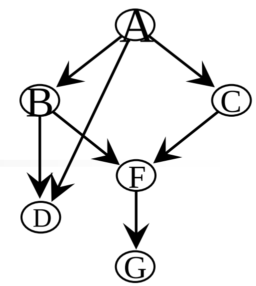
input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: An ordering of the nodes $d = (v_1, ..., v_n)$.
1. Place an arbitrary node in position 0.
2. **for** $j = 1$ to n do
3. $r \leftarrow$ a node in G that is connected to a largest subset of nodes in positions 1 to $j - 1$, breaking ties arbitrarily.
4. **endfor**

Figure 4.5 The max-cardinality (MC) ordering procedure.

Which greedy algorithm is best?

- MinFill, prefers a node who add the least number of fill-in arcs.
- Empirically, fill-in is the best among the greedy algorithms (MW,MIW,MF,MC)
- Complexity of greedy orderings?
- MW is O(?), MIW: O(?) MF (?) MC is O(mn)

From Bucket elimination to bucket-tree elimination



Bucket G: $P(G|F)$

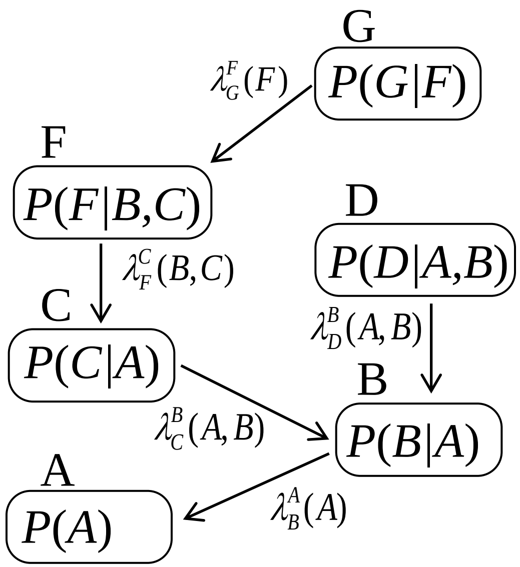
Bucket F: $P(F|B,C)$

Bucket D: $P(D|A,B)$

Bucket C: $P(C|A)$

Bucket B: $P(B|A)$

Bucket A: $P(A)$



Propagation in a Bucket Tree

Definitions:

- Let G be a Bayesian network, d, an ordering and $B_1 \ldots B_n$ the final bucket created processing along $d = x_1 \ldots x_n$.
- Let B_i be the set of variables appearing in bucket i when it is processed.

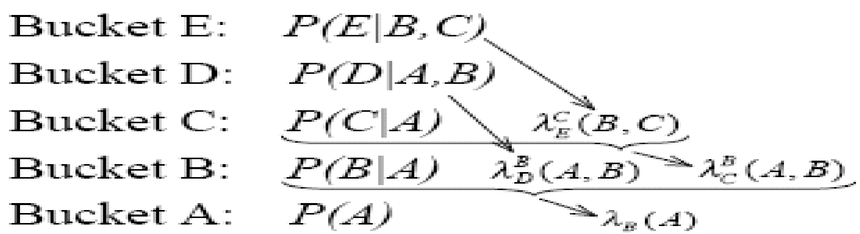
Bucket Tree:

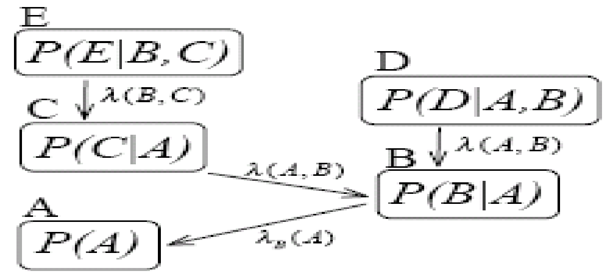
- A bucket tree has each B_i cluster as a node and there is an arc from B_i to B_j if the function created at B_i was placed in B_j

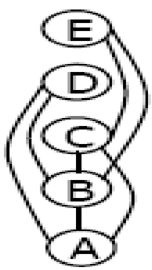
Graph-Based Definition:

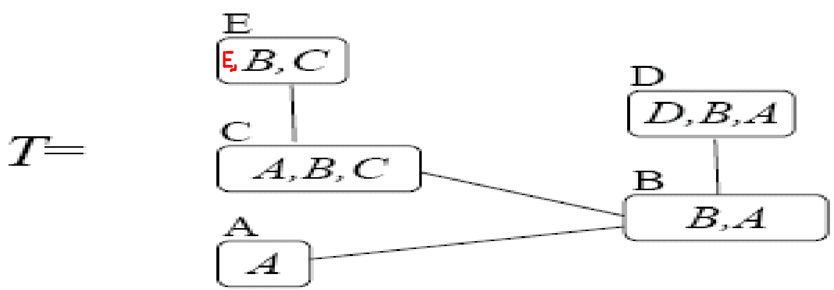
- Let G_d be the induced graph along d. Each variable x and it's earlier neighbors in a node, B_x. There is an arc from B_x to B_y if y is the closest parent of x.

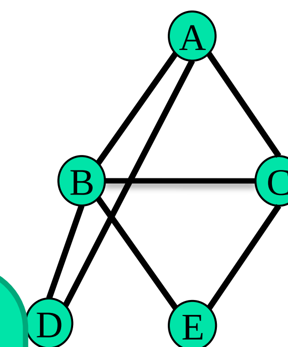
From Bucket-Elimination To Bucket Trees







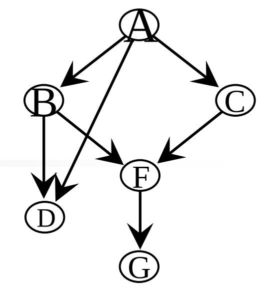




"Moral" graph

•The bucket tree is a tree of cliques of a chordal graph: the induced graph
•P is decomposable relative to the chordal graph. Therefore the tree of cliques is an i-map of P.

From Bucket elimination to bucket-tree elimination



If we want the marginal on D?

Bucket G: $P(G/F)$

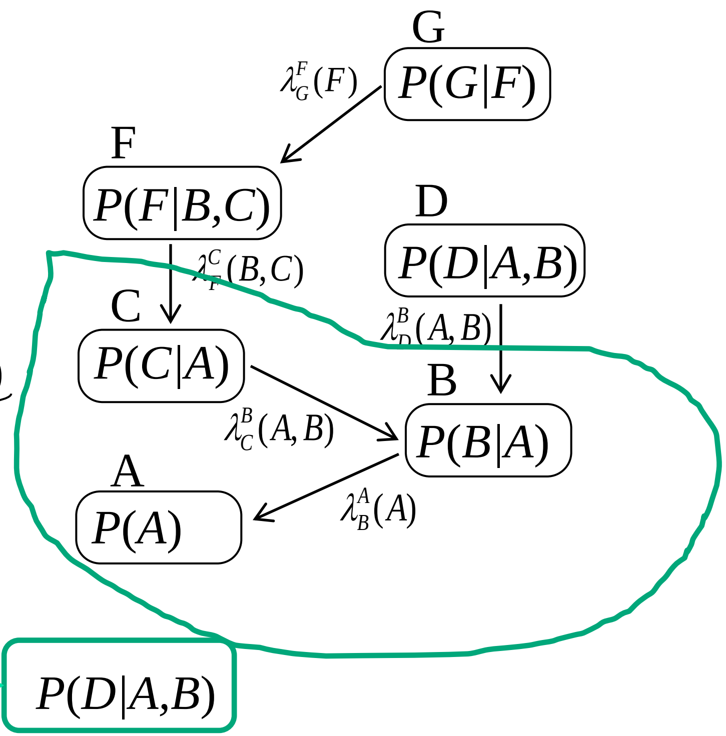
Bucket F: $P(F/B,C)$ $\quad \lambda_G^F(F)$

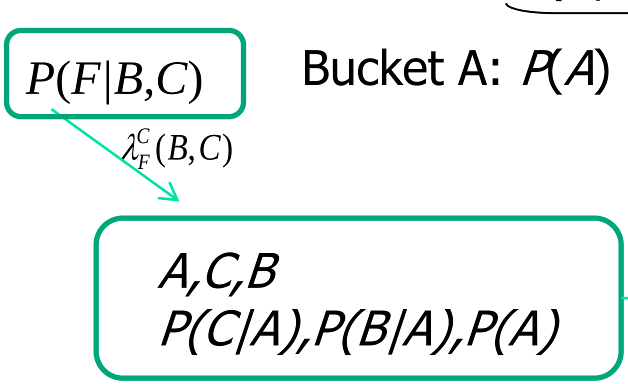
Bucket D: $P(D/A,B)$

Bucket C: $\underbrace{P(C/A) \quad \lambda_F^C(B,C)}$

Bucket B: $\underbrace{P(B/A) \quad \lambda_D^B(A,B) \quad \lambda_C^B(A,B)}$

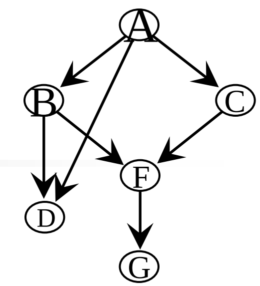
Bucket A: $P(A)$ $\quad \lambda_B^A(A)$





$$\pi_C^D(a,b) = \sum_C P(C \mid A)P(B \mid A)P(A)\lambda_F^C(B,C)$$

From Bucket elimination to bucket-tree elimination



If we want the marginal on C?

Bucket G: $P(G|F)$

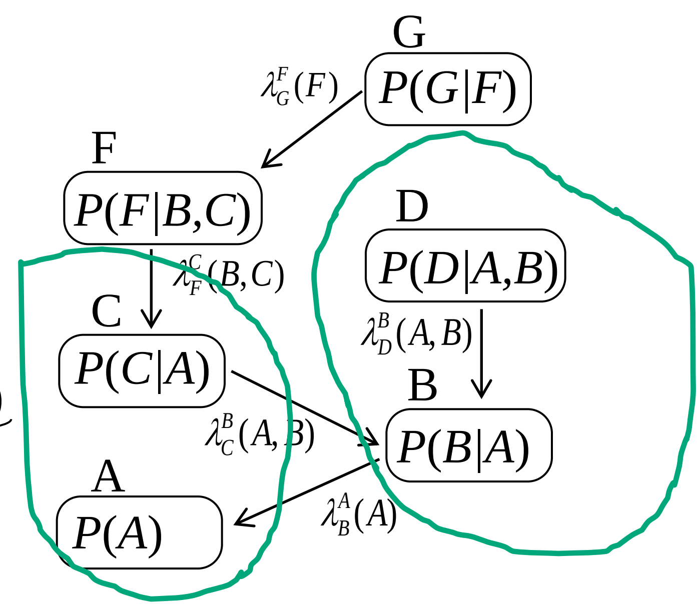
Bucket F: $P(F|B,C)$ $\lambda_G^F(F)$

Bucket D: $P(D|A,B)$

Bucket C: $P(C|A)$ $\lambda_F^C(B,C)$

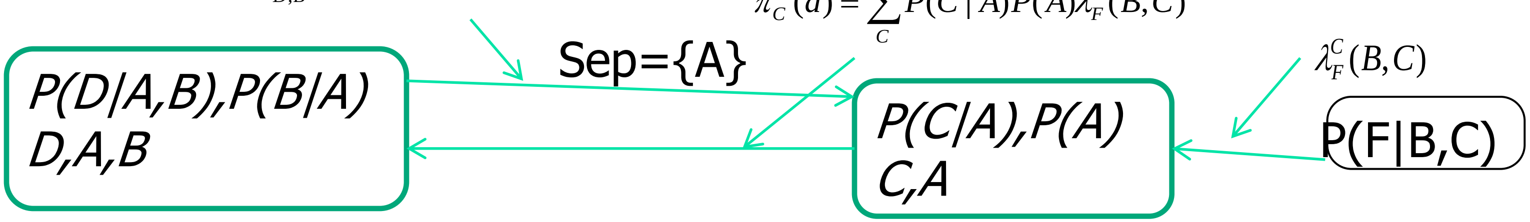
Bucket B: $P(B|A)$ $\lambda_D^B(A,B)$ $\lambda_C^B(A,B)$

Bucket A: $P(A)$ $\lambda_B^A(A)$



$$\pi_D^C(a) = \sum_{D,B} P(D \mid A,B)P(B \mid A)$$

$$\pi_C^D(a) = \sum_{C} P(C \mid A)P(A)\lambda_F^C(B,C)$$



BTE: full Execution

Each bucket can
Compute its
marginal probability

Bucket G: $P(G/F)$ $\quad\quad$ $\prod_F^G(F)$

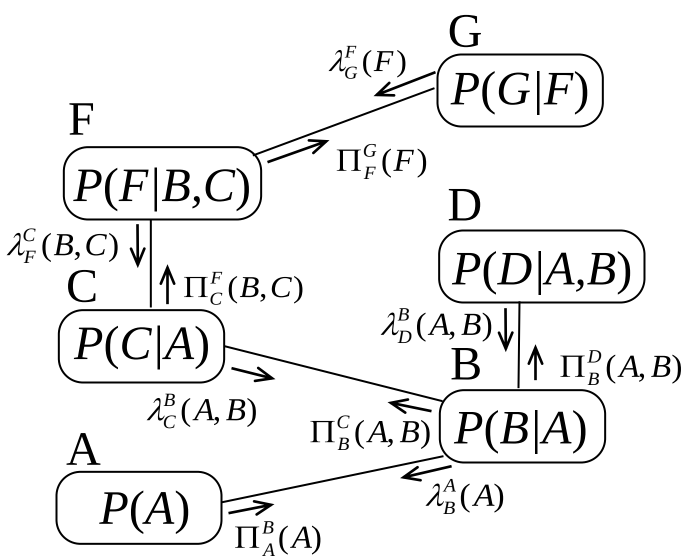
Bucket F: $P(F/B,C)$ $\quad \lambda_G^F(F)$ $\quad\quad$ $\prod_C^F(B,C)$

Bucket D: $P(D/A,B)$ $\quad\quad$ $\prod_B^D(A,B)$

Bucket C: $P(C/A)$ $\quad \lambda_F^C(B,C)$ $\quad\quad$ $\prod_B^C(A,B)$

Bucket B: $P(B/A)$ $\quad \lambda_D^B(A,B)$ $\quad \lambda_C^B(A,B)$ $\quad\quad$ $\prod_A^B(A)$

Bucket A: $P(A)$ $\quad \lambda_B^A(A)$



$$
\begin{aligned}
\pi_A^B(a) &= P(a)\\
\pi_B^C(c,a) &= P(b|a)\lambda_D^B(a,b)\pi_A^B(a)\\
\pi_B^D(a,b) &= P(b|a)\lambda_C^B(a,b)\pi_A^B(a,b)\\
\pi_C^F(c,b) &= \textstyle\sum_a P(c|a)\pi_B^C(a,b)\\
\pi_F^G(f) &= \textstyle\sum_{b,c} P(f|b,c)\pi_C^F(c,b)
\end{aligned}
$$

BTE

Theorem: When BTE terminates

The product of functions in each bucket is the beliefs of the variables joint with the evidence.

Top-down and bottom-up Messages obey same rule: Lets have one name, lambda

Algorithm bucket-tree elimination (BTE)
Input: A problem $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, ordering d.
Output: Augmented buckets containing the original functions and all the π and λ functions received from neighbors in the bucket-tree.
0. Pre-processing:
Place each function in the latest bucket, along d, that mentions a variable in its scope. Connect two buckets B_i and B_j if variable X_j is the latest earlier neighbor of X_i in the induced graph G_d.

1. Top-down phase: λ **messages** (BE)
For $i = n$ to 1, process bucket B_i:
Let $\lambda_{i_1}, ...\lambda_{i_r}$ be all the functions in B_i at the time B_i is processed, including the original functions of F. The message λ_i^j sent from X_i to its child X_j, is computed by

$$\lambda_i^j = \sum_{elim(j,i)} \prod_{k, k \neq j} \lambda_{i_k}$$

2. bottom-up phase: π **messages**
For $j = 1$ to n, process bucket B_j:
Let $\lambda_{j_1}, ..., \lambda_{j_r}$ be all the functions in B_j at the time B_j is processed, including the original functions of F. B_j takes the π message received from its child X_k, π_k^j, and computes a message π_j^i for each child bucket X_j by

$$\pi_j^i = \sum_{elim(j,i)} \pi_k^j \cdot (\prod_{r \neq i} \lambda_r^j)$$

3. Answering singleton queries (e.g., deriving beliefs)
The joint functions $F(B_X)$ in bucket B_X is computed by taking the product of all the functions in B_X (the original fs, the λ functions and π function): Namely, given the functions $f_1, ..., f_t$ in B_X at termination,

$$F(B_X) = \prod_i f_i$$

and the belief of X is computed by

$$Bel(x) = \sum_{B_X - \{X\}} \prod_j f_j$$

Figure 6.2: Algorithm Bucket-Tree Elimination

Bucket tree Elimination (BTE)
Bucket-tree Propagation (BTP)

Bucket-Tree Propagation (BTP)

Input: For each node X_i, its bucket B_i and its neighboring buckets. Let λ_j^i be the message sent to X_i from its neighbor X_j and $f_{i_1}, ..., f_{i_k}$ the original functions in bucket B_i.

The message X_i sends to a neighbor X_j is, once it received all the messages from its neighbors except from X_j is:

$$\lambda_i^j = \sum_{B_i - S(i,j)} (\prod_i f_i) \cdot (\prod_{k \neq j} \lambda_k^i)$$

Figure 6.5: The Bucket-tree propagation (BTP) for X

$$\lambda_i^j = \sum_{elim(j,i)} \prod_{k, k \neq j} \lambda_{ik} \qquad \pi_j^i = \sum_{elim(j,i)} \pi_k^j \cdot (\prod_{r \neq i} \lambda_r^j)$$

Properties of BTE

- Theorem (correctness) 6.1.4 *Algorithm BTE when applied to a Bayesian or Markov network is sound. Namely, in each bucket we can exactly compute the exact joint function of every subset of variables and the evidence.*
- *(follows from imapness of trees)*

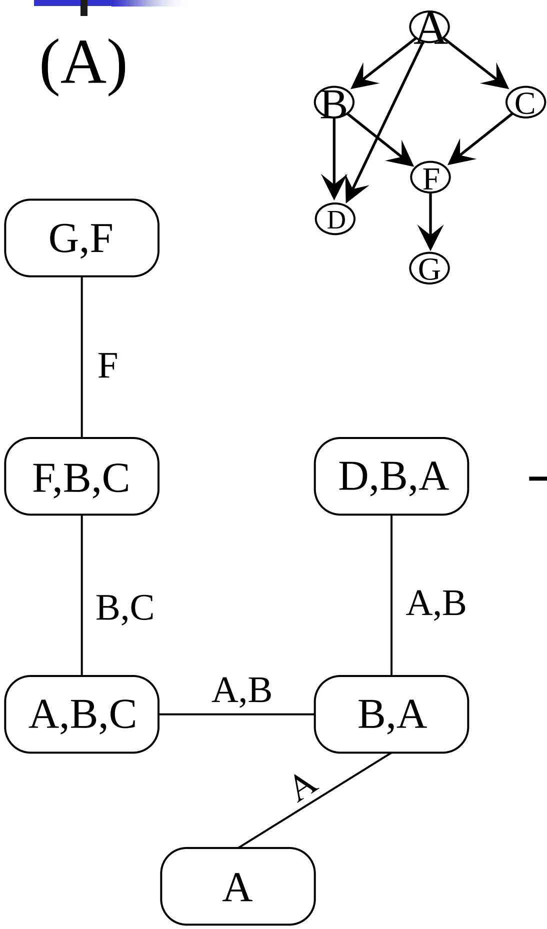
•Theorem 6.1.5 (Complexity of BTE) *Let w* be the induced width of G along ordering d, let r be the number of functions and k the maximum size of a domain of a variable. The time complexity of BTE is O(r deg k^{w*+1}), where deg is the maximum degree in the bucket-tree. The space complexity of BTE is O(n k^w*.)*

From a bucket-tree to a join-tree

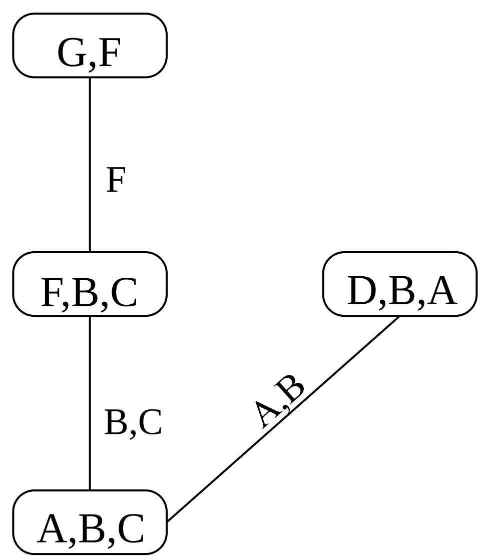
- Merge non-maximal buckets into maximal clusters.
- Connect clusters into a tree: each cluster to one with which it shares a largest subset of variiables.
- Separators are the intersection of variables on the arcs of the tree.
- The cluster tree is an i-map.

From buckets to superbucket to clusters

(A)

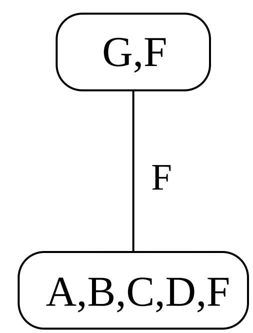


(B)



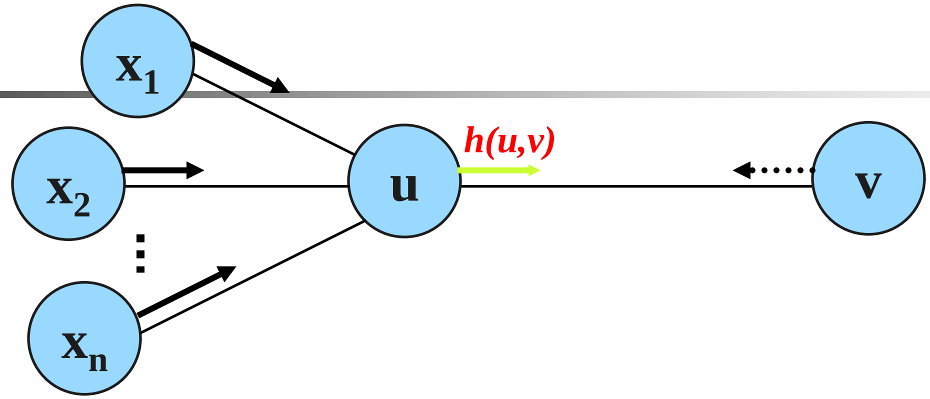
Allows time and space tradeoff

(C)



A super-bucket-tree is an i-map of the Bayesian network

Same Message Passing



$$cluster(u) = \psi(u) \cup \{h(x_1,u), h(x_2,u), \ldots, h(x_n,u), h(v,u)\}$$

Compute the message :

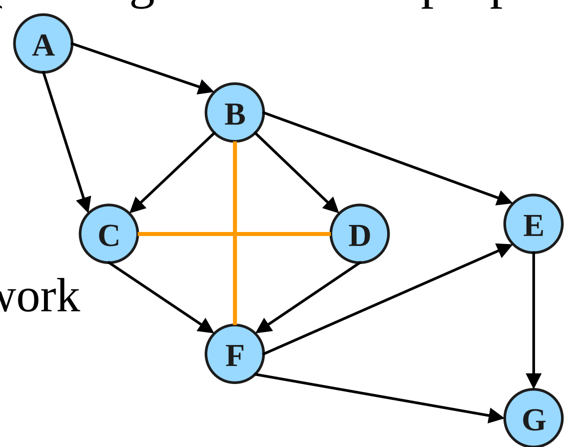
$$h(u,v) = \sum_{elim(u,v)} \prod_{f \in cluster(u) - \{h(v,u)\}} f$$

Elim(u,v) = cluster(u)-sep(u,v)

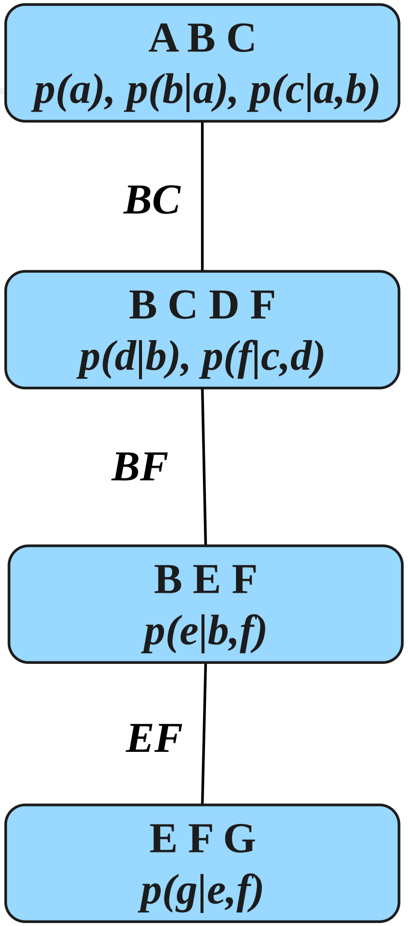
Tree decompositions (more formal)

A *tree decomposition* for a belief network $BN = < X, D, G, P >$ is a triple $< T, \chi, \psi >$, where $T = (V, E)$ is a tree and χ and ψ are labeling functions, associating with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ satisfying :

1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $scope(p_i) \subseteq \chi(v)$
2. For each variable $X_i \in X$ the set $\{v \in V | X_i \in \chi(v)\}$ forms a connected subtree (running intersection property)



Belief network



Tree decomposition

Minimal Tree-Decompositions

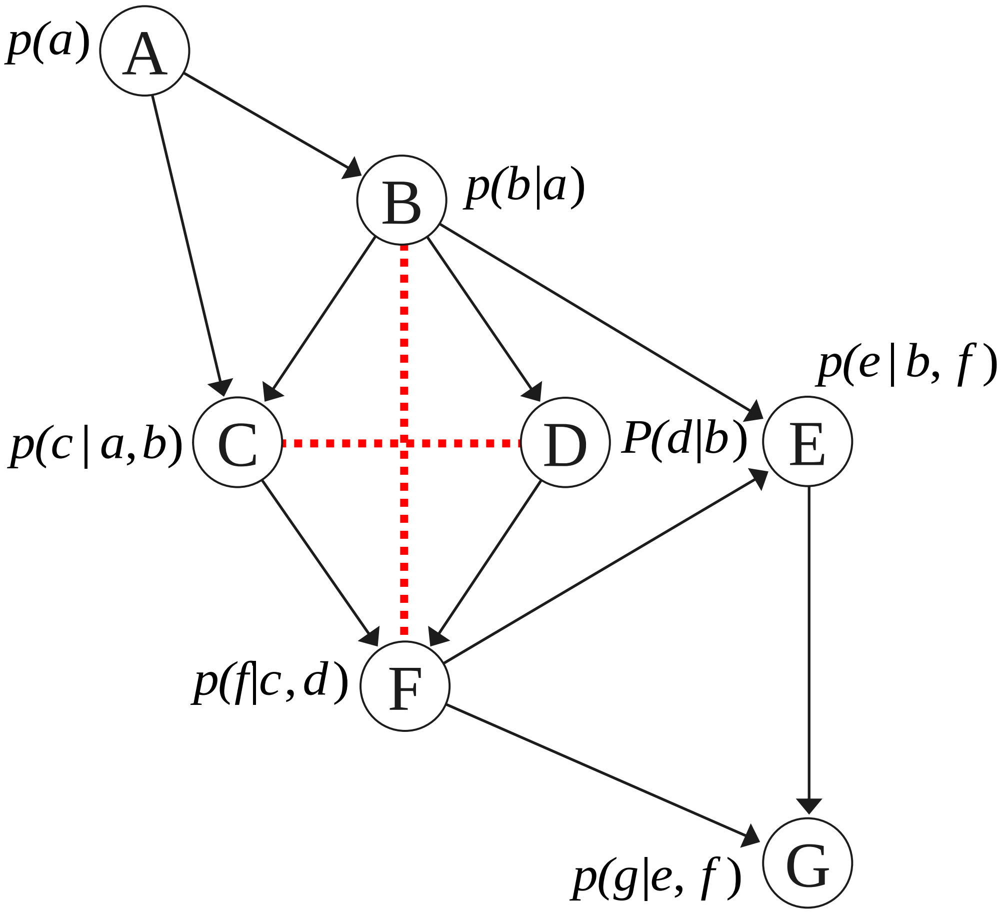
Notice that it may be that $sep(u, v) = \chi(u)$ (that is, all variables in vertex u belong to an adjacent vertex v). In this case the size of the tree-decomposition can be reduced by merging vertex u into v without increasing the tree-width of the tree-decomposition.

Definition 6.2.7 (minimal tree-decomposition) *A tree-decomposition is minimal if* $sep(u, v) \subset \chi(u)$ *and* $sep(u, v) \subset \chi(v)$.

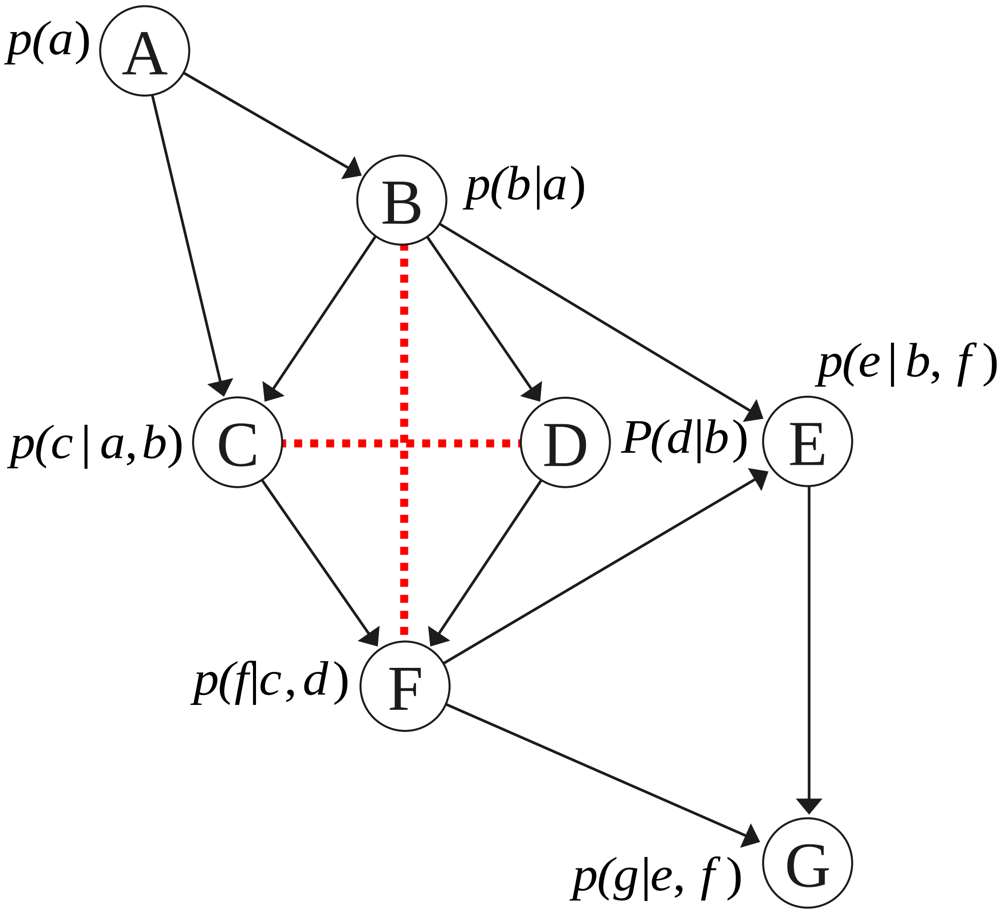
We immediately can observe that the bucket-tree is often not minimal. We can make it minimal however, by having each subsumed bucket be absorbed into its containing bucket, yielding *super-bucket* tree.

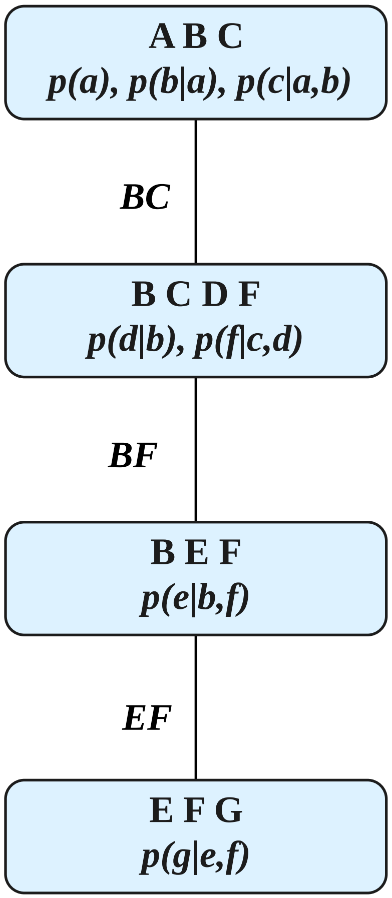
Minimal tree-decompositions are called Join-trees or junction-trees

Tree Decomposition for belief updating

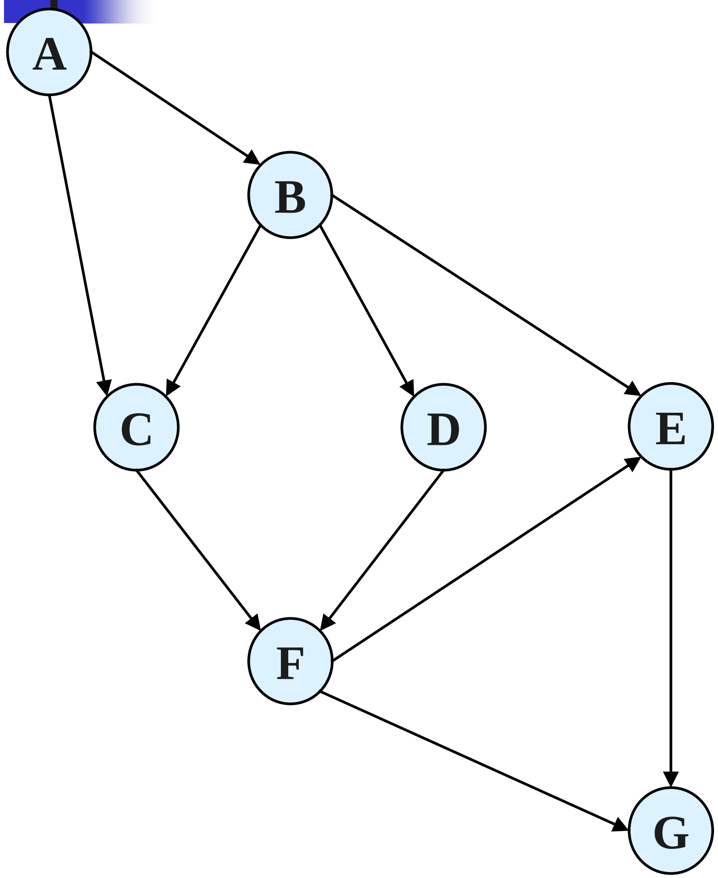


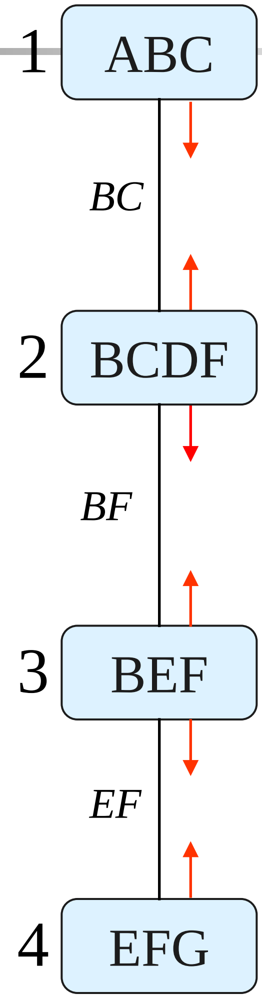
Tree Decomposition for belief updating





CTE: Cluster Tree Elimination





$$h_{(1,2)}(b,c) = \sum_{a} p(a) \cdot p(b \mid a) \cdot p(c \mid a,b)$$

$$h_{(2,1)}(b,c) = \sum_{d,f} p(d \mid b) \cdot p(f \mid c,d) \cdot h_{(3,2)}(b,f)$$

$$h_{(2,3)}(b,f) = \sum_{c,d} p(d \mid b) \cdot p(f \mid c,d) \cdot h_{(1,2)}(b,c)$$

$$h_{(3,2)}(b,f) = \sum_{e} p(e \mid b,f) \cdot h_{(4,3)}(e,f)$$

$$h_{(3,4)}(e,f) = \sum_{b} p(e \mid b,f) \cdot h_{(2,3)}(b,f)$$

$$h_{(4,3)}(e,f) = p(G = g_e \mid e,f)$$

Time: ***O (exp(w+1))***
Space: ***O (exp(sep))***

For each cluster P(X|e) is computed, also P(e)

Algorithm cluster-tree elimination (CTE)

Input: A tree decomposition $< T, \chi, \psi >$ for a problem $M =< X, D, F, \prod\} >$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$.

Output: An augmented tree whose vertices are clusters containing the original functions as well as messages received from neighbors. A solution computed from the augmented clusters.

Compute messages:

For every edge (u, v) in the tree, do

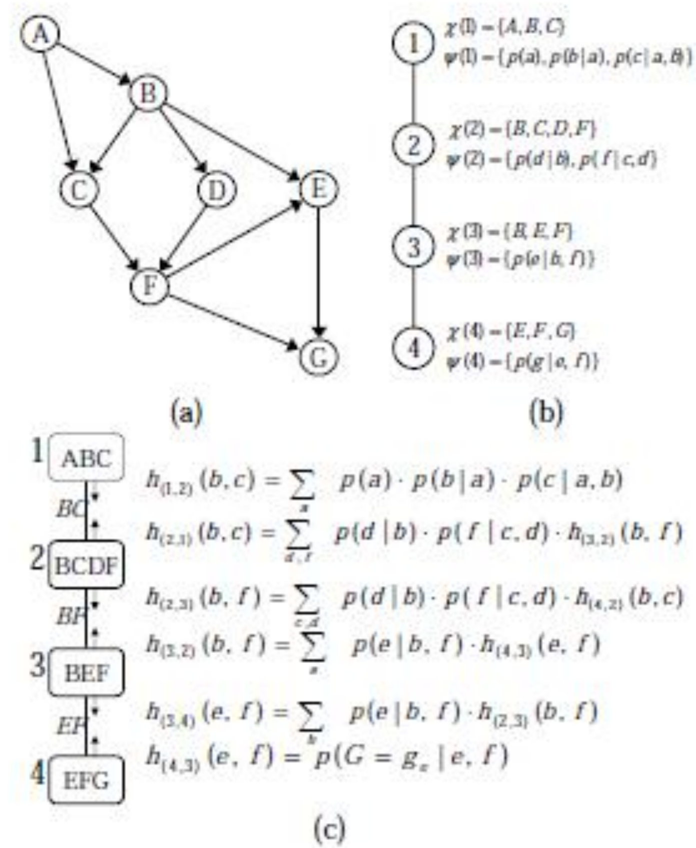
- Let $m_{(u,v)}$ denote the message sent by vertex u to vertex v.
- Let $cluster(u) = \psi(u) \cup \{m_{(i,u)} | (i, u) \in T\}$.
- If vertex u has received messages from all adjacent vertices other than v, then compute and send to v,

$$m_{(u,v)} = \sum_{sep(u,v)} \Big(\prod_{f \in cluster(u), f \neq m_{(v,u)}} f \Big)$$

Endfor

Note: functions whose scope does not contain elimination variables do not need to be processed, and can instead be directly passed on to the receiving vertex.

Return: A tree-decomposition augmented with messages, and for every $v \in T$



Let Ci and Cj two adjacent clusters and sep(i,j) be their separator

$$bel(sep) = \sum_{elim(i,j)} \prod_{f \in C_i} f = \sum_{elim(j,i)} \prod_{f \in C_j} f$$

CTE - properties

- Correctness and completeness: Algorithm CTE is correct, i.e. it computes the exact joint probability in each cluster and therefore of every single variable and the evidence.

- Time complexity: $O\ (\ deg \times (n+N) \times k^{\,w^*+1}\ )$

- Space complexity: $O\ (\ N \times k^{\,sep}\ )$

where
deg = the maximum degree of a node in the cluster-tree
n = number of variables (= number of CPTs)
N = number of nodes in the tree decomposition
k = the maximum domain size of a variable
*w** = the induced width
sep = the separator size

Treewidth & Separator

The *width* (also called tree-width) of a tree-decomposition $< T, \chi, \psi >$ is $\max_{v \in V} |\chi(v)|$, Given two adjacent vertices u and v of a tree-decomposition, a separator of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$.

Good Tree-width can be generated using good induced-width ordering heuristics

GRAPH TRIANGULATION (FILL-IN) ALGORITHM: Tarjan and Yannakakis [1984]

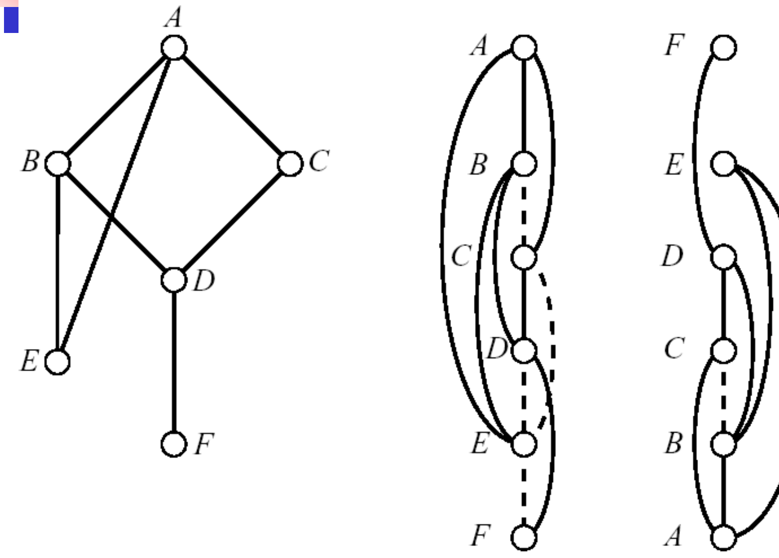
1. Compute an ordering for the nodes, using a *maximum cardinality search*, i.e., number vertices from 1 to $|V|$, in increasing order, always assigning the next number to the vertex having the largest set of previously numbered neighbors (breaking ties arbitrarily).
2. From $n = |V|$ to $n = 1$, recursively fill in edges between any two nonadjacent parents of n, i.e., neighbors of n having lower ranks than n (including neighbors linked to n in previous steps). If no edges are added the graph is chordal; otherwise, the new filled graph is chordal.

- Given a graph $G = (V, E)$ we can construct a join tree using the following procedure.

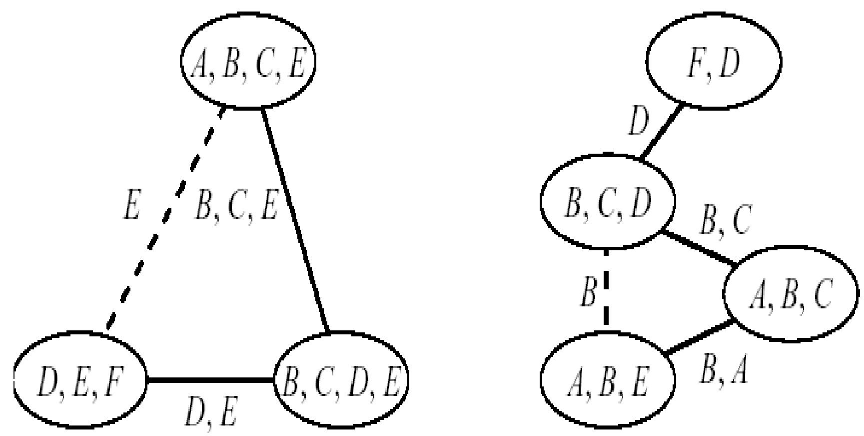
ASSEMBLING A JOIN TREE

1. Use the fill-in algorithm to generate a chordal graph G' (if G is chordal, $G = G'$).
2. Identify all cliques in G'. Since any vertex and its parent set (lower ranked nodes connected to it) form a clique in G', the maximum number of cliques is $|V|$.
3. Order the cliques $C_1, C_2, \ldots, C_t$ by rank of the highest vertex in each clique.
4. Form the join tree by connecting each C_i to a predecessor C_j ($j < i$) sharing the highest number of vertices with C_i.

Example of tree-clustering

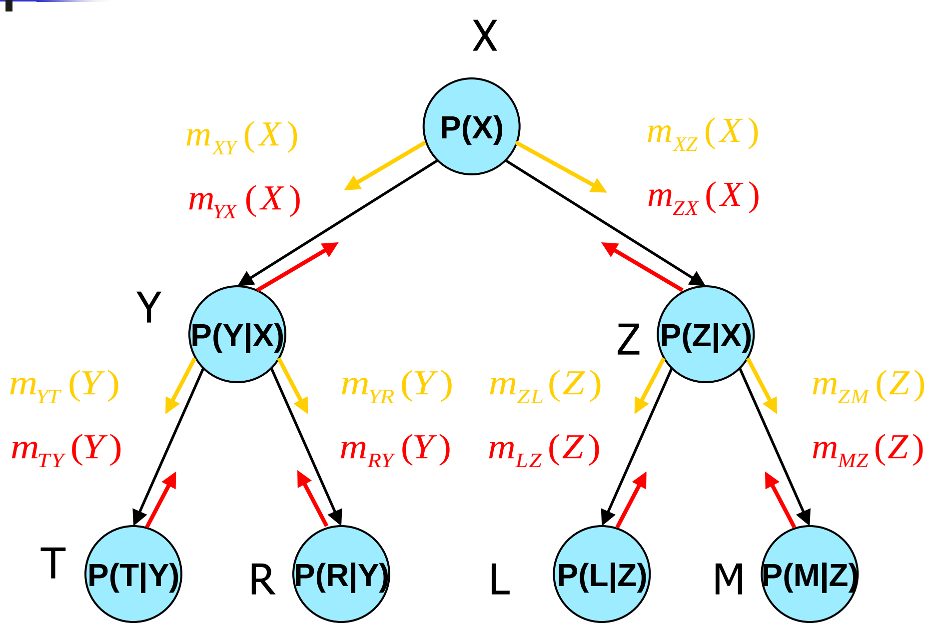


(a) (b) (c)



(a) (b)

Inference on trees is easy and distributed



$$m_{MZ}(Z) = \sum_{M} P(M \mid Z)$$

$$m_{LZ}(Z) = \sum_{L} P(L \mid Z)$$

$$m_{ZX}(X) = \sum_{Z} P(Z \mid X) \cdot m_{MZ}(Z) \cdot m_{LZ}(Z)$$

$$m_{XZ}(X) = P(X) \cdot m_{YX}(X)$$

$$m_{ZL}(Z) = \sum_{X} P(Z \mid X) \cdot m_{XZ}(X) \cdot m_{MZ}(Z)$$

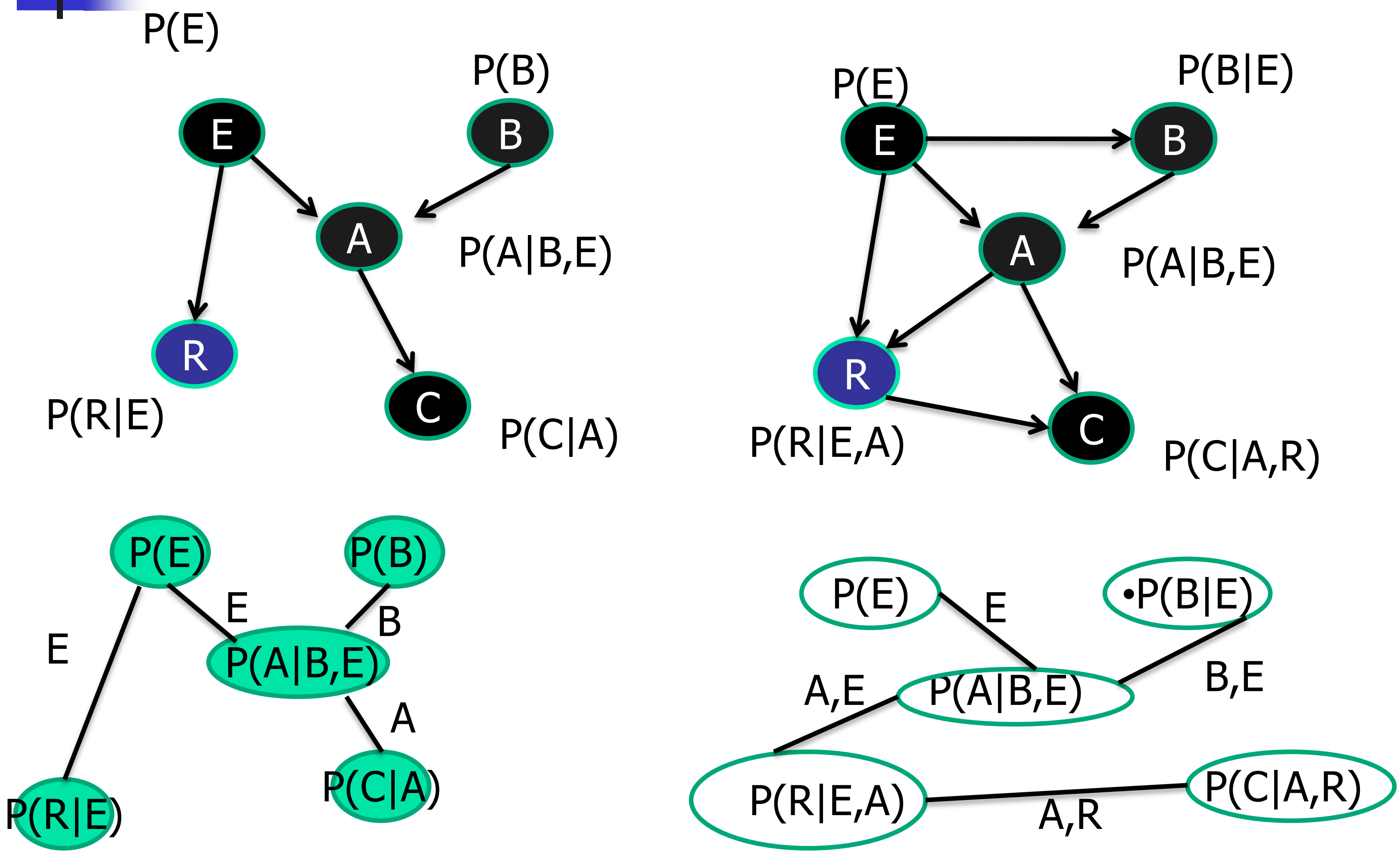
$$m_{ZM}(Z) = \sum_{X} P(Z \mid X) \cdot m_{XZ}(X) \cdot m_{LZ}(Z)$$

Belief updating = sum-prod

A tree of binary functions
Is a chordal graph with clusters of size 2

Inference is time and space linear on trees

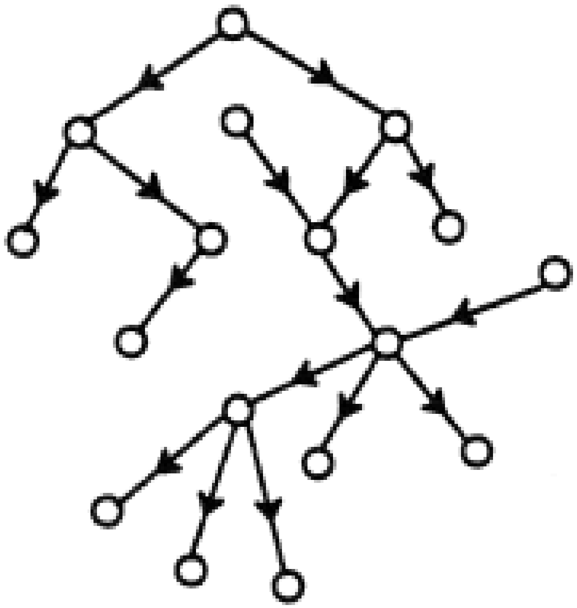
Acyclic-Networks: Belief Propagation is easy on



Polytrees and Acyclic networks

- **Polytree:** a BN whose undirected skeleton is a tree
- **Acyclic network**: A network is acyclic if it has a tree-decomposition where each node has a single original CPT.
- Dual network: each scope-cpt is a node and each arc is denoted by intersection.
- Acylic network (alternative definition): when the dual graph has a join-tree
- BP is exact on an acyclic network.
- Tree-clustering converts a network into an acyclic one.

A Glimpse into Pearl's BP



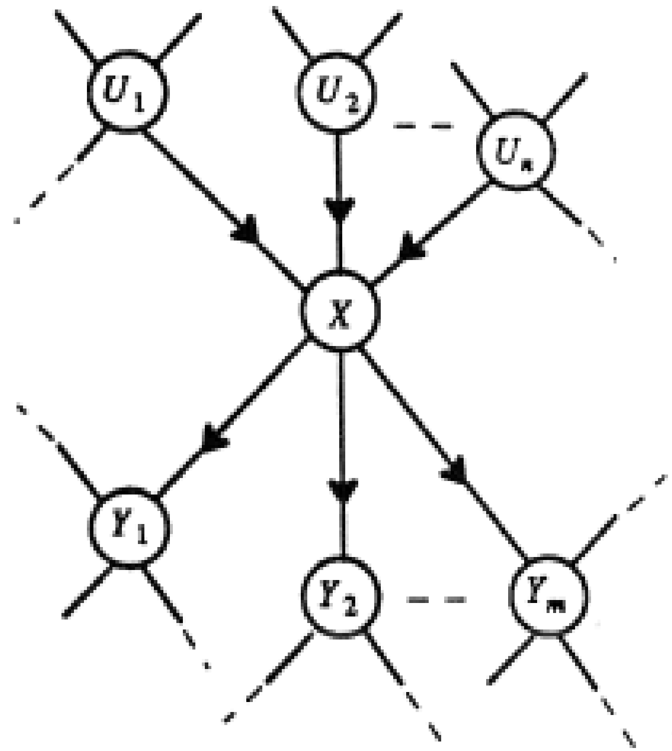


Figure 4.18. (a) A fragment of a polytree and (b) the parents and children of a typical node X.

EVIDENCE DECOMPOSITION

$e^-_{XY_j}$ stands for evidence contained in the subnetwork on the *head* side of the link $X \rightarrow Y_j$,

$e^+_{U_iX}$ stands for evidence contained in the subnetwork on the *tail* side of the link $U_i \rightarrow X$.

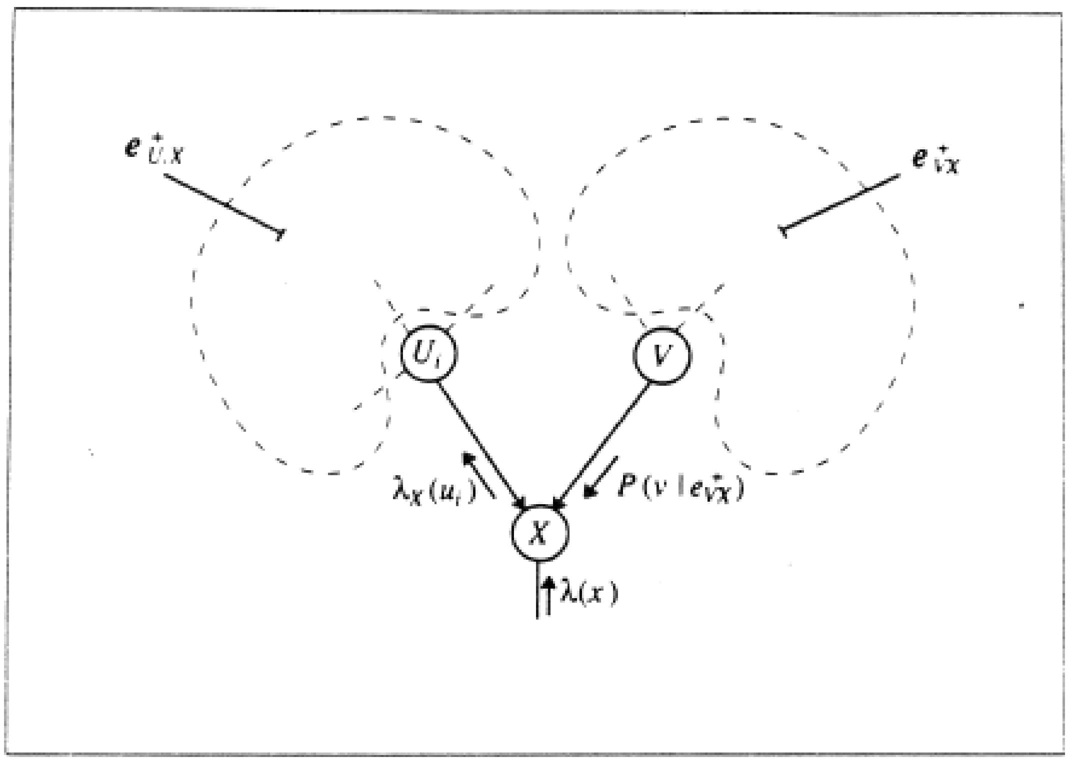


Figure 4.19. Variables, messages, and evidence sets used in the derivation of $\lambda_X(u_i)$.

Step 1 - Belief updating: When node X is activated, it simultaneously inspects the messages $\pi_X(u_i)$, $i = 1, \ldots, n$ communicated by its parents and the messages $\lambda_{Y_j}(x)$, $j = 1, \ldots, m$ communicated by its children. Using this input, it updates its belief measure to

$$BEL(x) = \alpha \, \lambda(x) \, \pi(x), \tag{4.49}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x), \tag{4.50}$$

$$\pi(x) = \sum_{u_1, \ldots, u_n} P(x \mid u_1, \ldots, u_n) \prod_i \pi_X(u_i), \tag{4.51}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.

Step 2 - Bottom-up propagation: Using the messages received, node X computes new λ messages to be sent to its parents. For example, the new message $\lambda_X(u_i)$ that X sends to its parents U_i is computed by

$$\lambda_X(u_i) = \beta \sum_x \lambda(x) \sum_{u_k : k \neq i} P(x \mid u_1, \ldots, u_n) \prod_{k \neq i} \pi_X(u_k) . \quad (4.52)$$

Step 3 - Top-down propagation: Each node computes new π messages to be sent to its children. For example, the new $\pi_{Y_j}(x)$ message that X sends to its child Y_j is computed by

$$\pi_{Y_j}(x) = \alpha \left[\prod_{k \neq j} \lambda_{Y_k}(x) \right] \sum_{u_1, \ldots, u_n} P(x \mid u_1, \ldots, u_n) \prod_i \pi_X(u_i) \quad (4.53)$$

$$= \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)} .$$

SUMMARY OF PROPAGATION RULES FOR POLYTREES

- The steps involved in polytree propagation are similar to those used with trees. We shall now summarize these steps by considering a typical node X having m children, $Y_1, \ldots, Y_m$, and n parents, $U_1, \ldots, U_n$, as in Figure 4.18*b*.

- The belief distribution of variable X can be computed if three types of parameters are made available:

1. The current strength of the *causal* support π contributed by each incoming link $U_i \rightarrow X$:

$$\pi_X(u_i) = P(u_i \mid e^+_{U_i X}) . \tag{4.47}$$

2. The current strength of the *diagnostic* support, λ, contributed by each outgoing link $X \rightarrow Y_j$:

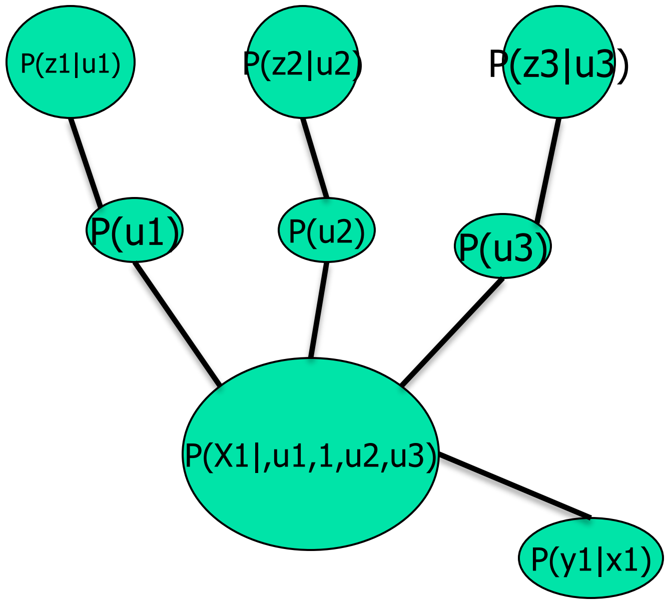
$$\lambda_{Y_j}(x) = P(e^-_{XY_j} \mid x) . \tag{4.48}$$

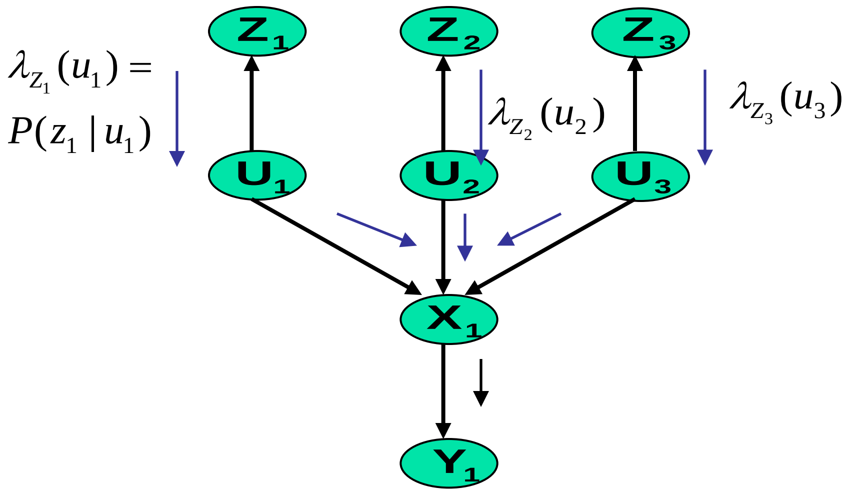
3. The fixed conditional-probability matrix $P(x \mid u_1, \ldots, u_n)$ that relates the variable X to its immediate parents.

Belief propagation is easy on polytree: Pearl's Belief Propagation

A polytree: a tree with
Larger families

A polytree decomposition

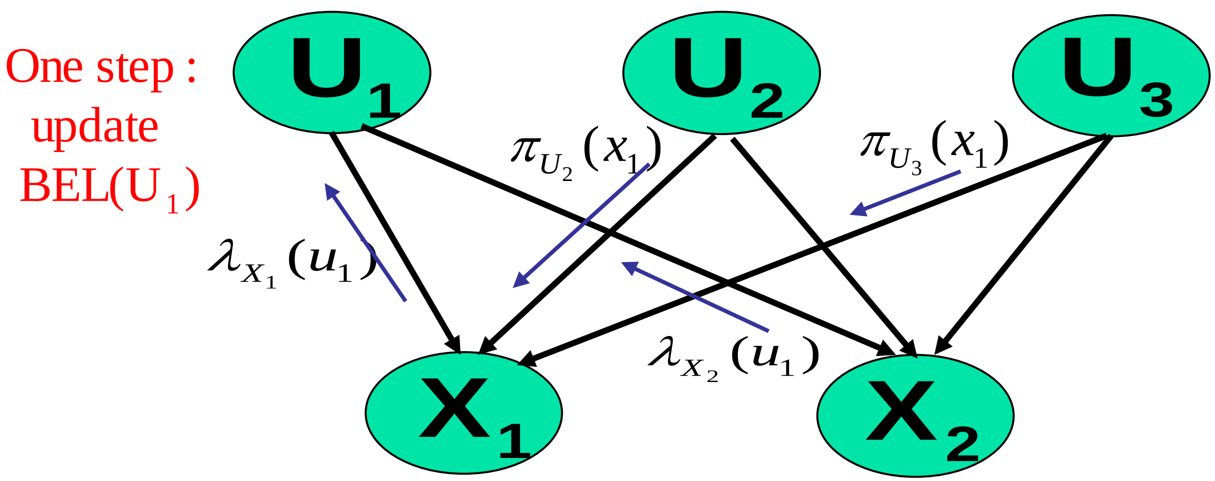




Running CTE = running Pearl's BP over the dual graph
Dual-graph: nodes are cpts, arcs connect non-empty intersections.
BP is Time and space linear

From exact to approximate: Iterative Belief Propagation

- Belief propagation is exact for poly-trees
- IBP - applying BP iteratively to cyclic networks



- No guarantees for convergence
- Works well for many coding networks

Dual graphs, join-graphs

Definition 6.4.5 (dual graphs, join dual graphs, arc-minimal dual-graphs) *Given a graphical model* $\mathcal{M} =< X, D, F, \prod >$.

- *The* dual graph $\mathcal{D}_{\mathcal{F}}$ *of the graphical model* $\mathcal{M}$*, is an arc-labeled graph defined over the its functions. Namely, it has a node for each function labeled with the function's scope and a labeled arc connecting any two nodes that share a variable in the function's scope. The arcs are labeled by the shared variables.*

- *A* dual join-graph *is a labeled arc subgraph of* $\mathcal{D}_{\mathcal{F}}$ *whose arc labels are subsets of the labels of* $\mathcal{D}_{\mathcal{F}}$ *such that the* running intersection property*, is satisfied.*

- *An* arc-minimal *dual join-graph is a dual join-graph for which none of its labels can be further reduced while maintaining the connectedness property.*

Dual join-graphs examples

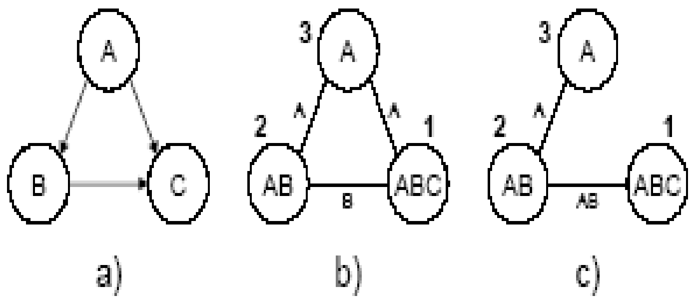


Figure 6.13: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

Proposition 6.4.6 *The dual graph of any Bayesian network has an arc-minimal dual join-graph where each arc is labeled by a single variable.*

Iterative Belief propagation

Algorithm IBP

Input: An arc-labeled dual join-graph $DJ = (V, E, L)$ for a graphical model $\mathcal{M} =< X, D, F, \prod >$.

Output: An augmented graph whose nodes include the original functions and the messages received from neighbors. Denote by: h_u^v the message from u to v; $ne(u)$ the neighbors of u in V; $ne_v(u) = ne(u) - \{v\}$; l_{uv} the label of $(u, v) \in E$; $elim(u, v) = scope(u) - scope(v)$.

- **One iteration of IBP**

  For every node u in DJ in a topological order and back, do:

  1. **Process observed variables**

     Assign evidence variables to the each p_i and remove them from the labeled arcs.

  2. **Compute and send to v the function:**

$$h_u^v = \sum_{elim(u,v)} (p_u \cdot \prod_{\{h_i^u, i \in ne_v(u)\}} h_i^u)$$

  Endfor

- **Compute approximations of $P(F_i|e)$, $P(X_i|e)$:**

  For every $X_i \in X$ let u be the vertex of family F_i in DJ,

  $P(F_i|e) = \alpha(\prod_{h_i^u, u \in ne(i)} h_i^u) \cdot p_u$;

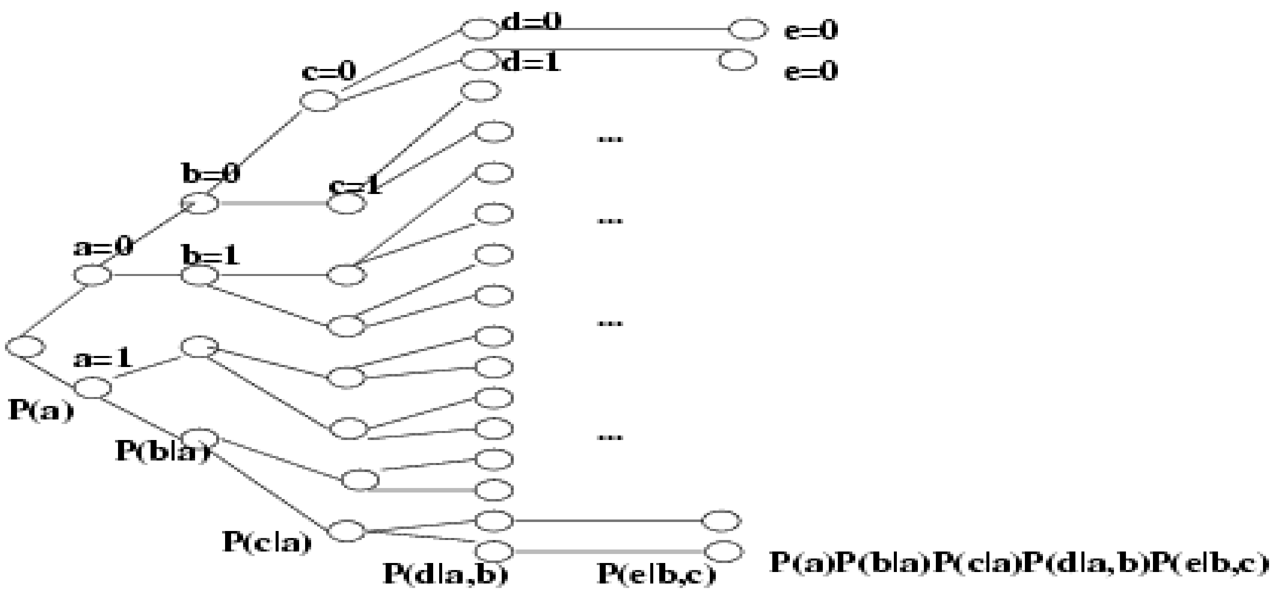
  $P(X_i|e) = \alpha \sum_{scope(u)-\{X_i\}} P(F_i|e)$.

Exact Reasoning by Search

- Why consider search?
- Can we do any better in search?
- Can we combine search and inference?

Conditioning generates the probability tree

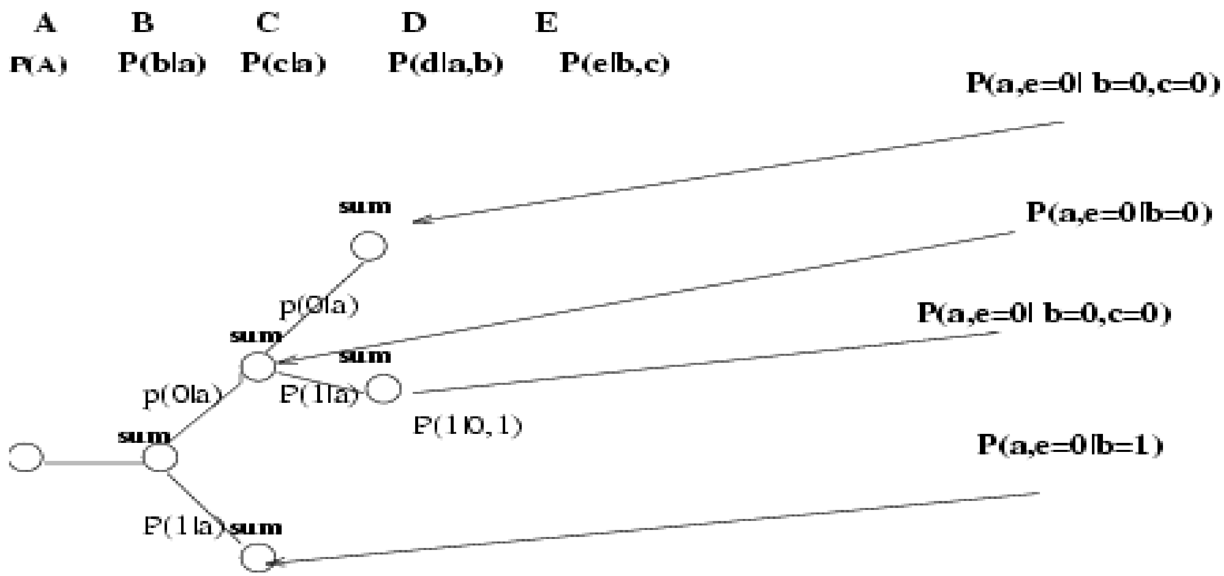
$$P(a, e=0) = P(a)\sum_b P(b \mid a)\sum_c P(c \mid a)\sum_b P(d \mid a, b)\sum_{e=0} P(e \mid b, c)$$



Complexity of conditioning: exponential time, linear space

Conditioning+Elimination

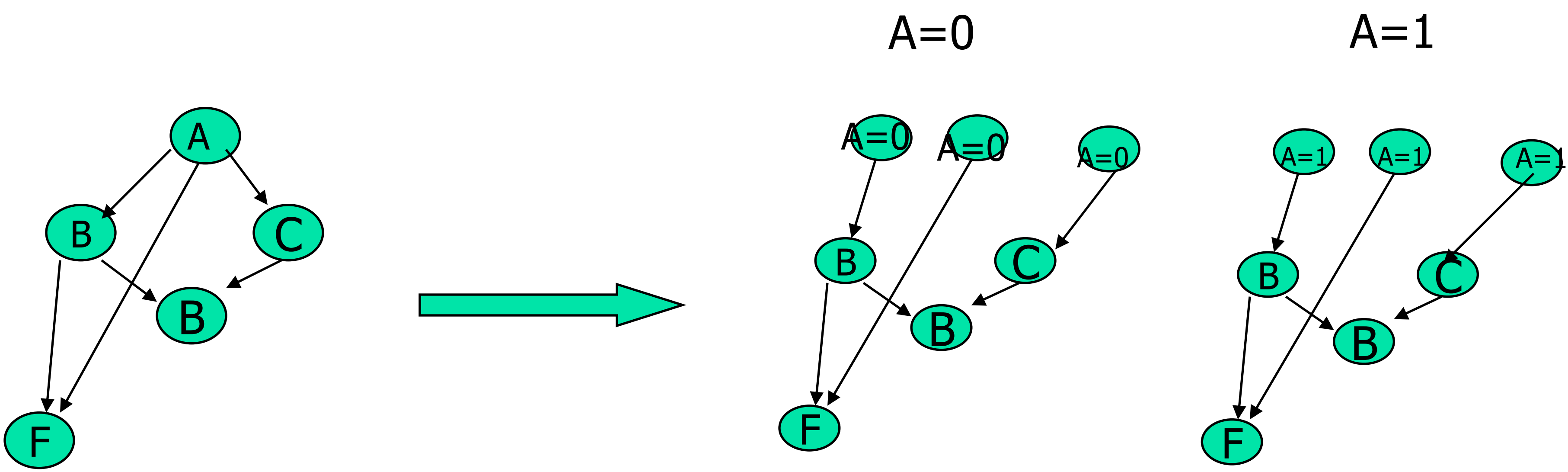
$$P(a,e=0)=P(a)\sum_b P(b|a)\sum_c P(c|a)\sum_d P(d|a,b)\sum_{e=0} P(e|b,c)$$



Idea: conditioning until w^* of a (sub)problem gets small

Loop-cutset decomposition

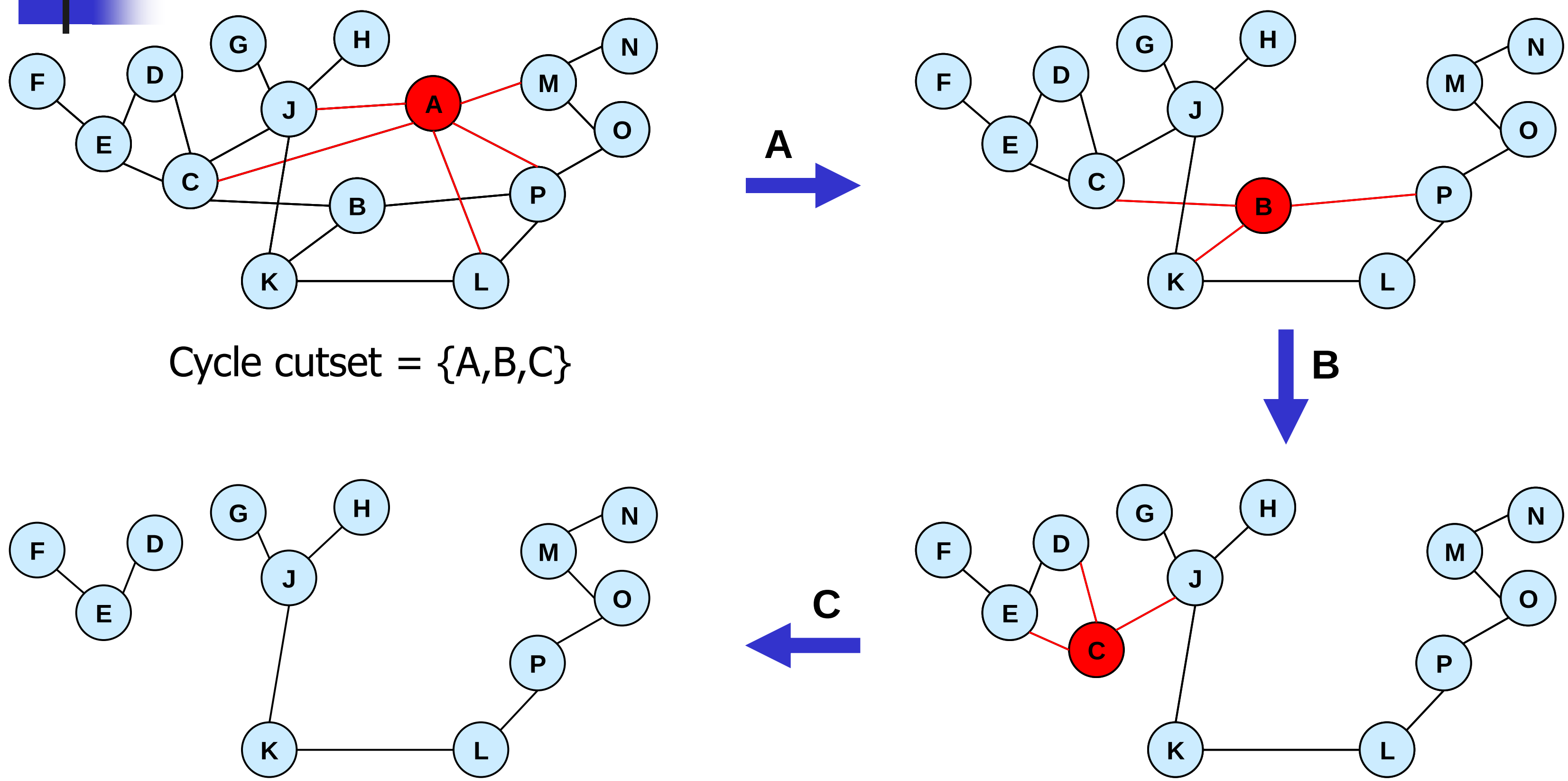
- You condition until you get a polytree



P(B|F=0) = P(B, A=0|F=0)+P(B,A=1|F=0)

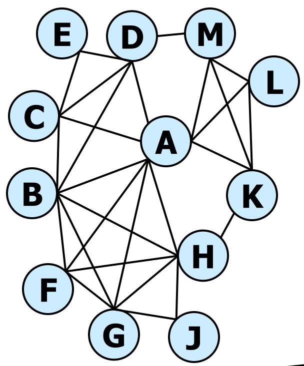
Loop-cutset method is time exp in loop-cutset size and linear space. For each cutset we can do BP

Conditioning and Cycle cutset

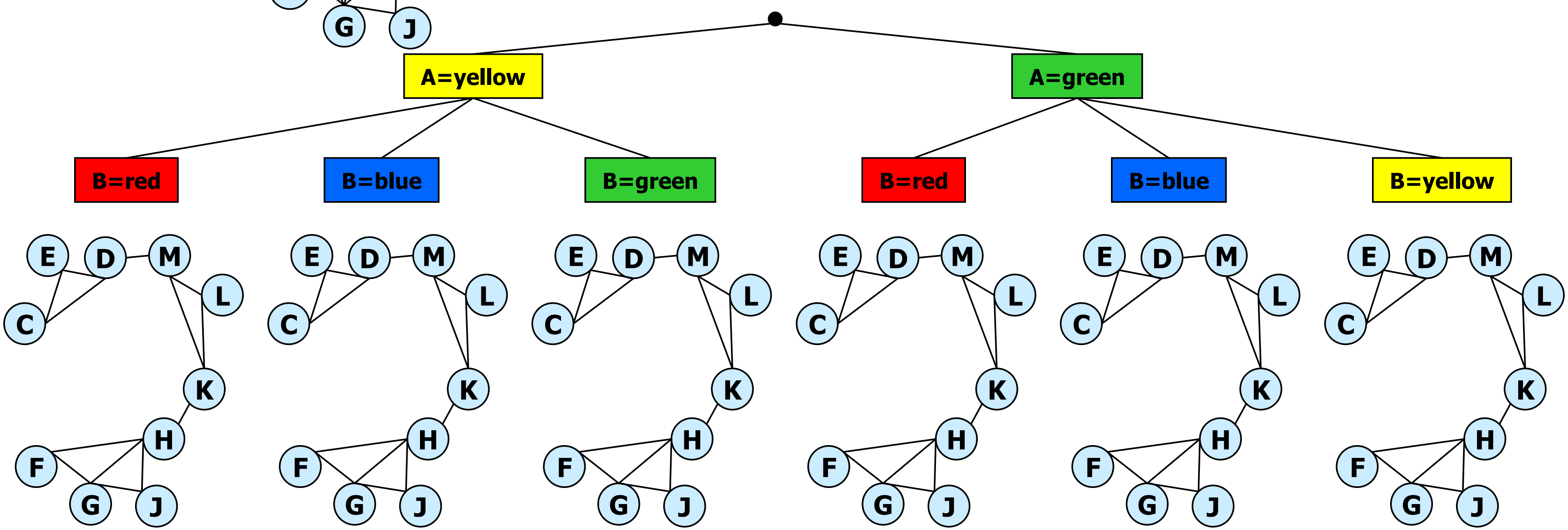


Search over the Cutset (cont)

Graph Coloring problem



- Inference may require too much memory
- **Condition** on some of the variables



Variable elimination with conditioning; w-cutset algorithms

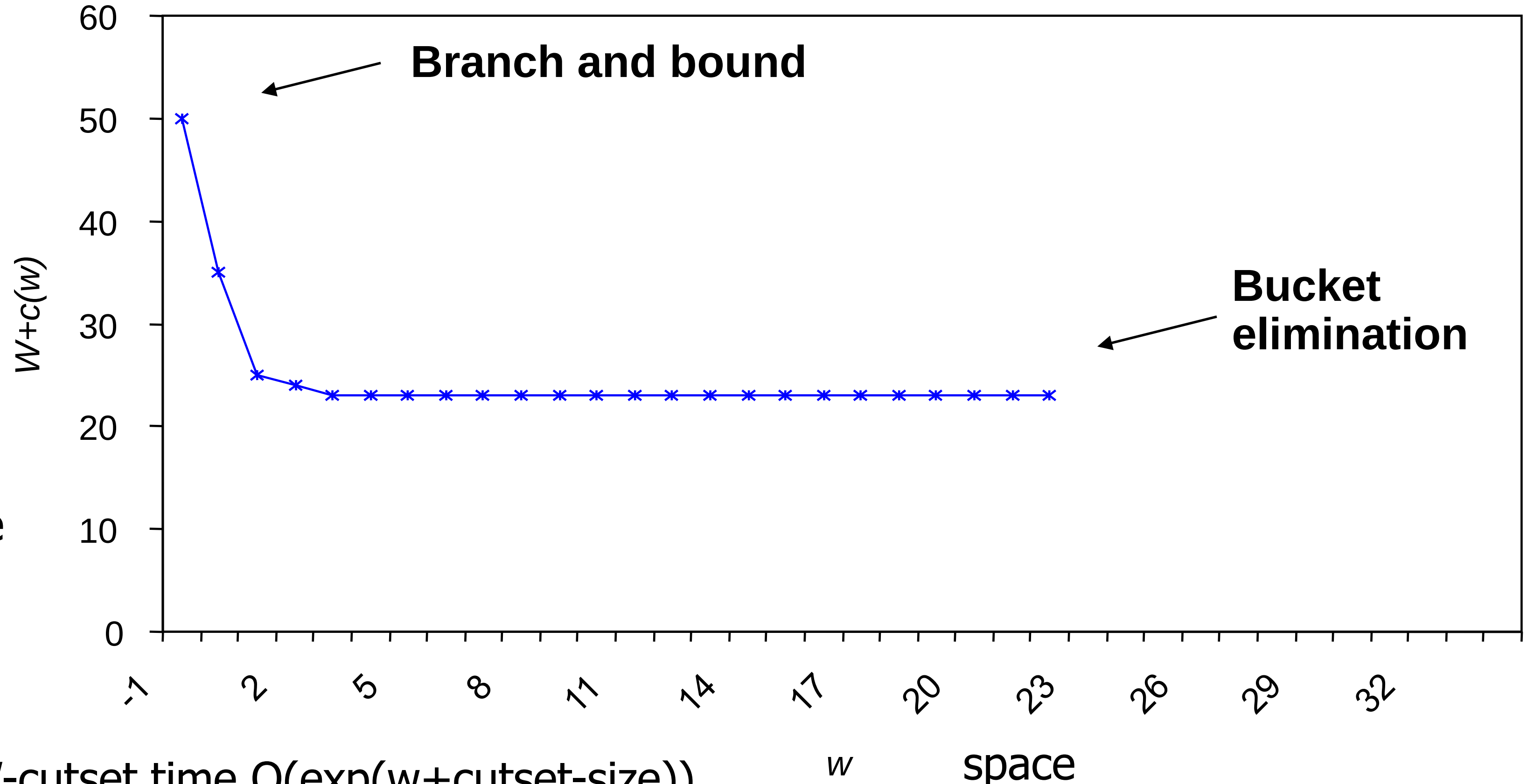
- VEC-bel:
- Identify a w-cutset, c_w, of the network
- For each assignment to the cutset solve by CTE the conditioned sub-problem
- Aggregate the solutions over all cutset assignments.
- Time complexity: exp(|C_w|+w)
- Space complexity: exp(w)

Time vs Space for w-cutset

(Dechter and El-Fatah, 2000)
(Larrosa and Dechter, 2001)
(Rish and Dechter 2000)

- **Random Graphs (50 nodes, 200 edges, average degree 8, *w*≈23*)**

time



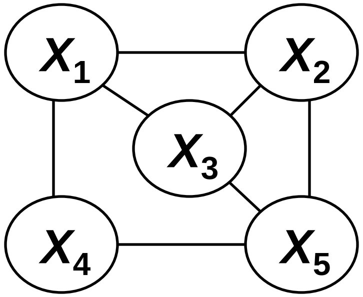
space

W-cutset time O(exp(w+cutset-size))
Space O(exp(w))

Hybrid of Variable-elimination and Search

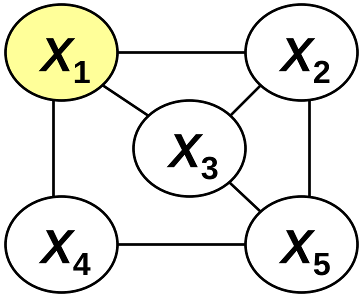
- Tradeoff space and time

Search Basic Step: Conditioning

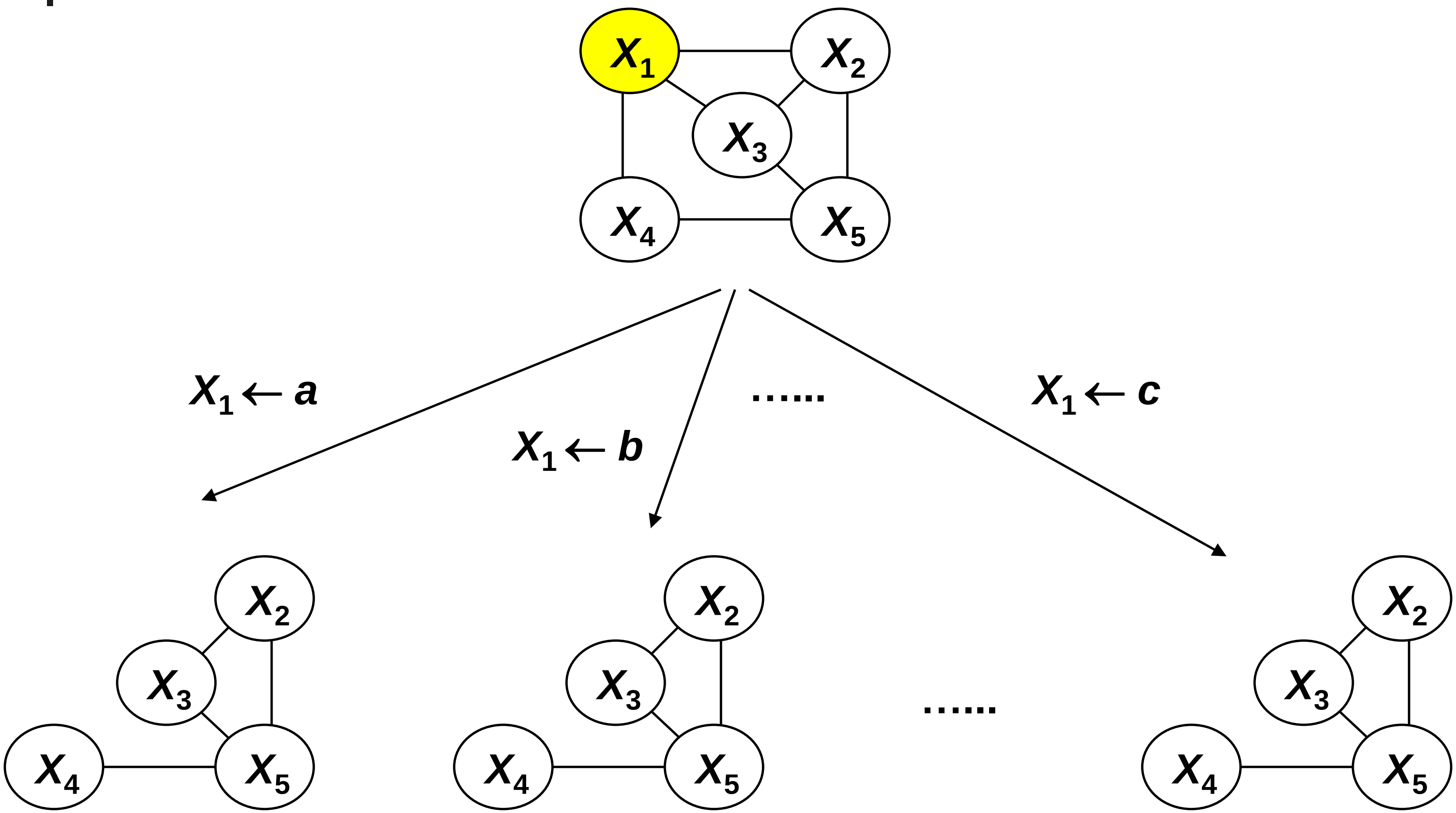


Search Basic Step: Conditioning

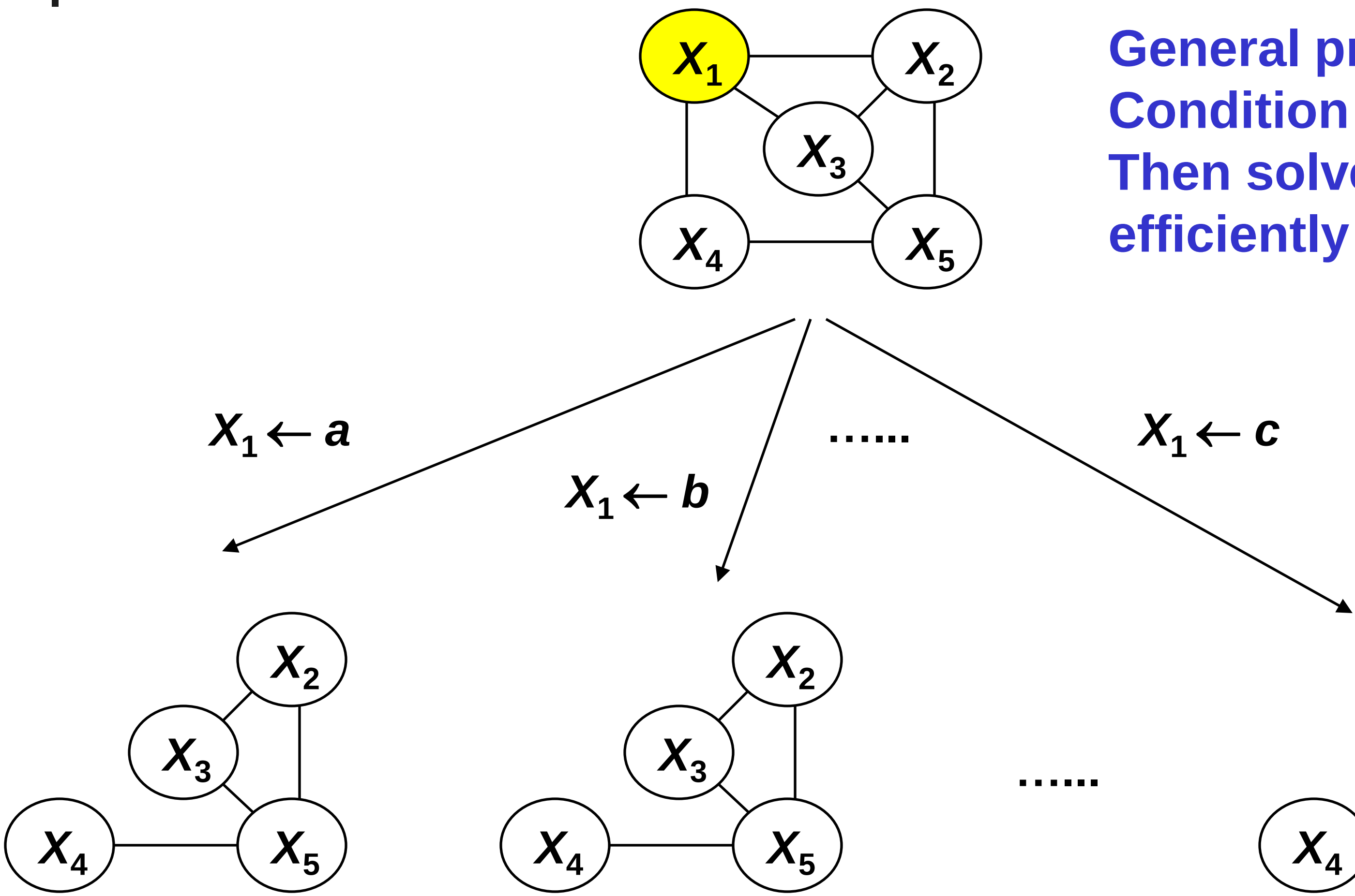
- **Select a variable**



Search Basic Step: Conditioning

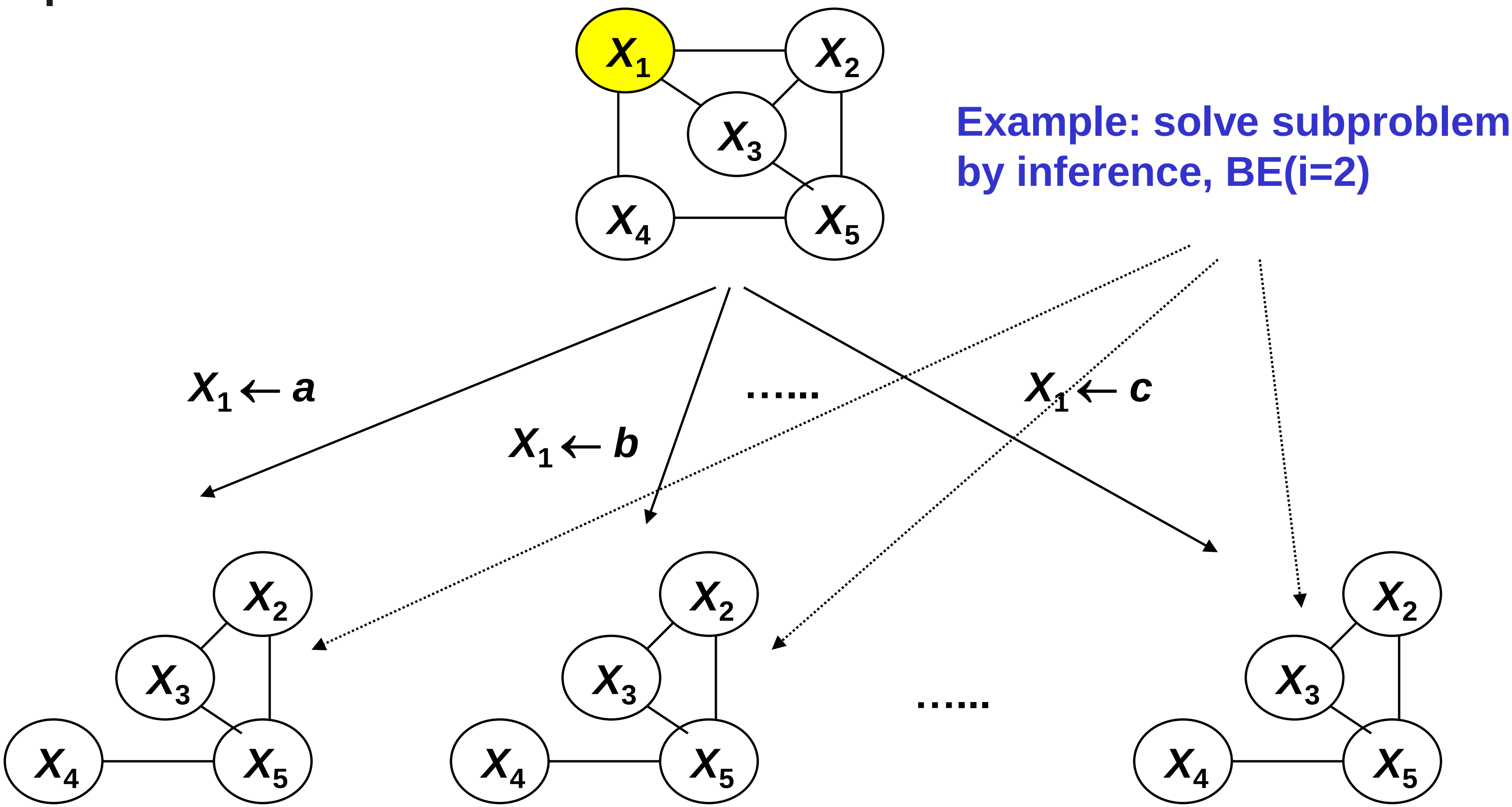


Search Basic Step:
Variable Branching by Conditioning



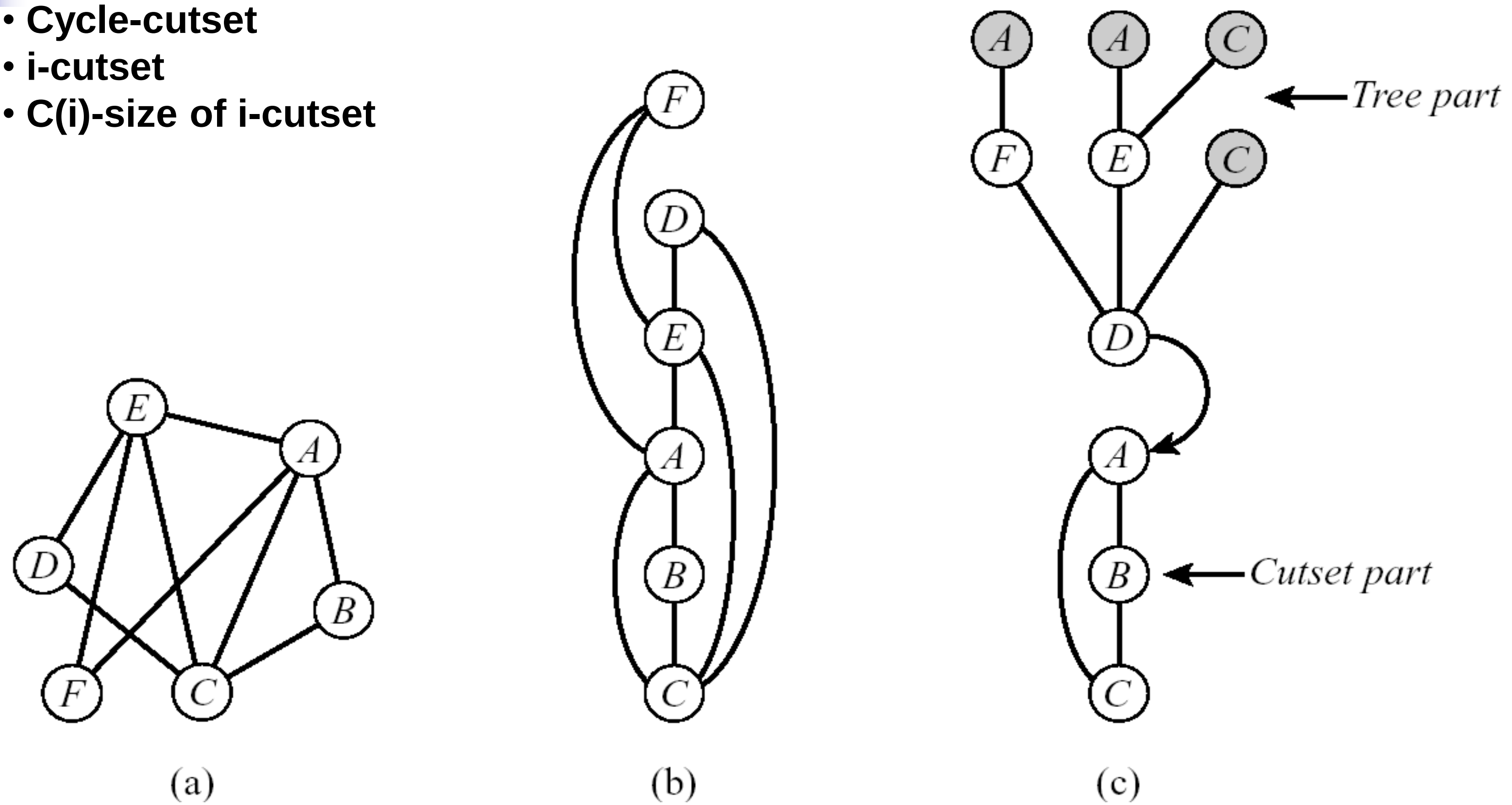
General principle:
Condition until tractable
Then solve sub-problems efficiently

Search Basic Step:
Variable Branching by Conditioning



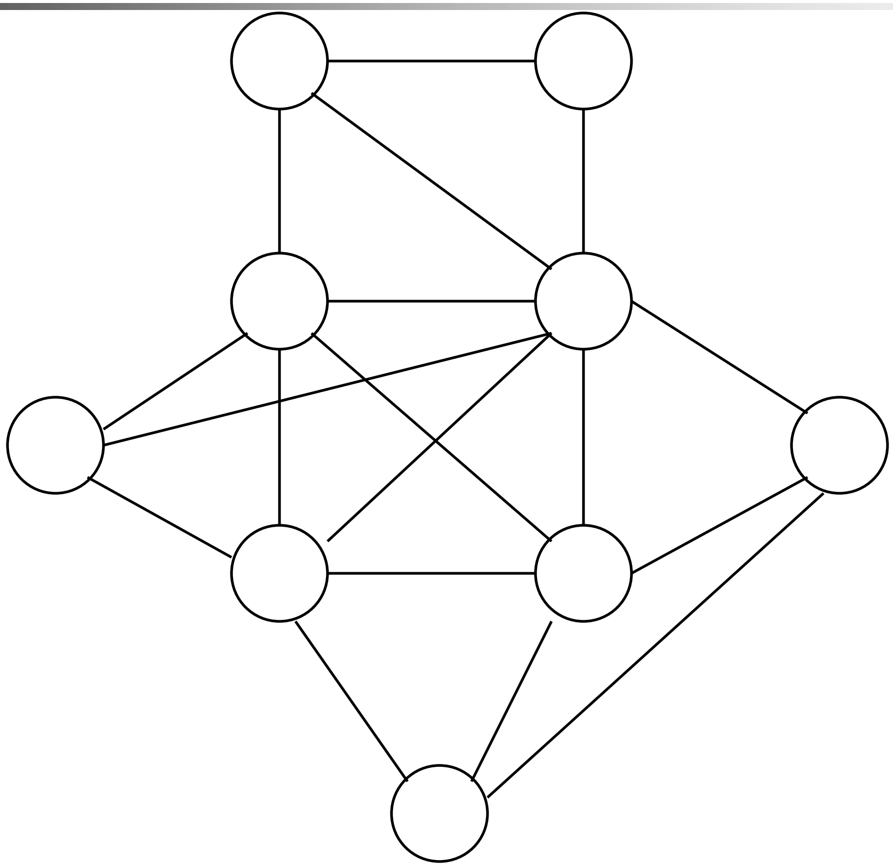
The Cycle-Cutset Scheme:
Condition Until Treeness

- **Cycle-cutset**
- **i-cutset**
- **C(i)-size of i-cutset**

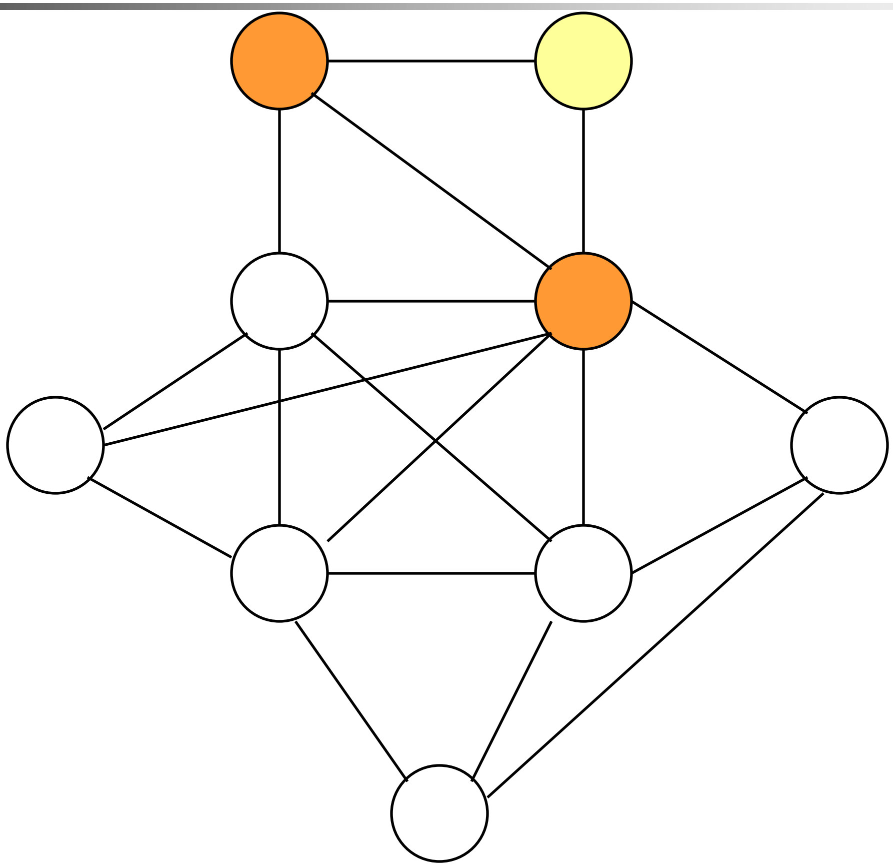


Space: exp(i), Time: O(exp(i+c(i))

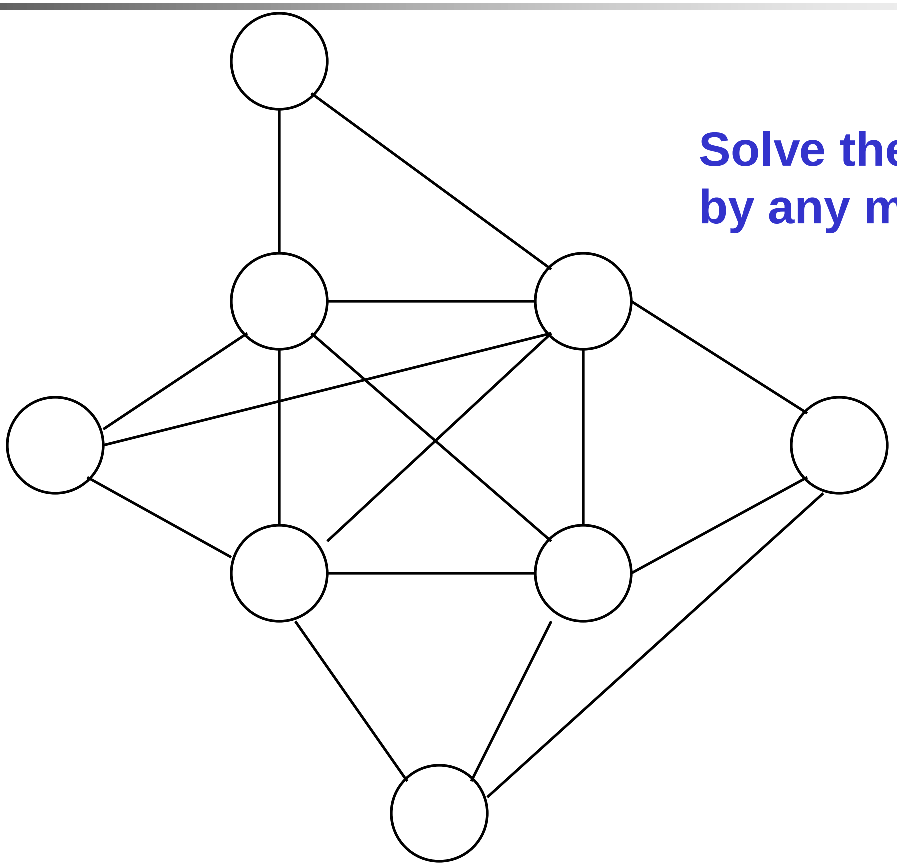
Eliminate First



Eliminate First



Eliminate First

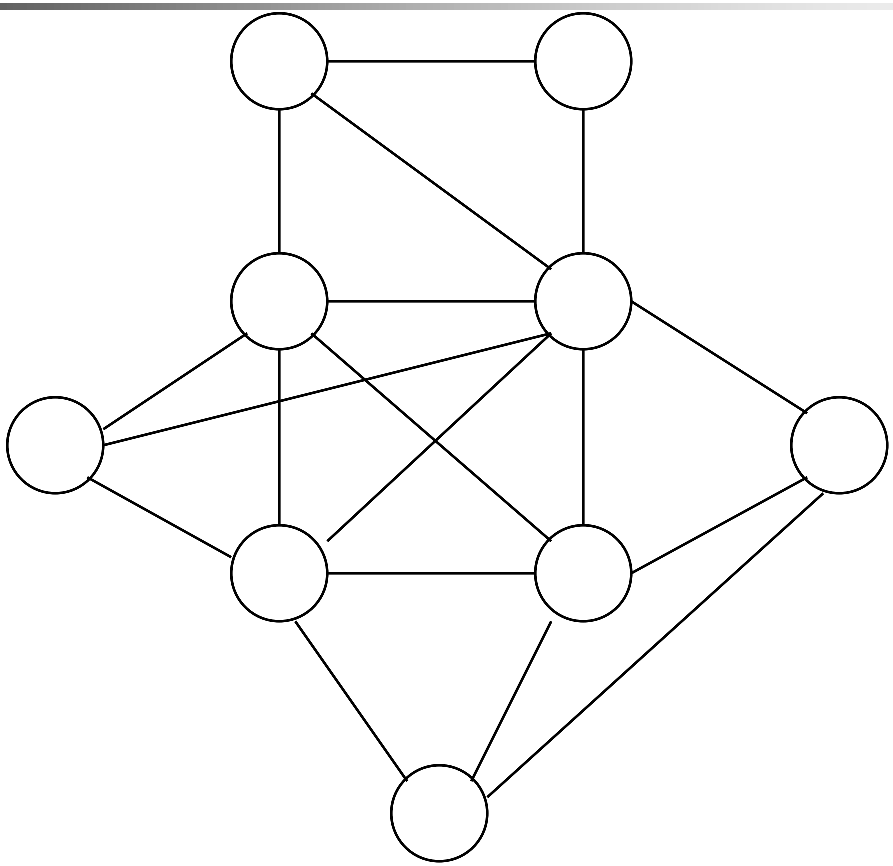


Solve the rest of the problem by any means

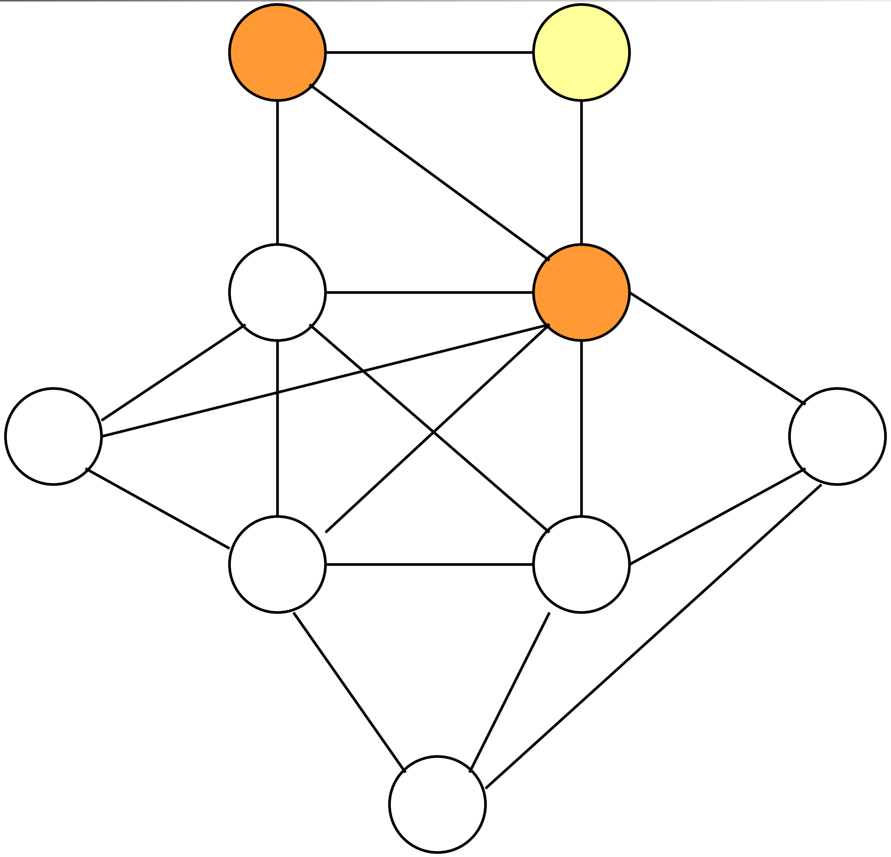
Hybrids Variants

- Condition, condition, condition … and then only eliminate (w-cutset, cycle-cutset)
- Eliminate, eliminate, eliminate … and then only search
- Interleave conditioning and elimination (elim-cond(i), VE+C)

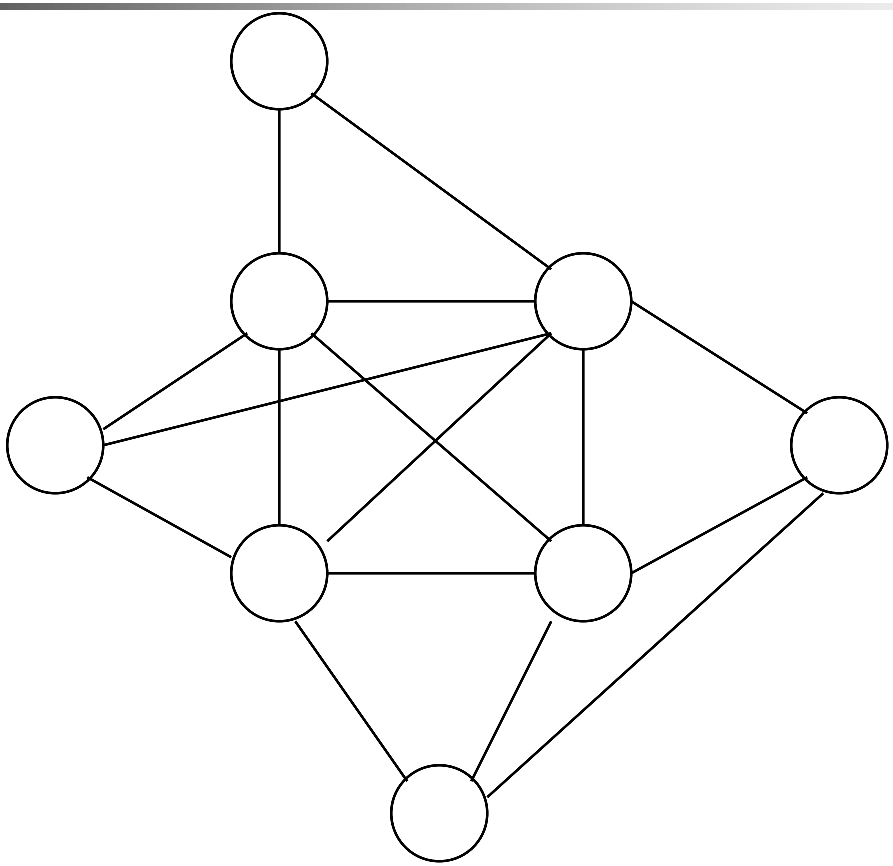
Interleaving Conditioning and Elimination
(Larrosa & Dechter, CP'02)



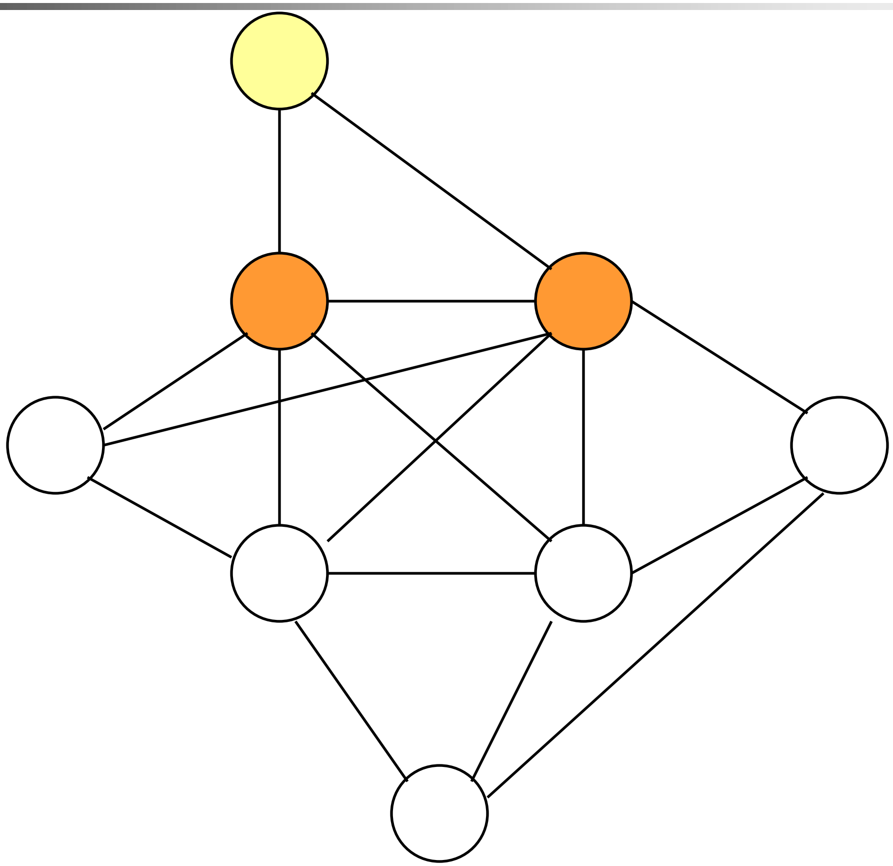
Interleaving Conditioning and Elimination



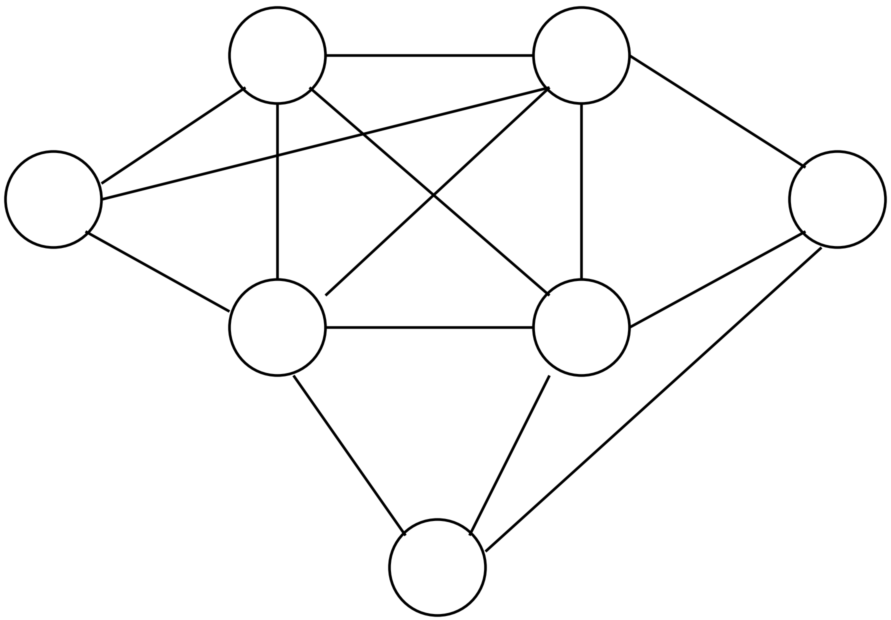
Interleaving Conditioning and Elimination



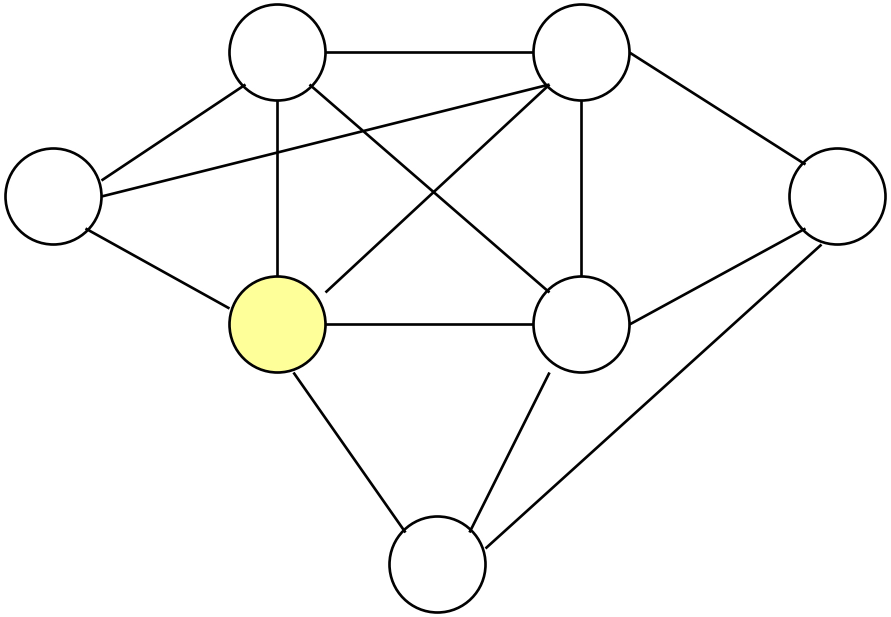
Interleaving Conditioning and Elimination



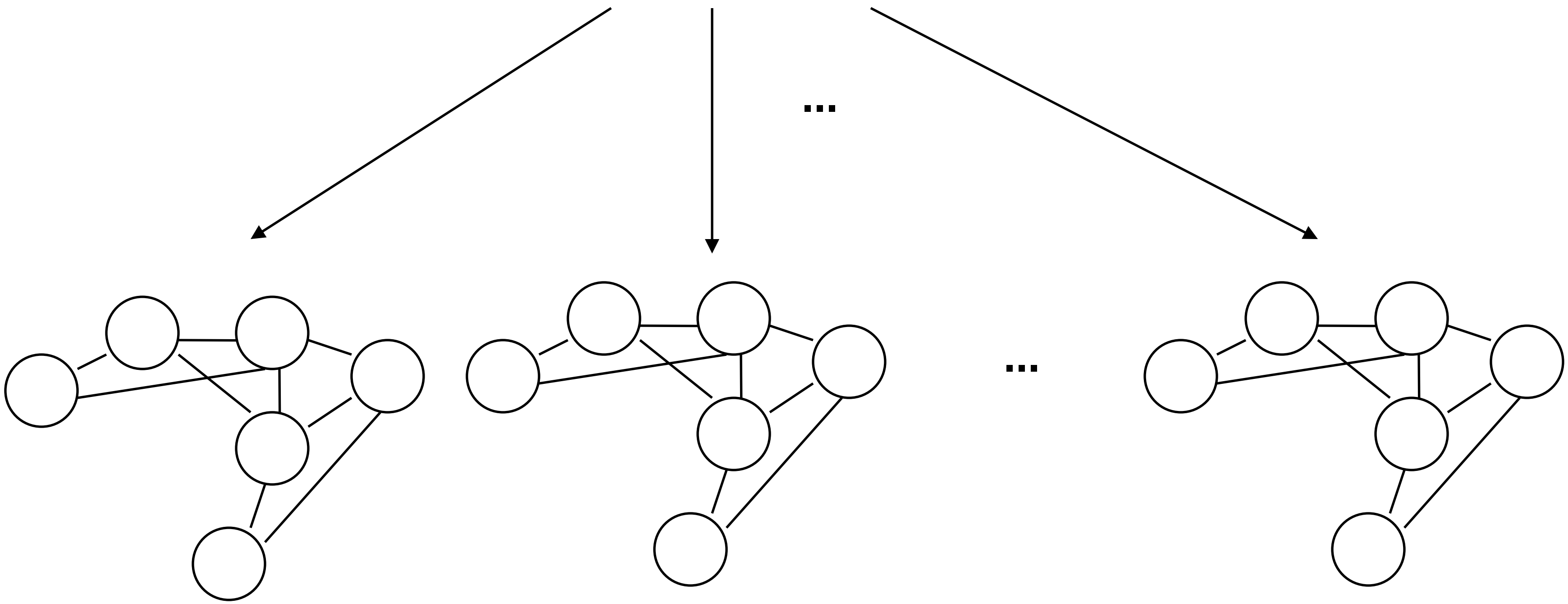
Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Algorithm VEC (Variable-elimination with conditioning)

Algorithm VEC

Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, d; a subset C of conditioned variables; observations e.

Output: $Bel(A)$.

Initialize: $\lambda = 0$.

1. For every assignment $C = c$, do
   - $\lambda_1 \leftarrow$ The output of BE-bel with $c \cup e$ as observations.
   - $\lambda \leftarrow \lambda + \lambda_1$. (update the sum).
2. **Return** λ.

Complexity of w-cutset (VEC)

Theorem 6.5.1 *Given a set of conditioning variables, C, the space complexity of algorithm elim-cond-bel is $O(n \cdot exp(w^*(d, c \cup e))$, while its time complexity is $O(n \cdot exp(w^*(d, e \cup c) + |C|))$, where the induced width $w^*(d, c \cup e)$, is computed on the ordered moral graph that was adjusted relative to e and c.* □

Definition 6.5.3 (secondary-optimization task) *Given a graph $G = (V, E)$ and a constant r, find a smallest subset of nodes C_r, such that $G' = (V - C_r, E')$, where E' includes all the edgs in E that are not incident to nodes in C_r, has induced-width less or equal r.*

What hybrid should we use?

- w=1? (loop-cutset?)
- w=0? (Full search?)
- w=w* (Full inference)?
- w in between?
- depends… on the graph
- What is relation between cycle-cutset and the induced-width?