

# Exact Inference Algorithms for Probabilistic Reasoning; BTE and CTE



---

COMPSCI 276, Spring 2013  
Set 6: Rina Dechter

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



# Probabilistic Inference Tasks

---

- Belief updating:

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

- Finding most probable explanation (MPE)

$$\bar{\mathbf{x}}^* = \mathbf{argmax}_{\bar{\mathbf{x}}} \mathbf{P}(\bar{\mathbf{x}}, \mathbf{e})$$

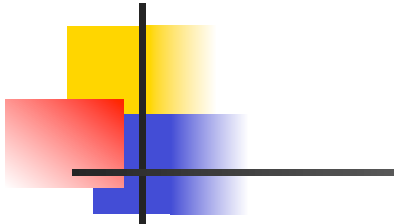
- Finding maximum a-posteriori hypothesis

$$(\mathbf{a}_1^*, \dots, \mathbf{a}_k^*) = \mathbf{argmax}_{\bar{\mathbf{a}}} \sum_{\bar{\mathbf{x}} \perp \bar{\mathbf{a}}} \mathbf{P}(\bar{\mathbf{x}}, \mathbf{e}) \quad \begin{array}{l} A \subseteq X: \\ \text{hypothesis variables} \end{array}$$

- Finding maximum-expected-utility (MEU) decision

$$(\mathbf{d}_1^*, \dots, \mathbf{d}_k^*) = \mathbf{argmax}_{\bar{\mathbf{d}}} \sum_{\bar{\mathbf{x}} \perp \bar{\mathbf{d}}} \mathbf{P}(\bar{\mathbf{x}}, \mathbf{e}) \mathbf{U}(\bar{\mathbf{x}}) \quad \begin{array}{l} D \subseteq X: \text{ decision variables} \\ U(\bar{\mathbf{x}}): \text{ utility function} \end{array}$$

1998 roadmap



## Outline; Road Map

Tasks Methods	CSP	SAT	Optimization	Belief updating	MPE, MAP, MEU	Solving linear equalities/inequalities
elimination	adaptive consistency join-tree	directional resolution	dynamic programming	join-tree, VE, SPI, elim-bel	join-tree, elim-mpe, elim-map	Gaussian/ Fourier elimination
conditioning	backtracking search	backtracking (Davis-Putnam)	branch-and-bound, best-first search		branch-and-bound, best-first search	
elimination + conditioning	cycle-cutset forward checking	DCCR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedy local search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

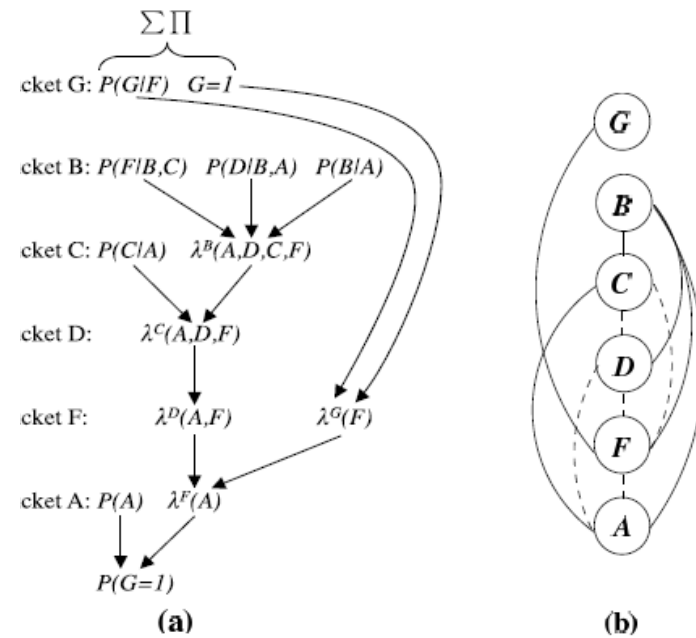
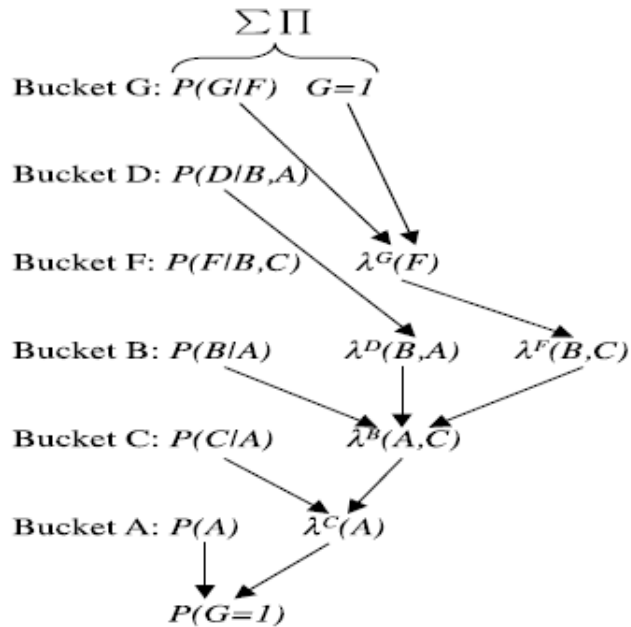
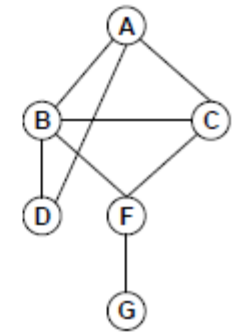
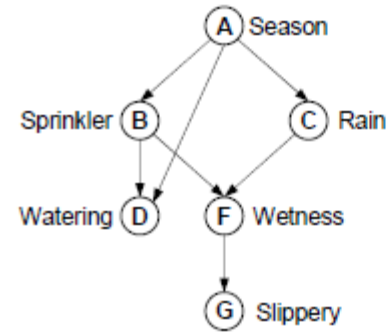


# Agenda

---

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, the join-tree algorithm
- Belief-propagation on acyclic probabilistic networks (poly-trees)
- The role of induced-width/tree-width
- Conditioning with elimination

# A Bayesian Network Processed by BE

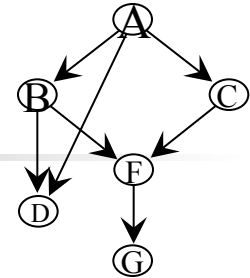


The bucket's output when processing along  $d_2 = A, F, D, C, B, G$

Figure 4.2: Bucket elimination along ordering  $d_1 = A, C, B, F, D, G$ .

Complexity exponential in  $w^*_d$

# From Bucket Elimination to Bucket-Tree Elimination



What if we want the marginal on B?

Bucket G:  $P(G|F)$

Bucket F:  $P(F|B, C)$   $\lambda_{G \rightarrow F}(F)$

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

Bucket C:  $P(C|A)$   $\lambda_{F \rightarrow C}(B, C)$

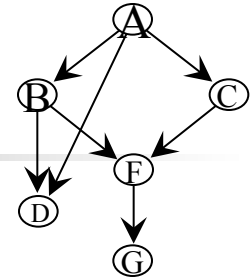
Bucket B:  $P(B|A)$   $\lambda_{D \rightarrow B}(A, B)$   $\lambda_{C \rightarrow B}(A, B)$

Bucket A:  $P(A)$   $\lambda_{B \rightarrow A}(A)$



Observation 1: BE is a message propagation down a bucket-tree

# From Bucket Elimination to Bucket-Tree Elimination



**What If we want the marginal on B?**

Bucket G:  $P(G|F)$

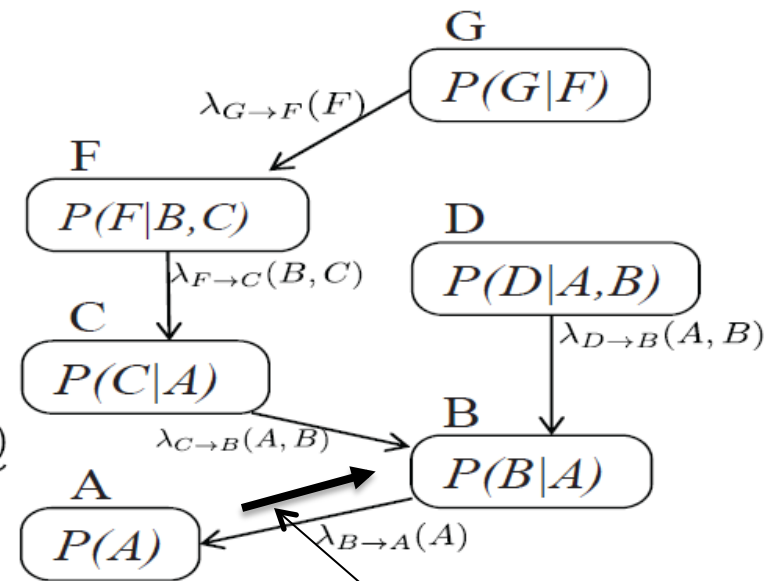
Bucket F:  $P(F|B,C)$

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

Bucket C:  $P(C|A)$

Bucket B:  $P(B|A)$

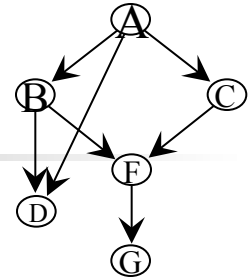
Bucket A:  $P(A)$



$$\pi_{A \rightarrow B}(a) = P(A),$$

$$P(B) = \sum P(B | A)P(A)\lambda_C^B(B, C)\lambda_D^B(A, B)$$

# From Bucket Elimination to Bucket-Tree Elimination



**If we want the marginal on D?  
Imagine combining B and A, D  
 $d = (\{A,D,B\}, C, F, G)$**

Bucket G:  $P(G|F)$

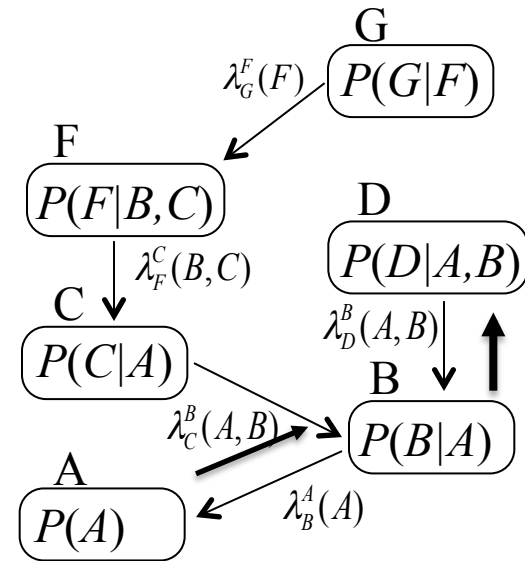
Bucket F:  $P(F|B,C) \rightarrow \lambda_G^F(F)$

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

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

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

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



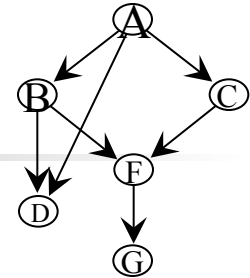
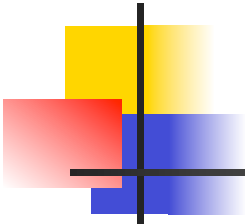
$$\pi_{A \rightarrow B}(a) = P(A),$$

$$\pi_{B \rightarrow D}(a, b) = p(b|a) \cdot \pi_{A \rightarrow B}(a) \cdot \lambda_{C \rightarrow B}(b)$$

$$bel(d) = \alpha \sum_{a,b} P(d|a, b) \cdot \pi_{B \rightarrow D}(a, b).$$



# From Bucket Elimination to Bucket-Tree Elimination



**If we want the marginal on D?**  
**Imagine combining B and A, D**  
 **$d = (\{A, D, B\}, C, F, G)$**

Bucket G:  $P(G|F)$

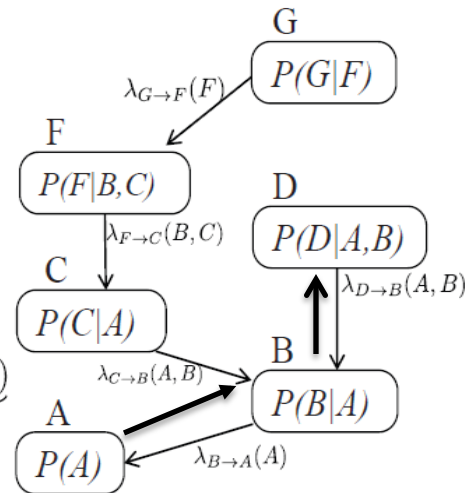
Bucket F:  $P(F|B, C)$

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

Bucket C:  $P(C|A)$

Bucket B:  $P(B|A)$

Bucket A:  $P(A)$



$$\pi_{A \rightarrow B}(a) = P(A),$$

$$\pi_{B \rightarrow D}(a, b) = p(b|a) \cdot \pi_{A \rightarrow B}(a) \cdot \lambda_{C \rightarrow B}(b)$$

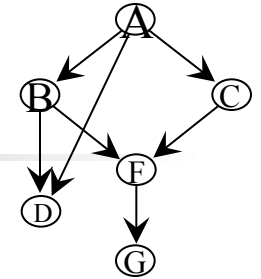
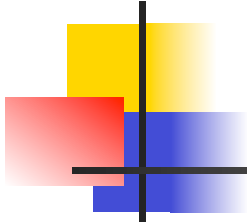
$$bel(d) = \alpha \sum_{a, b} P(d|a, b) \cdot \pi_{B \rightarrow D}(a, b).$$

# Idea of BTE

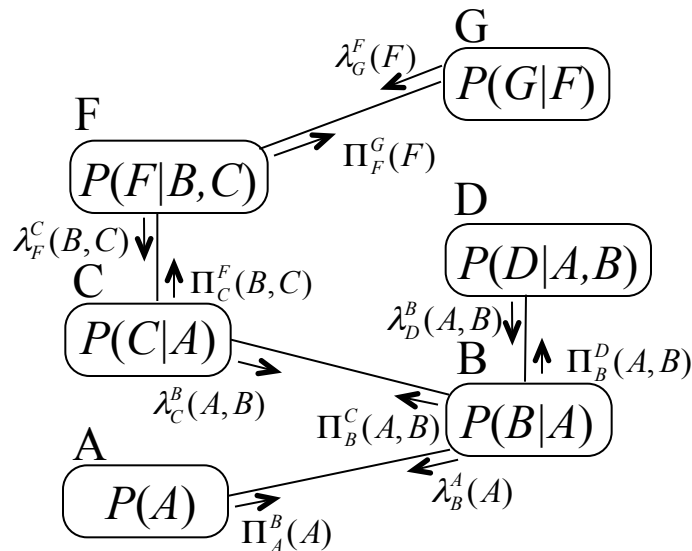
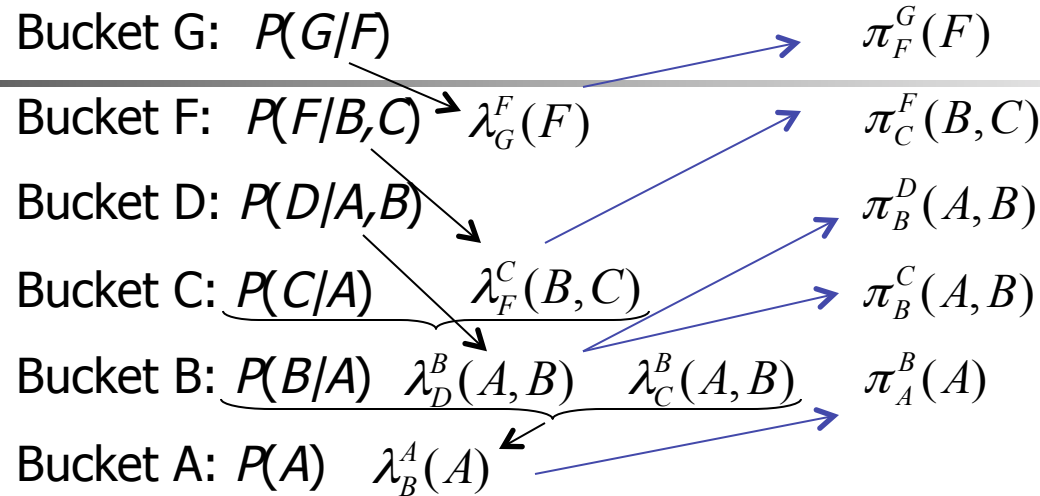
This example can be generalized. We can compute the belief for every variable by a second message passing from the root to the leaves along the original bucket-tree, such that at termination the belief for each variable can be computed locally consulting only the functions in its own bucket. In the following we will describe the idea of message

in Bayesian networks. Given an ordering of the variables  $d$  the first step generates the bucket-tree by partitioning the functions into buckets and connecting the buckets into a tree. The subsequent *top-down* phase is identical to general bucket-elimination. The *bottom-up* messages are defined as follows. The messages sent from the root up to the leaves will be denoted by  $\pi$ . The message from  $B_j$  to a child  $B_i$  is generated by combining (e.g., multiplying) all the functions currently in  $B_j$  including the  $\pi$  messages from its parent bucket and all the  $\lambda$  messages from its *other* child buckets and marginalizing (e.g., summing) over the eliminator from  $B_j$  to  $B_i$ . By construction, downward messages are generated by eliminating a single variable. Upward messages, on the other hand, may be generated by eliminating zero, one or more variables.

# BTE: Allows Messages Both Ways



Each bucket can  
Compute its  
marginal probability



$$\pi_A^B(a) = P(a)$$

$$\pi_B^C(c, a) = P(b|a)\lambda_D^B(a, b)\pi_A^B(a)$$

$$\pi_B^D(a, b) = P(b|a)\lambda_C^B(a, b)\pi_A^B(a, b)$$

$$\pi_C^F(c, b) = \sum_a P(c|a)\pi_B^C(a, b)$$

$$\pi_F^G(f) = \sum_{b,c} P(f|b, c)\pi_C^F(c, b)$$



# A Bucket Tree of a Bayesian Network

---

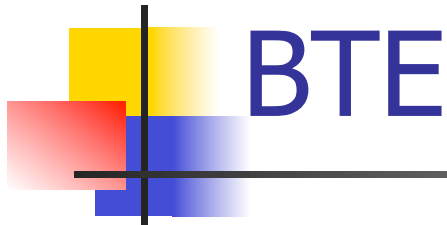
- The bucket-tree is the bucket-structure connected into a tree:
  - Nodes are the buckets. Each has functions (assigned initially) and variables: itself+ induced-parents
  - There is an arc from  $B \downarrow i$  to  $B \downarrow j$  iff the function created at bucket  $B \downarrow i$  is placed at bucket  $B \downarrow j$
  - We have a separator and eliminator between two adjacent buckets



## Bucket-tree Generation from the Graph

---

1. Pick a (good) variable ordering,  $d$ .
2. Generate the induced ordered graph
3. From top to bottom, each bucket of  $X$  is mapped to (variables, functions) pairs
4. The variables are the clique of  $X$ , the functions are those placed in the bucket
5. Connect Bucket of  $X$  to earlier bucket of  $Y$  if  $Y$  is closest node connected to  $X$



**Theorem:** When BTE terminates The product of functions in each bucket is the beliefs of the variables joint with the evidence.

ALGORITHM BUCKET-TREE ELIMINATION (BTE)

**Input:** A problem  $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \Pi \rangle$ , ordering  $d$ .  $X = \{X_1, \dots, X_n\}$  and  $F = \{f_1, \dots, f_r\}$   
Evidence  $E = e$ .

**Output:** Augmented buckets  $\{B'_i\}$ , containing the original functions and all the  $\pi$  and  $\lambda$  functions received from neighbors in the bucket-tree.

1. **Pre-processing:** Partition functions to the ordered buckets as usual and generate the bucket-tree.

2. **Top-down phase:**  $\lambda$  messages (BE) do

for  $i = n$  to 1, in reverse order of  $d$  process bucket  $B_i$ :

The message  $\lambda_{i \rightarrow j}$  from  $B_i$  to its parent  $B_j$ , is:

$$\lambda_{i \rightarrow j} \leftarrow \sum_{elim(i,j)} \psi_i \cdot \prod_{k \in child(i)} \lambda_{k \rightarrow i}$$

endfor

3. **bottom-up phase:**  $\pi$  messages

for  $j = 1$  to  $n$ , process bucket  $B_j$  do:

$B_j$  takes  $\pi_{k \rightarrow j}$  received from its parent  $B_k$ , and computes a message  $\pi_{j \rightarrow i}$  for each child bucket  $B_i$  by

$$\pi_{j \rightarrow i} \leftarrow \sum_{elim(j,i)} \pi_{k \rightarrow j} \cdot \psi_j \cdot \prod_{r \neq i} \lambda_{r \rightarrow j}$$

endfor

4. **Output:** and answering singleton queries (e.g., deriving beliefs).

Output augmented buckets  $B'_1, \dots, B'_n$ , where each  $B'_i$  contains the original bucket functions and the  $\lambda$  and  $\pi$  messages it received.



# Query answering

---

## COMPUTING MARGINAL BELIEFS

**Input:** a bucket-tree processed by BTE with augmented buckets:  $B'_1, \dots, B'_n$

**output:** beliefs of each variable, bucket and probability of evidence.

$$bel(B_i) \Leftarrow \prod_{f \in B'_i} f$$

$$bel(x_i) \Leftarrow \sum_{B_i - \{X_i\}} \prod_{f \in B'_i} f$$

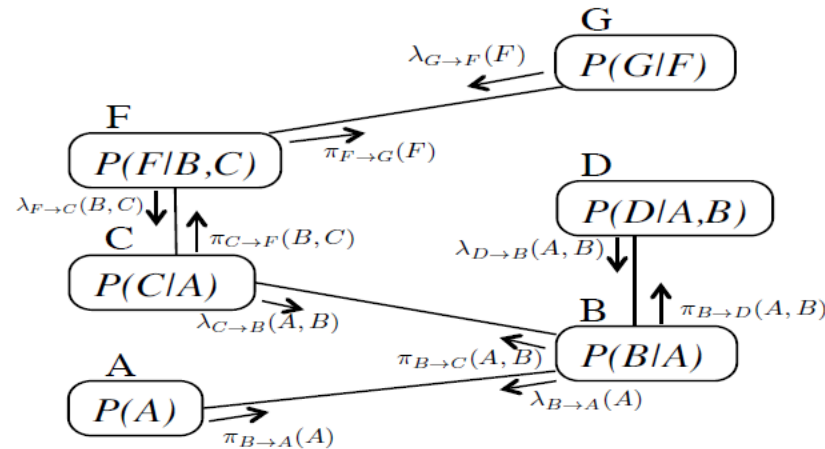
$$P(\text{evidence}) \Leftarrow \sum_{B_i} \prod_{f \in B'_i} f$$

Figure 6.4: Query answering

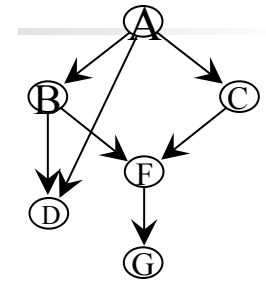
# BTE: Allows messages both ways



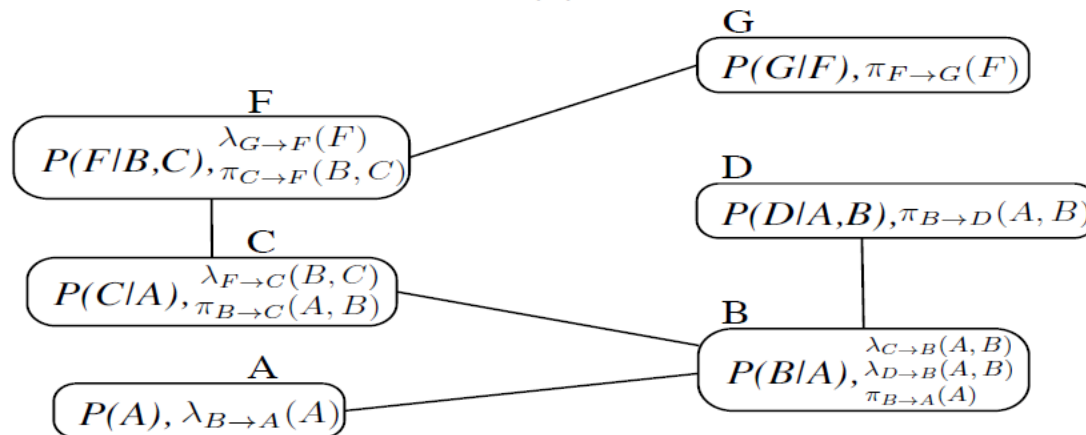
Initial buckets  
+ messages



(a)



Output buckets



(b)





# Explicit functions

---

**Definition 6.1.4 (explicit function and explicit sub-model)** *Given a graphical model  $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$ , and reasoning tasks defined by marginalization  $\sum$  and given a subset of variables  $Y$ ,  $Y \subseteq \mathbf{X}$ , we define  $\mathcal{M}_Y$ , the explicit function of  $\mathcal{M}$  over  $Y$ :*

$$\mathcal{M}_Y = \sum \prod f, \quad (6.4)$$

*We denote by  $F_Y$  any set of functions whose scopes are subsumed in  $Y$  over the same domains and ranges as the functions in  $\mathbf{F}$ . we say that  $(Y, F_Y)$  is an explicit submodel of  $M$  iff*

$$\prod_{f \in F_Y} f = \mathcal{M}_Y \quad (6.5)$$



# 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 immapness 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  $O(r \text{ deg } k^{\{w^*+1\}})$ , where  $\text{deg}$  is the maximum degree in the bucket-tree. The space complexity of BTE is  $O(n k^{w^*})$ .*

# Asynchronous BTE: Bucket-tree Propagation (BTP)

## BUCKET-TREE PROPAGATION (BTP)

**Input:** A problem  $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \Pi \rangle$ , ordering  $d$ .  $X = \{X_1, \dots, X_n\}$  and  $F = \{f_1, \dots, f_r\}$ ,  $E = e$ . An ordering  $d$  and a corresponding bucket-tree structure, in which for each node  $X_i$ , its bucket  $B_i$  and its neighboring buckets are well defined.

**Output:** Explicit buckets.

1. **for** bucket  $B_i$  **do**:
2.     **for** each neighbor bucket  $B_j$  **do**,  
        once all messages from all other neighbors were received, **do**  
            compute and send to  $B_j$  the message  
             $\lambda_{i \rightarrow j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \rightarrow i})$
3. **endfor**

$$\lambda_{i \rightarrow j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \rightarrow i})$$



# Complexity of BTE

---

**Theorem 6.1.6 (Complexity of BTE)** *Let  $w^*$  be the induced width of  $G$  along ordering  $d$ , of the primal graph of a graphical model  $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \Pi, \Sigma \rangle$ ,  $r$  be the number of functions in  $\mathbf{F}$  and  $k$  be the maximum domain size. The time complexity of BTE is  $O(r \cdot \text{deg} \cdot k^{w^*+1})$ , where  $\text{deg}$  is the maximum degree in the bucket-tree. The space complexity of BTE is  $O(n \cdot k^{w^*})$ .*

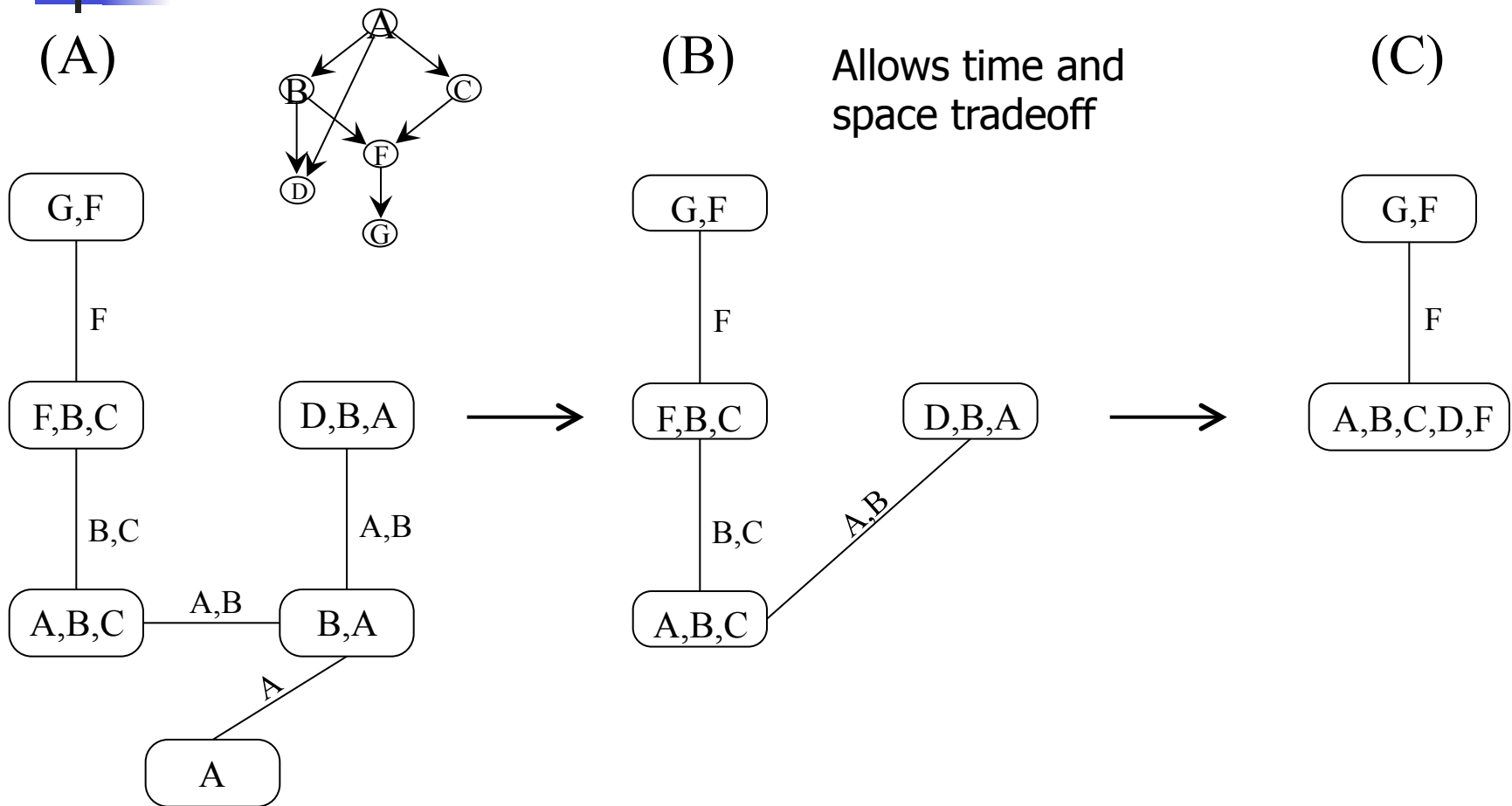
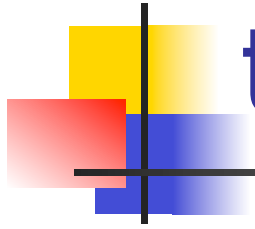


# Agenda

---

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, the join-tree algorithm
- Belief-propagation on acyclic probabilistic networks (poly-trees)
- The role of induced-width/tree-width

# From buckets to superbucket to clusters



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



# 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 variables.
- Separators are the intersection of variables on the arcs of the tree.
- The cluster-tree is an i-map.



# Tree-decompositions

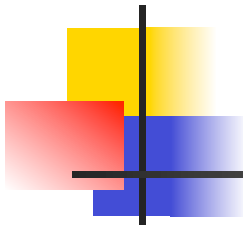
---

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

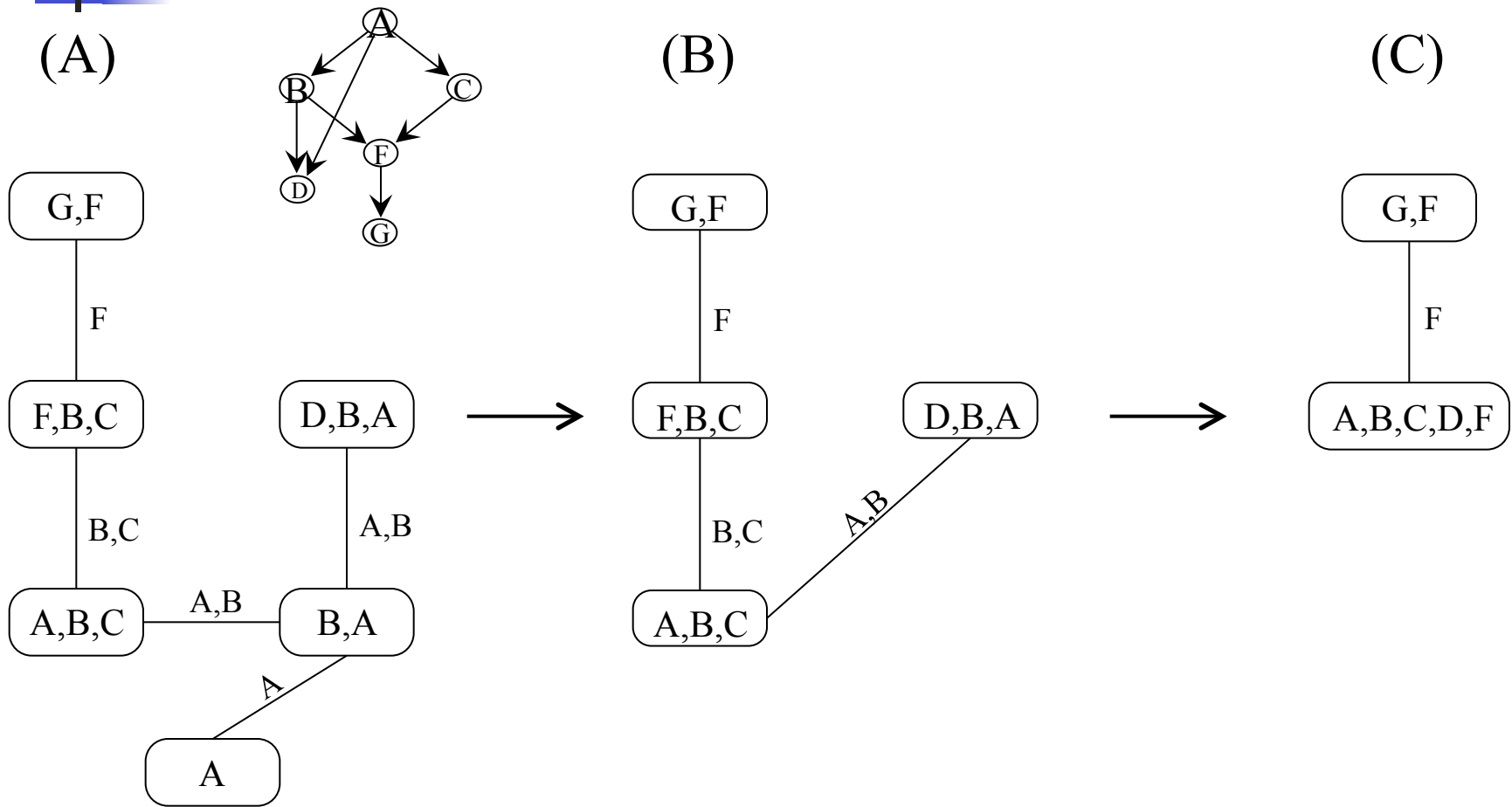
1. For each function  $p_i \in P$  there is exactly one vertex such that  $p_i \in \psi(v)$  and  $scope(p_i) \subseteq \chi(v)$
2. For each variable  $X_i \in X$  the set  $\{v \in V | X_i \in \chi(v)\}$  forms a connected subtree (running intersection property)

**Definition 6.2.8 (treewidth, separator-width, eliminator)** *The treewidth [4] of a tree-decomposition  $\langle T, \chi, \psi \rangle$  is  $\max_{v \in V} |\chi(v)|$  minus 1. Given two adjacent vertices  $u$  and  $v$  of a tree-decomposition, the separator of  $u$  and  $v$  is  $sep(u, v) = \chi(u) \cap \chi(v)$ , and the eliminator of  $u$  with respect to  $v$  is  $elim(u, v) = \chi(u) - \chi(v)$ . The separator-width is the maximum over all separators.*

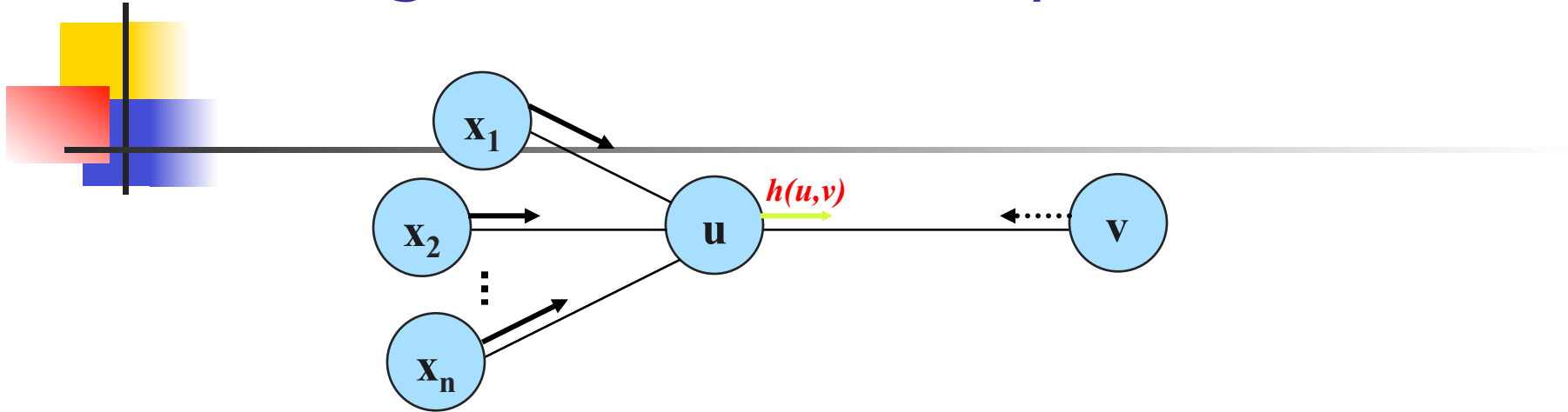




# Examples



# The general Message Passing on a general tree-decomposition



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

For max-product  
Just replace  $\sum$  with  
With max.

Compute the message :

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

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



# Cluster-Tree Elimination

## CLUSTER-TREE ELIMINATION (CTE)

**Input:** A tree decomposition  $\langle T, \chi, \psi \rangle$  for a problem  $M = \langle X, D, F, \Pi \rangle$ ,  
 $X = \{X_1, \dots, X_n\}$ ,  $F = \{f_1, \dots, f_r\}$ . Evidence  $E = e$ ,  $\psi_u = \prod_{f \in \psi(u)} f$

**Output:** An augmented tree-decomposition whose clusters are all model explicit.

Namely, a tree decomposition  $\langle T, \chi, \bar{\psi} \rangle$  where for  $u \in T$ ,  $\chi(u), \bar{\psi}(u)$  is model explicit.

1. **Initialize:** Let  $m_{u \rightarrow v}$  denote the message sent from vertex  $u$  to vertex  $v$ .
2. **Compute messages:**

**for** every edge  $(u, v)$  in the tree, **do**

        If vertex  $u$  has received messages from all adjacent vertices other than  $v$ , then compute (and send to  $v$ )

$$m_{u \rightarrow v} \leftarrow \sum_{sep(u,v)} \psi_u \cdot \prod_{r \in neighbor(v)} m_{r \rightarrow u}$$

**endfor**

Note: functions whose scopes do not contain any separator variable

do not need to be combined and can be directly passed on to the receiving vertex.

3. **Return:** The explicit tree-decomposition  $\langle T, \chi, \bar{\psi} \rangle$ , where

$$\bar{\psi}(v) \leftarrow \psi(v) \cup_{u \in neighbor(v)} \{m_{u \rightarrow v}\}.$$



# Generating tree-decomposition

---

**Proposition 6.2.12** *If  $T$  is a tree-decomposition, then any tree obtained by merging adjacent clusters is also a tree-decomposition.*

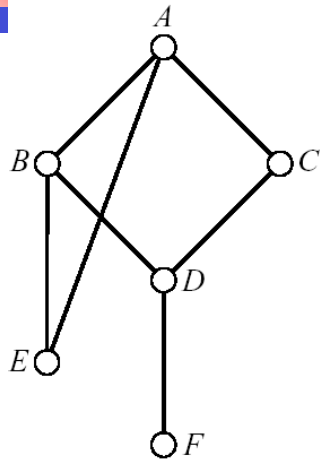
## GRAPH TRIANGULATION (FILL-IN) ALGORITHM: Tarjan and Yannakakis [1984]

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.

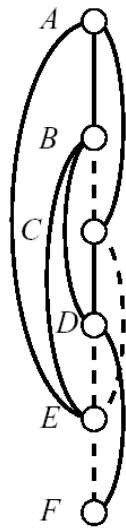
### ASSEMBLING A JOIN TREE

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  $C_1, C_2, \dots, C_t$  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  $C_i$ .

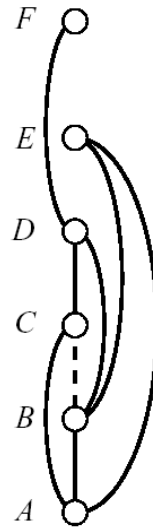
# Examples of tree-clustering



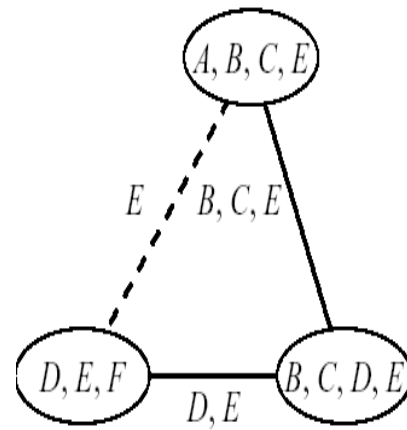
(a)



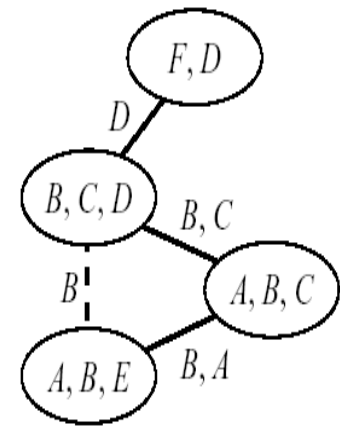
(b)



(c)

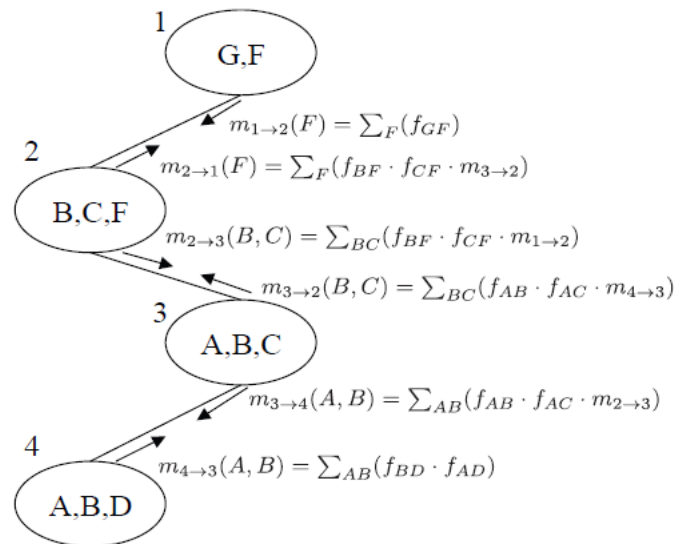
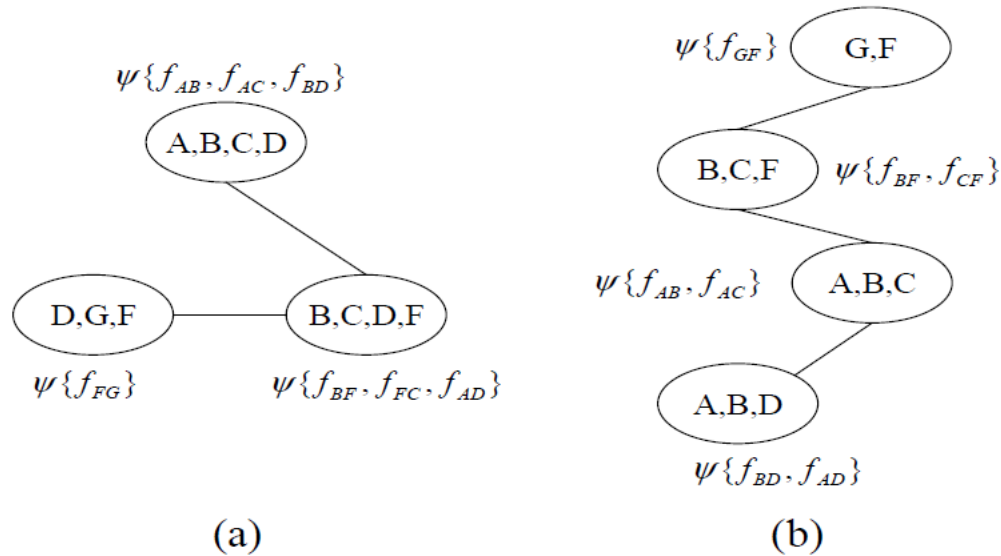
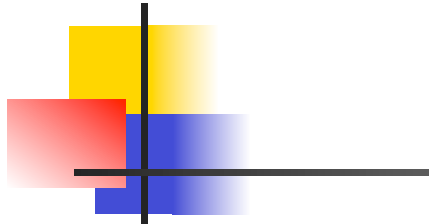


(a)



(b)

# Tree-clustering and message-passing



# Cluster-Tree Elimination - Properties

- Correctness and completeness: Algorithm CTE is correct, i.e. it computes the exact joint probability of a single variable and the evidence.
- Time complexity:
  - $O(deg \times (n+N) \times d^{w^*+1})$
- Space complexity:  $O(N \times d^{sep})$   
where
  - $deg$  = the maximum degree of a node
  - $n$  = number of variables (= number of CPTs)
  - $N$  = number of nodes in the tree decomposition
  - $d$  = the maximum domain size of a variable
  - $w^*$  = the induced width
  - $sep$  = the separator size



# Example

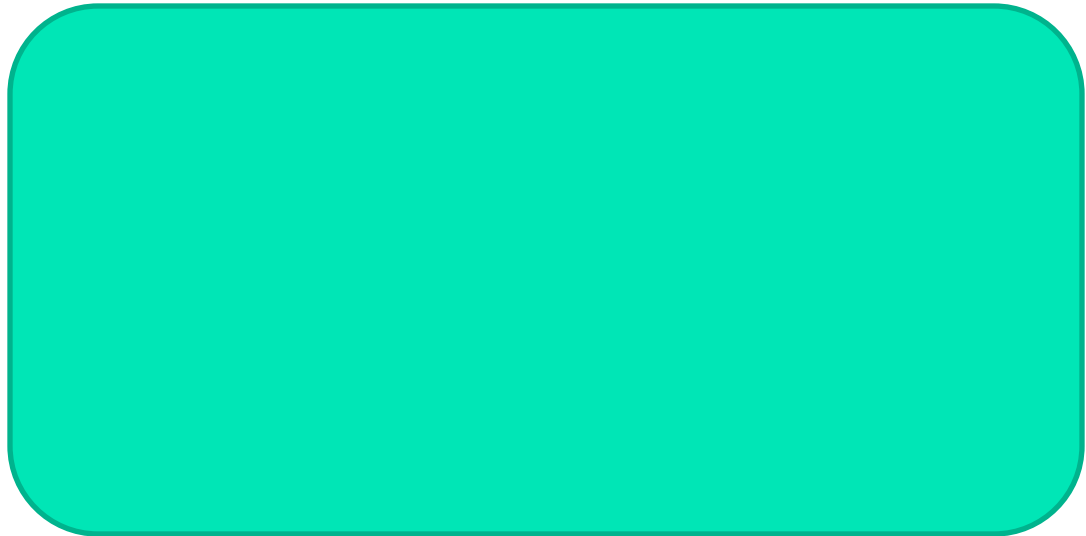
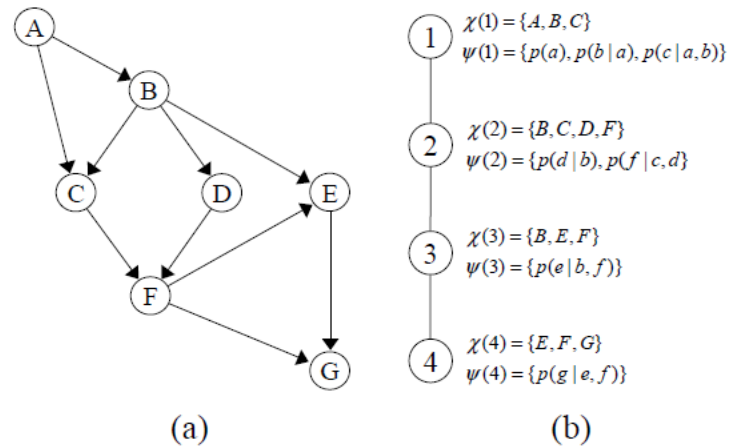


Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

# Example

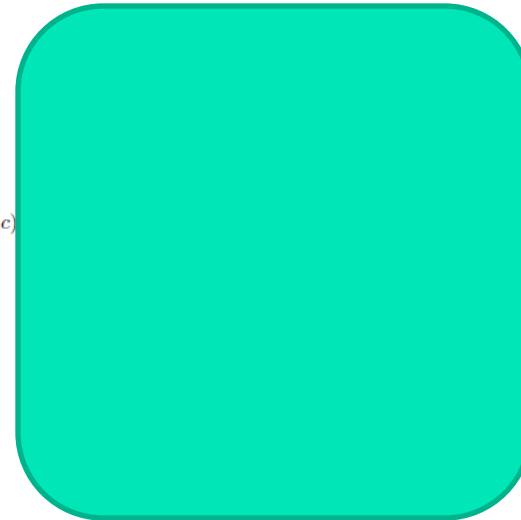
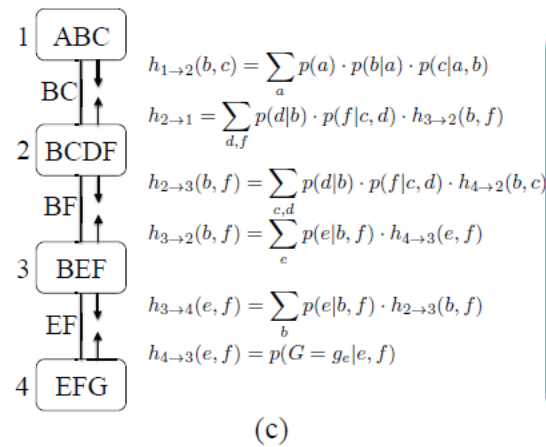
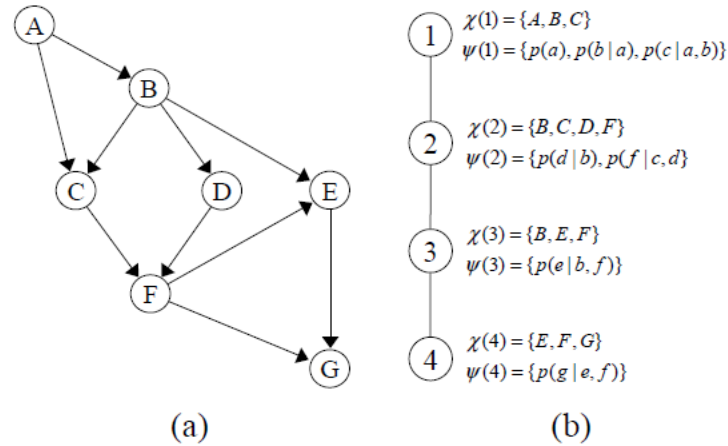


Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

# Example

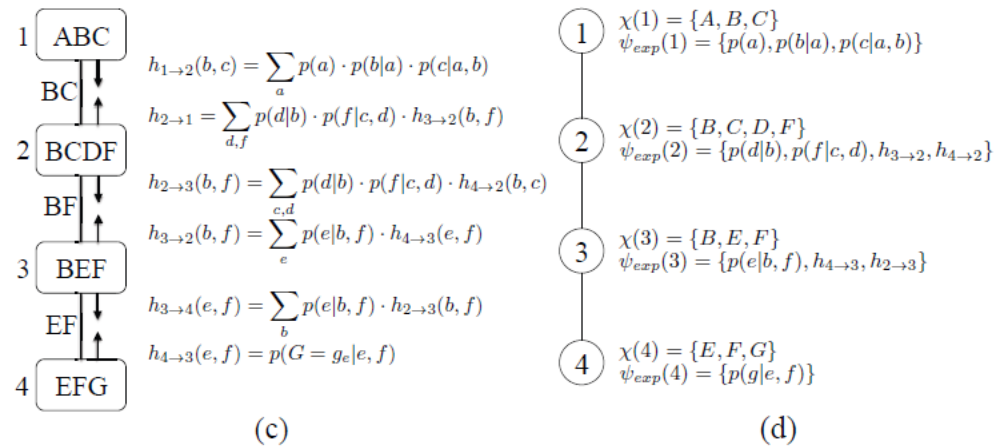
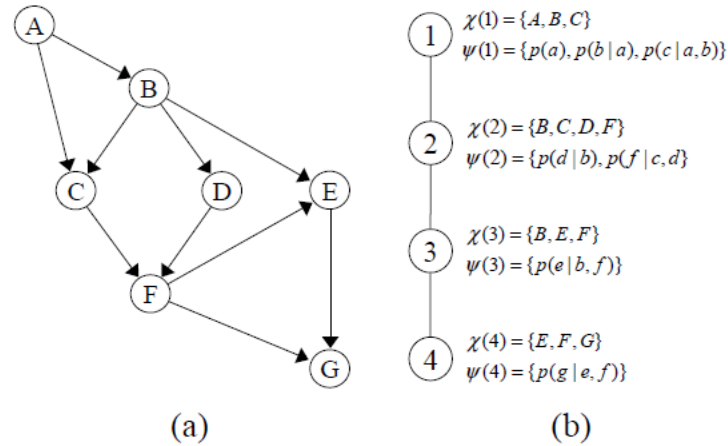
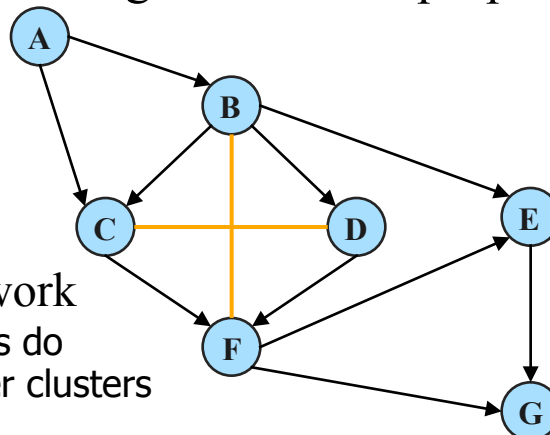


Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

# Example

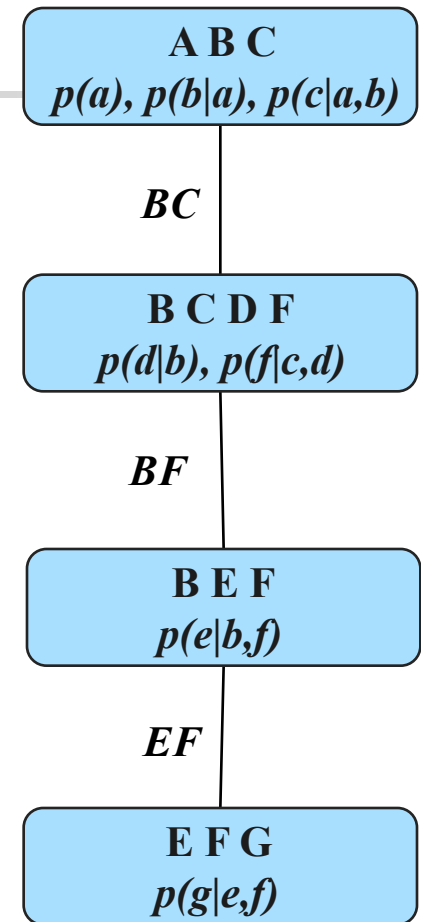
A *tree decomposition* for a belief network  $BN = \langle X, D, G, P \rangle$  is a triple  $\langle T, \chi, \psi \rangle$ , where  $T = (V, E)$  is a tree and  $\chi$  and  $\psi$  are labeling functions, associating with each vertex  $v \in V$  two sets,  $\chi(v) \subseteq X$  and  $\psi(v) \subseteq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



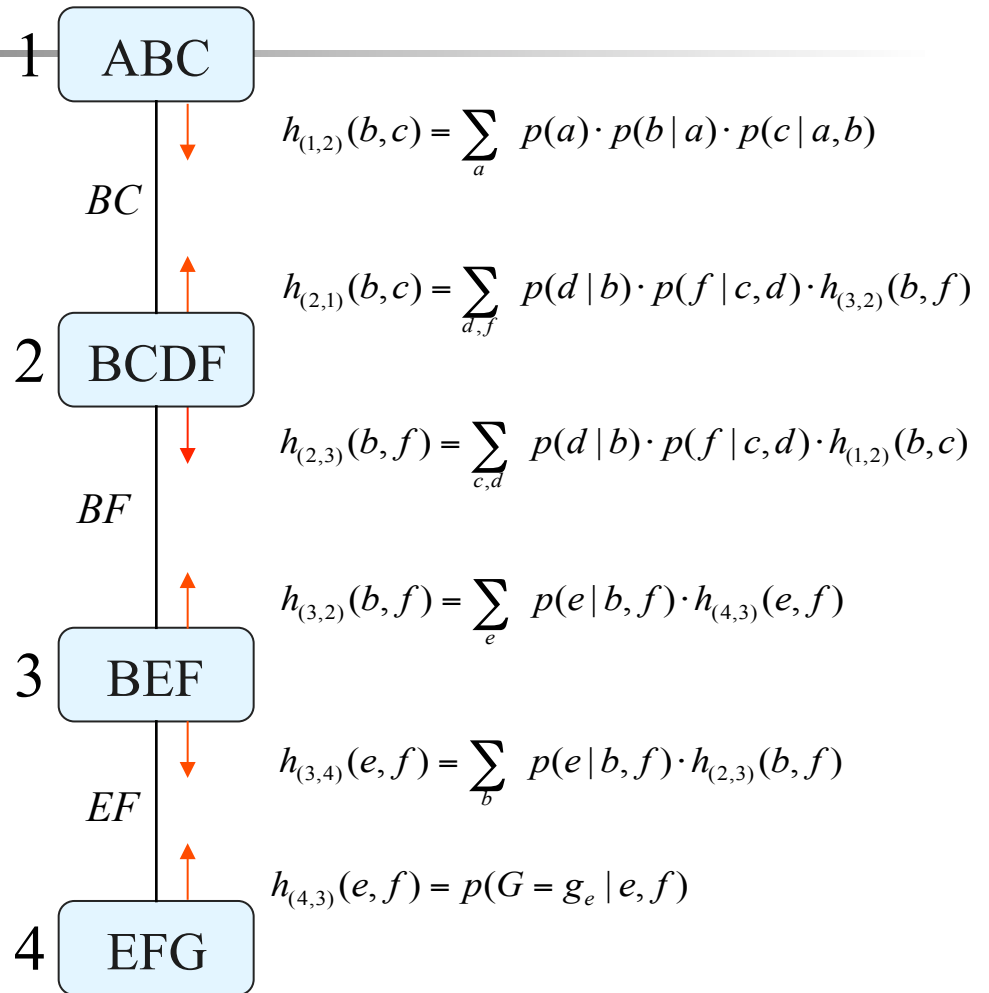
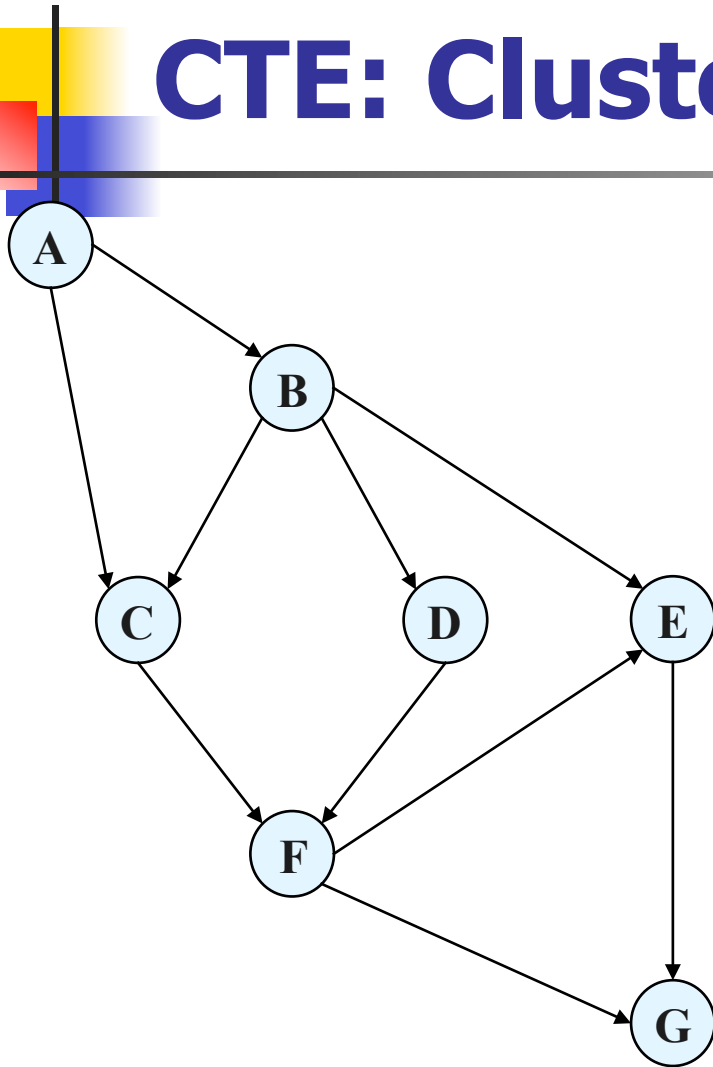
Belief network

Minimal tree-decompositions do  
Not contain other clusters



Tree decomposition

# CTE: Cluster Tree Elimination



**Time:**  $O(\exp(w+1))$

**Space:**  $O(\exp(sep))$

For each cluster  $P(X|e)$  is computed, also  $P(e)$  41

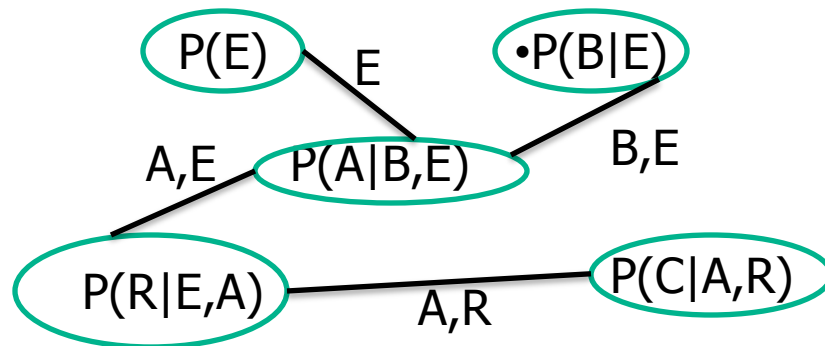
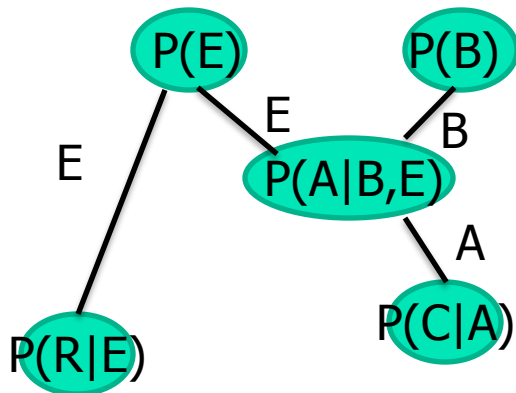
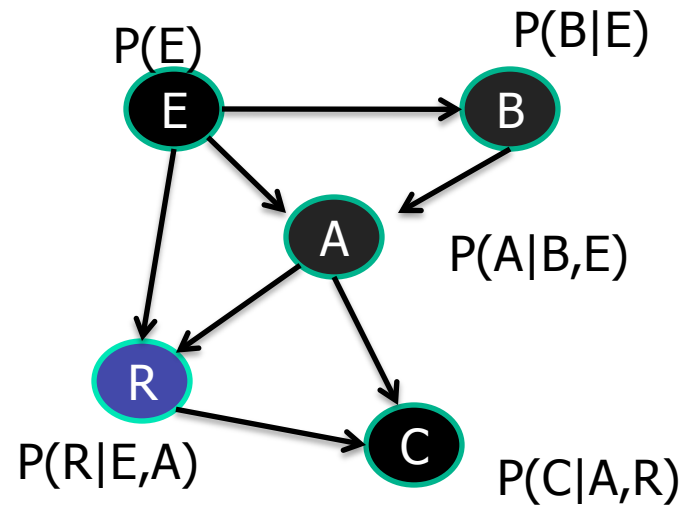
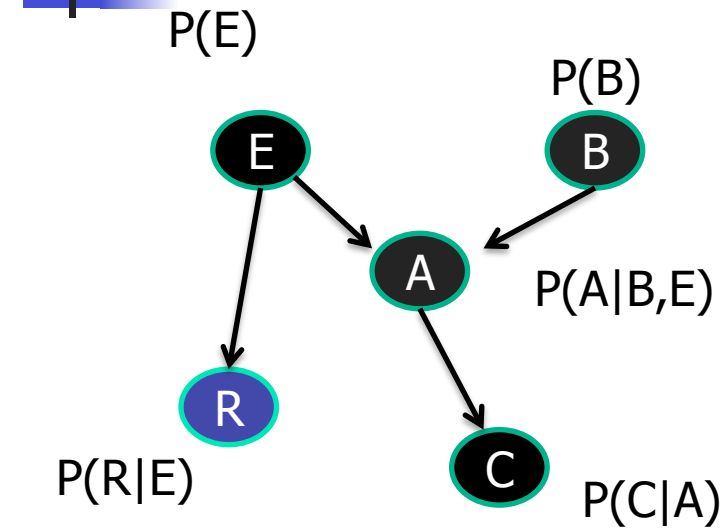


# Agenda

---

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, the join-tree algorithm
- Belief-propagation on acyclic probabilistic networks (poly-trees)
- The role of induced-width/tree-width

# 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.
- **Acyclic 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.

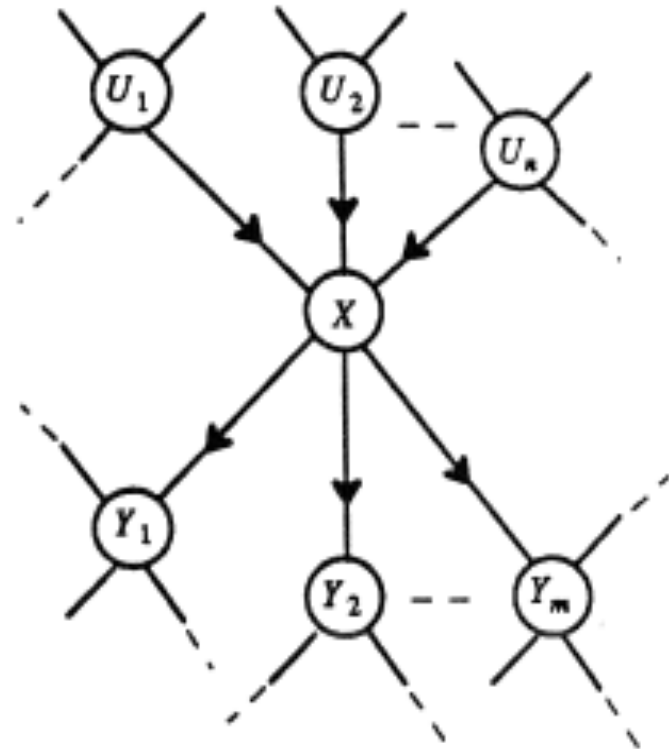
*Definition 6.4.4 (belief propagation (BP)) Given a polytree and a directed dual graph which is a poly-tree decomposition, The belief propagation algorithm (BP) is algorithm CTE, whose messages down the polytree-decomposition are called  $\lambda$  and up are called  $\pi$ .*



## A Glimpse into Pearl's BP



(a)



(b)

**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  $X \rightarrow Y_j$ ,

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

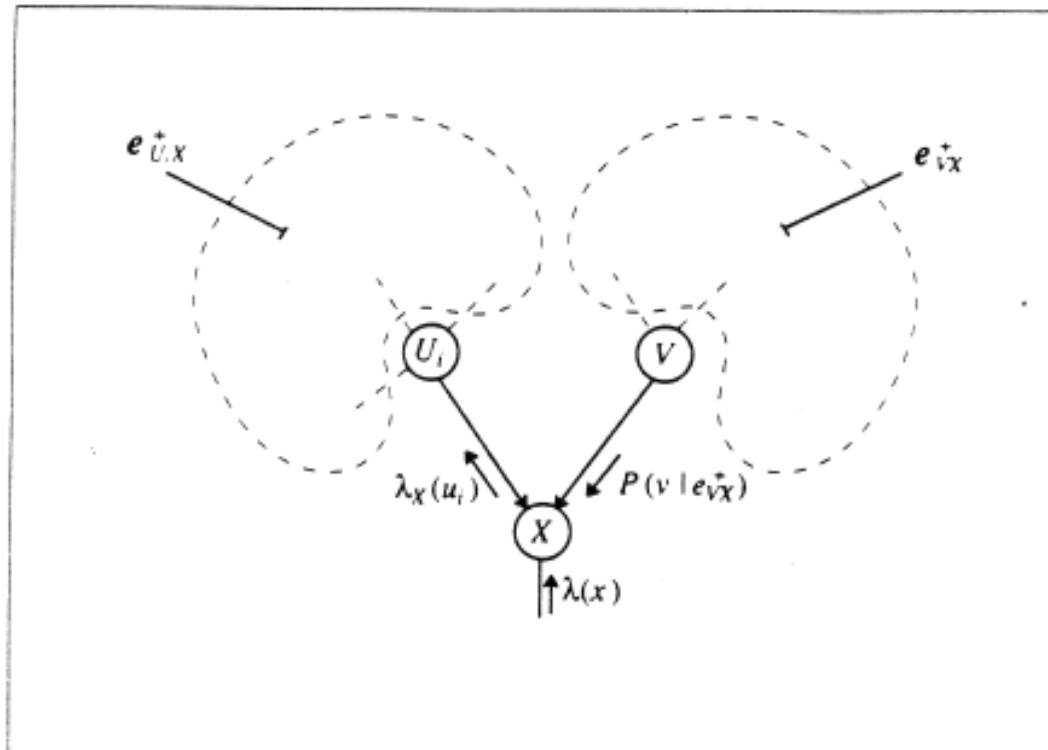


Figure 4.19. Variables, messages, and evidence sets used in the derivation of  $\lambda_X(u_i)$ .

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

$$BEL(x) = \alpha \lambda(x) \pi(x), \quad (4.49)$$

where

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

$$\pi(x) = \sum_{u_1, \dots, u_n} P(x | u_1, \dots, u_n) \prod_i \pi_X(u_i), \quad (4.51)$$

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

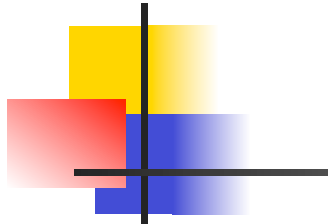
**Step 2 - Bottom-up propagation:** Using the messages received, node  $X$  computes new  $\lambda$  messages to be sent to its parents. For example, the new message  $\lambda_X(u_i)$  that  $X$  sends to its parents  $U_i$  is computed by

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

**Step 3 - Top-down propagation:** Each node computes new  $\pi$  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

$$\begin{aligned} \pi_{Y_j}(x) &= \alpha \left[ \prod_{k \neq j} \lambda_{Y_k}(x) \right] \sum_{u_1, \dots, u_n} P(x | u_1, \dots, u_n) \prod_i \pi_X(u_i) \quad (4.53) \\ &= \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)}. \end{aligned}$$

## 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,  $Y_1, \dots, Y_m$ , and  $n$  parents,  $U_1, \dots, U_n$ , as in Figure 4.18b.
- 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  $\pi$  contributed by each incoming link  $U_i \rightarrow X$ :

$$\pi_X(u_i) = P(u_i | e_{U_i X}^+). \quad (4.47)$$

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

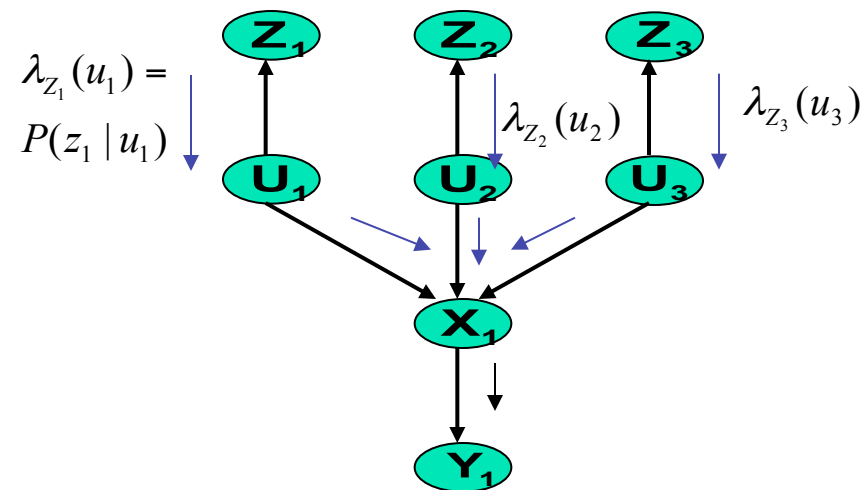
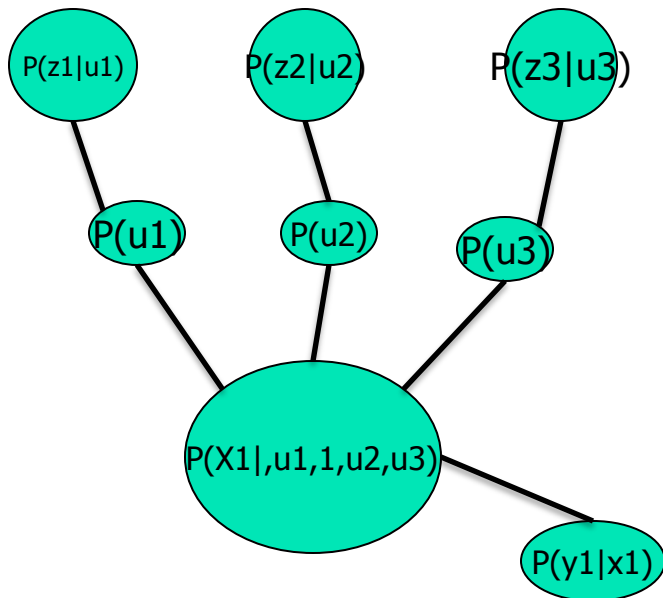
$$\lambda_{Y_j}(x) = P(e_{XY_j}^- | x). \quad (4.48)$$

3. The fixed conditional-probability matrix  $P(x | u_1, \dots, u_n)$  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

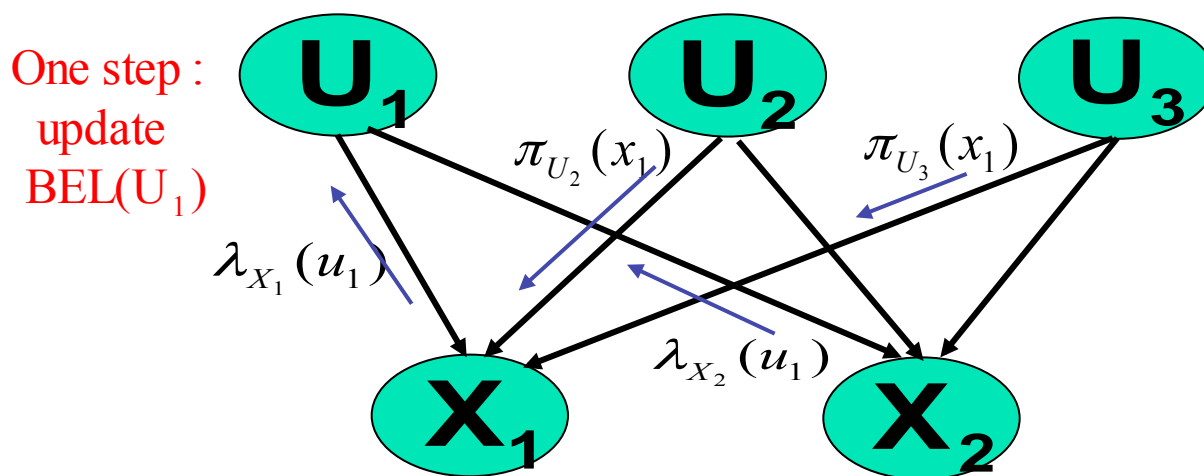


Running CTE = running Pearl's BP over the dual graph  
**Dual-graph**: nodes are cpts, arcs connect non-empty intersections.

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  $\mathcal{M} = \langle X, D, F, \prod \rangle$ .*

- *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

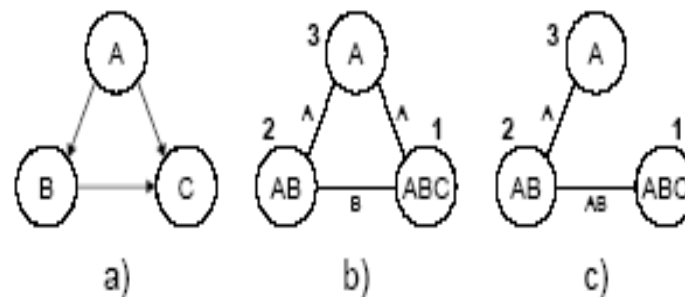


Figure 6.13: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

**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  $\mathcal{M} = \langle X, D, F, \prod \rangle$ .

**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  $DJ$ ,

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

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



# Agenda

---

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, the join-tree algorithm
- Belief-propagation on acyclic probabilistic networks (poly-trees)
- The role of induced-width/tree-width
- Conditioning with elimination (Chapter 4, class notes)



# The Idea of Cutset-Conditioning

---

We observed that when variables are assigned connectivity reduces.  
The magnitude of saving is reflected through the “conditioned-induced graph”

Cutset-conditioning exploit this in a systematic way:  
Select a subset of variables, assign them values, and solve the conditioned problem by elimination.  
Repeat for all assignments to the cutset.

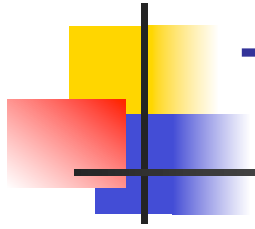
Algorithm VEC



# Exact Reasoning by Search

---

- Enumeration in VEC can be done by dfs
- In the extreme we can do only search



# The Principle of Cutset Conditioning

---

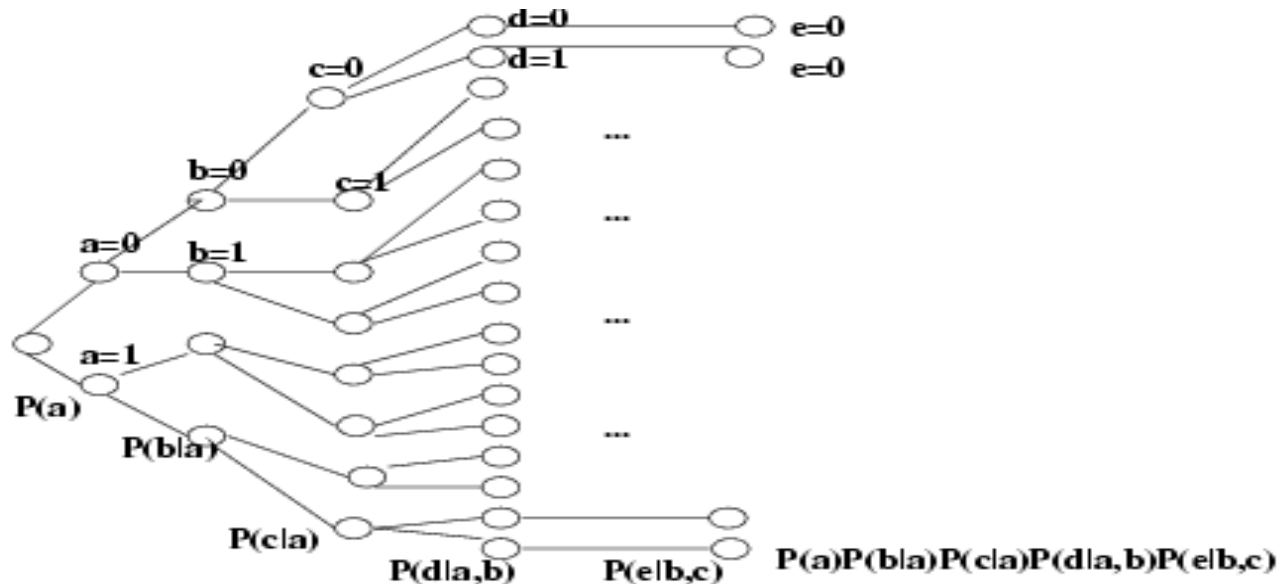
Enumeration in VEC can be done by dfs  
In the extreme we can do only search, we

For example, we can compute expression below for the probability of evidence in the network of Figure 2.4 by traversing the search-tree in Figure 4.18 along the ordering, from first variable to last variable.

$$\begin{aligned} Bel(A = a) &= \sum_{c,b,f,d,g} P(g|f)P(f|b,c)P(d|a,b)P(c|a)P(b|a)P(a) \\ &= P(a) \sum_c P(c|a) \sum_b P(b|a) \sum_f P(f|b,c) \sum_d P(d|b,a) \sum_g P(g|f), \end{aligned} \quad (6.1)$$

# Conditioning generates the probability tree

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



Complexity of conditioning: exponential time, linear space



## Algorithm VEC (Variable-elimination with conditioning)

### ALGORITHM VEC-EVIDENCE

**Input:** A belief network  $\mathcal{B} = \langle \mathcal{X}, \mathcal{D}, \mathcal{G}, \mathcal{P} \rangle$ , an ordering  $d = (x_1, \dots, x_n)$ ; evidence  $e$  over  $E$ , a subset  $C$  of conditioned variables;

**output:** The probability of evidence  $P(e)$

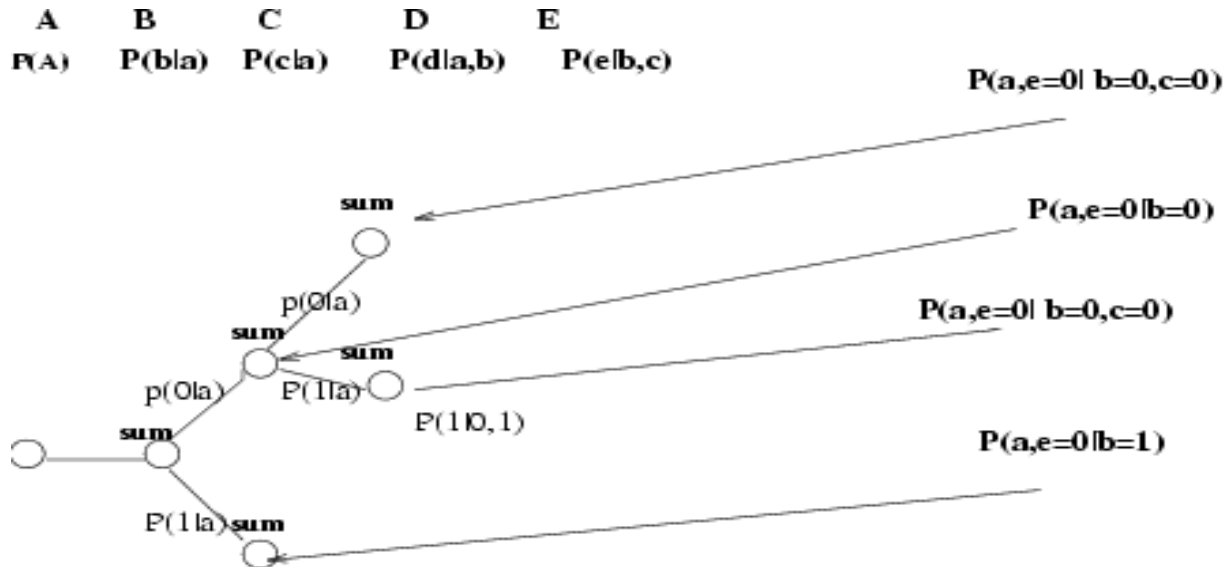
**Initialize:**  $\lambda = 0$ .

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**  $P(e) = \alpha \cdot \lambda$  ( $\alpha$  is a normalization constant. )



# Conditioning+Elimination

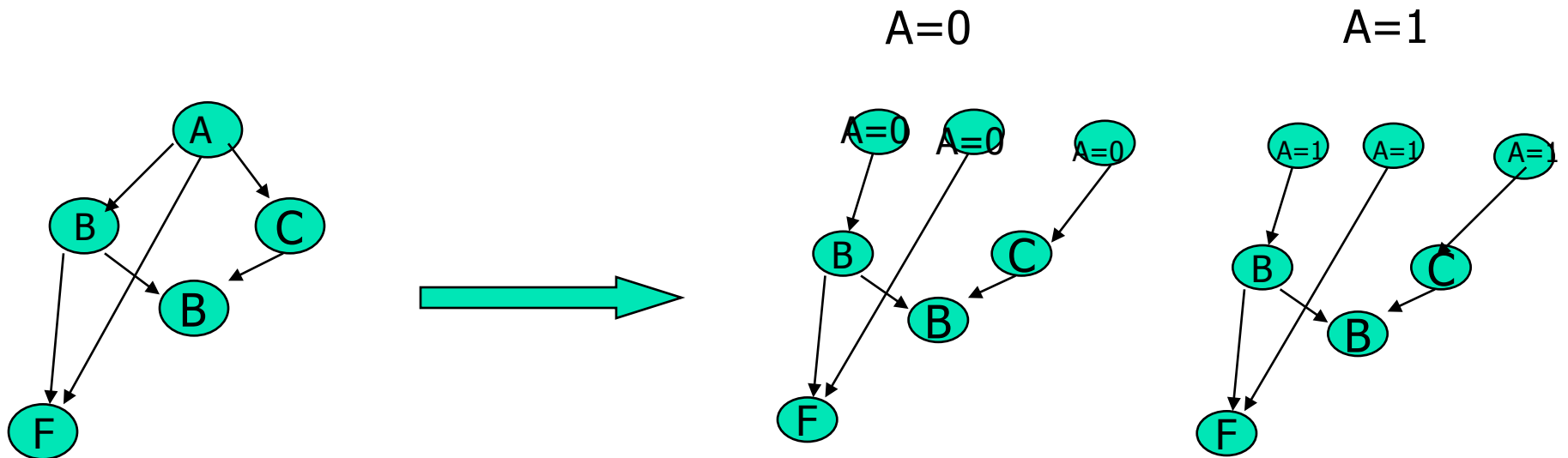
$$P(a, e = 0) = P(a) \sum_b P(b|a) \sum_c P(c|a) \sum_d P(d|a,b) \sum_{e=0} P(e|b,c)$$



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

# Loop-cutset decomposition

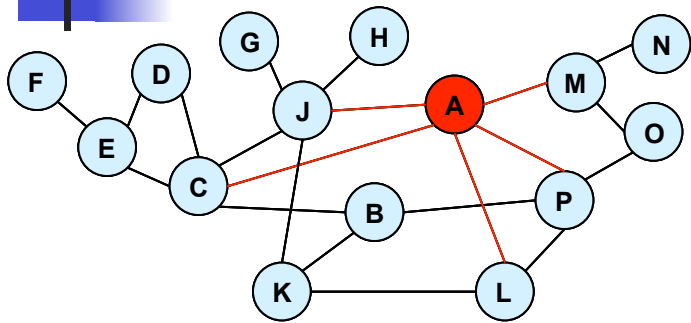
- You condition until you get a polytree



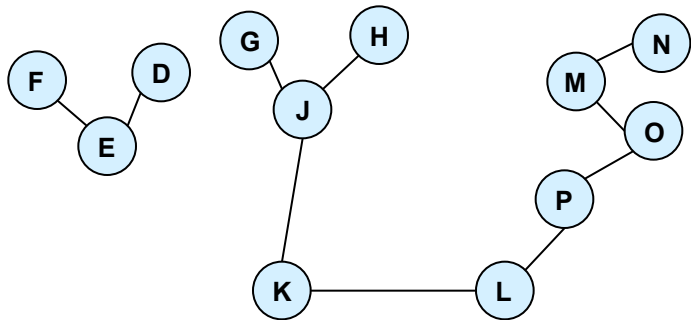
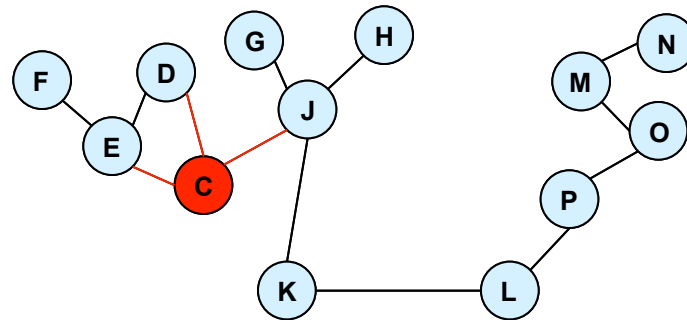
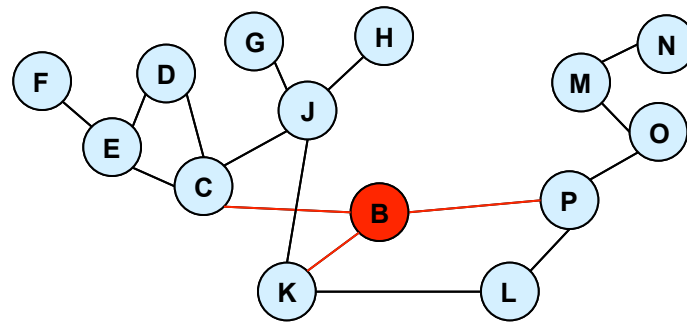
$$P(B|F=0) = P(B, A=0|F=0) + P(B, A=1|F=0)$$

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

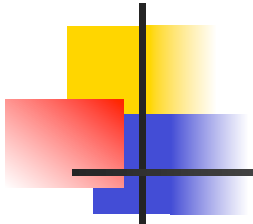
# Conditioning and Cycle cutset



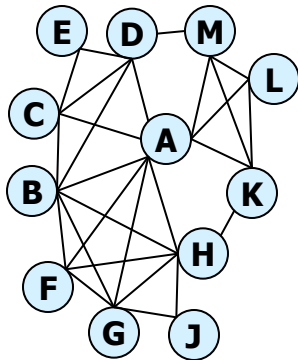
Cycle cutset = {A,B,C}



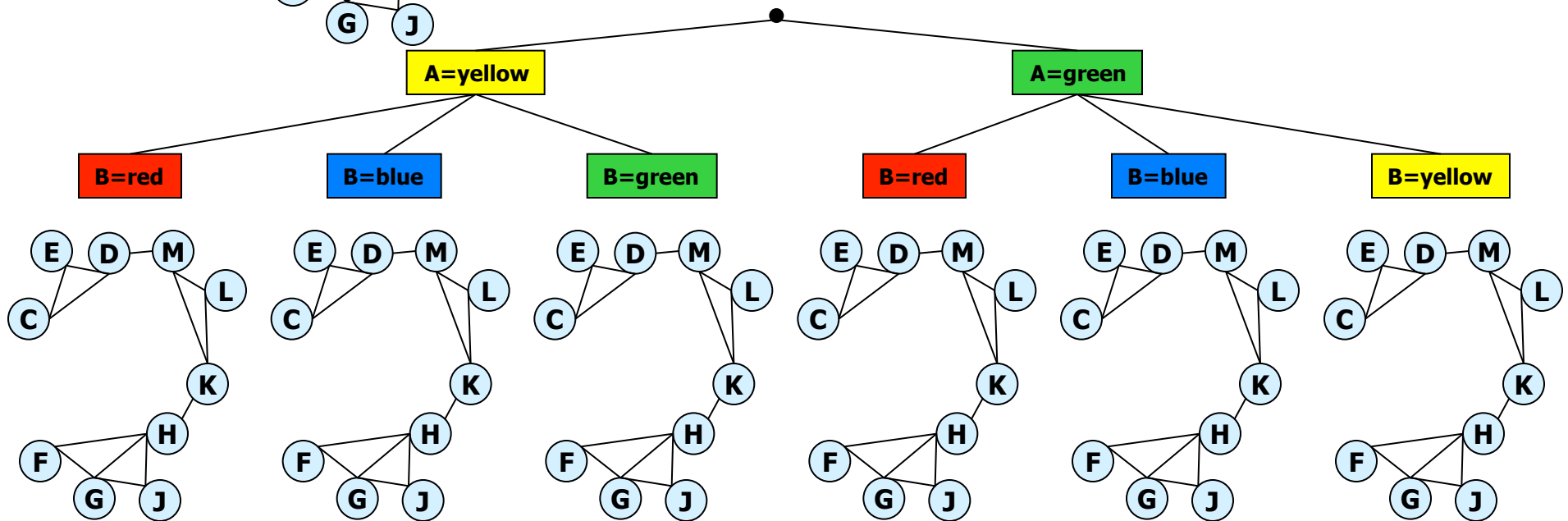
# Search over the Cutset (cont)



Graph Coloring problem



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





## Variable elimination with conditioning; w-cutset algorithms

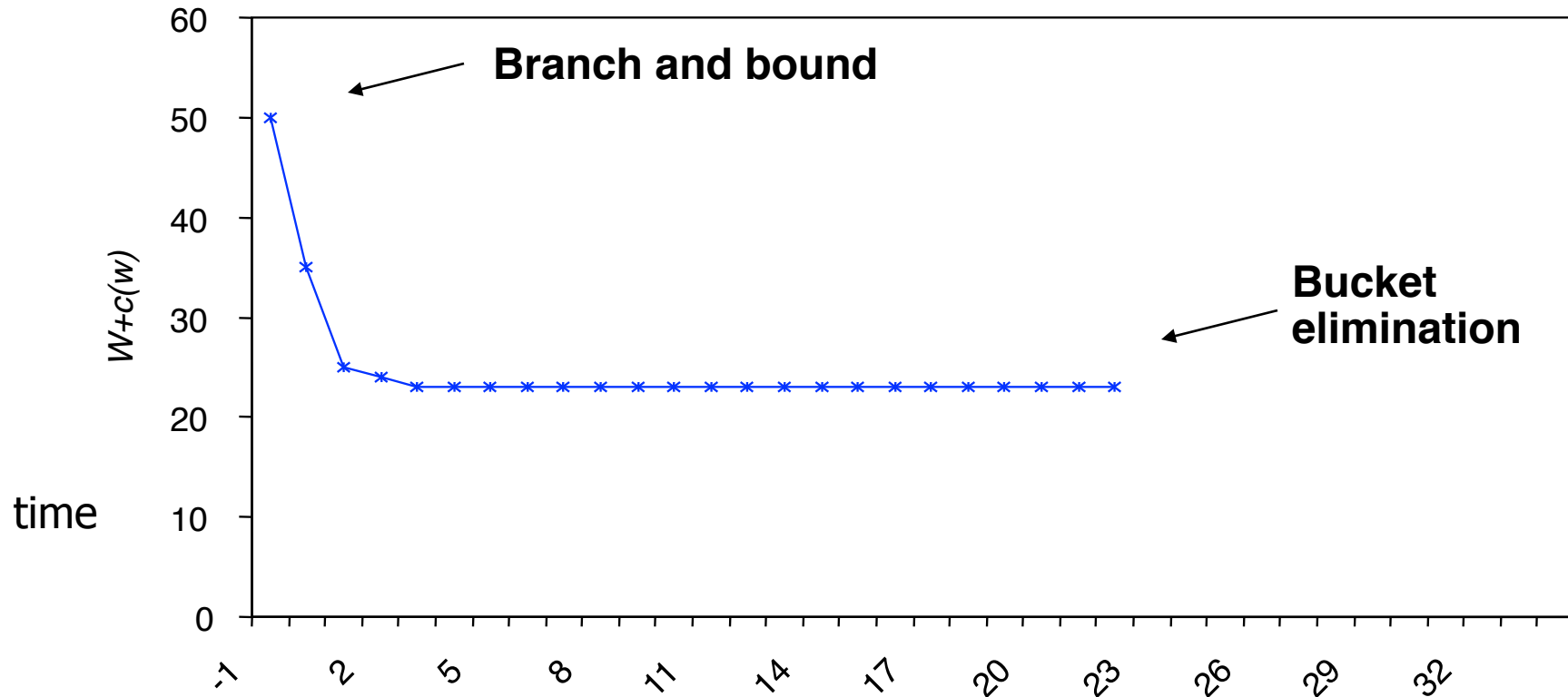
---

- VEC-bel:
- Identify a w-cutset,  $c_w$ , of the network
- For each assignment to the cutset solve by CTE or BE 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,  $w^* \approx 23$ )



W-cutset time  $O(\exp(w + \text{cutset-size}))$   
Space  $O(\exp(w))$

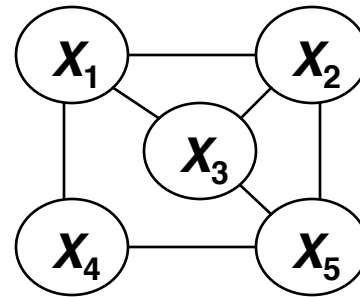


## Hybrid of Variable-elimination and conditioning-Search

---

- Tradeoff space and time

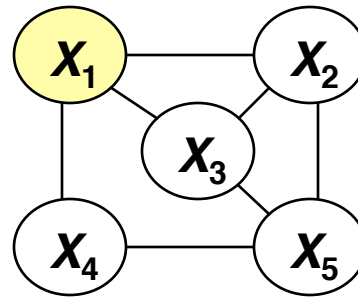
# Search Basic Step: Conditioning



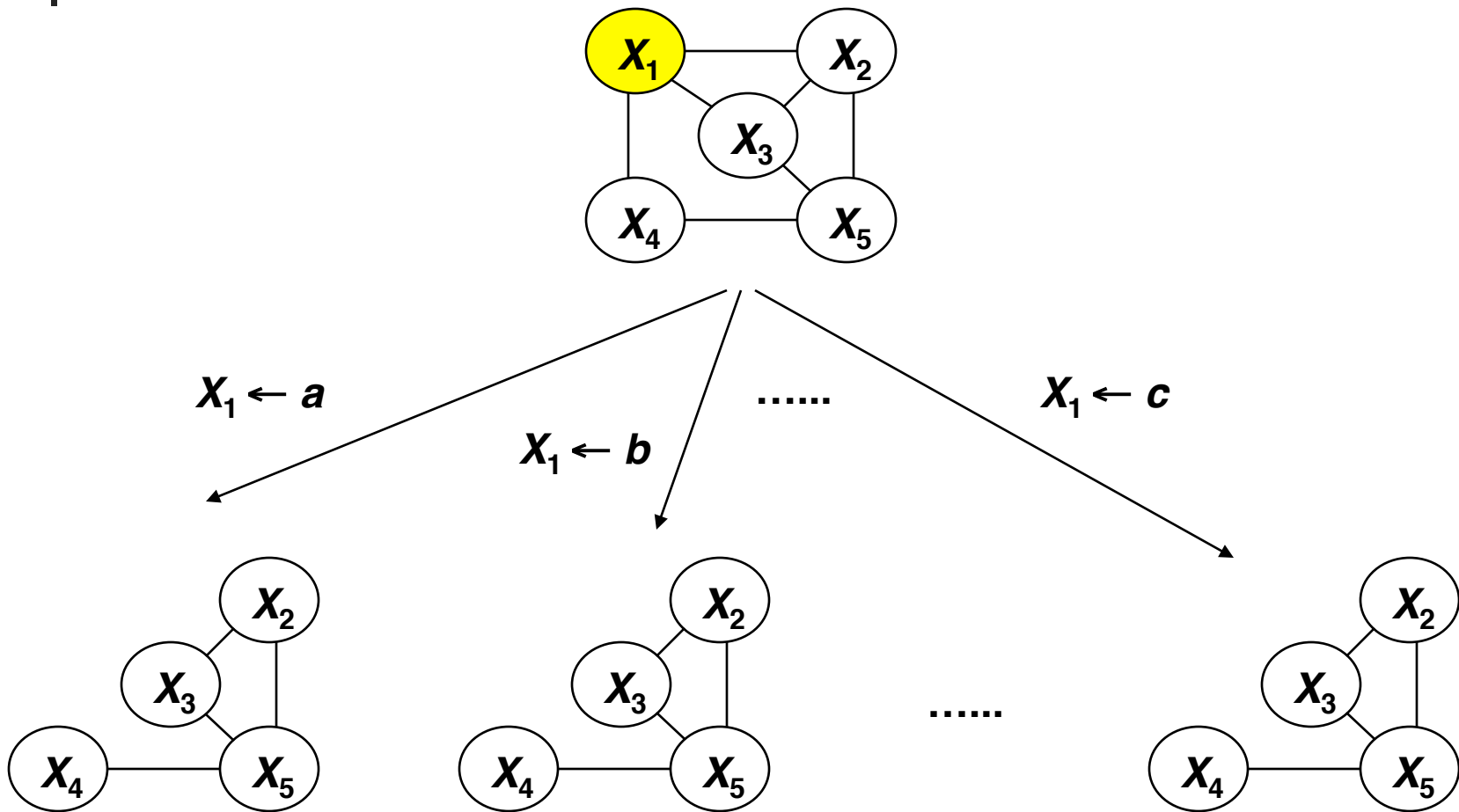


# Search Basic Step: Conditioning

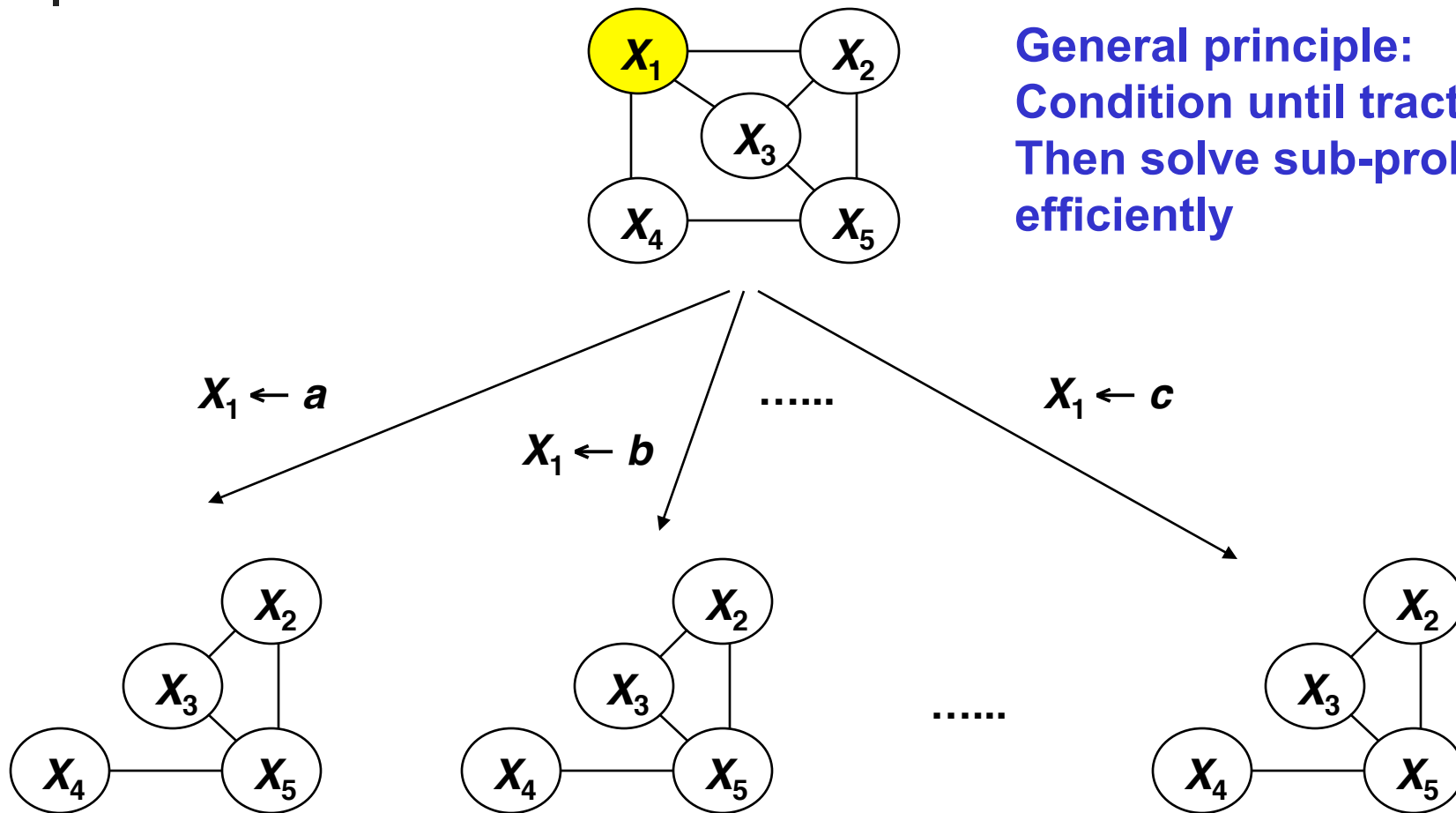
- Select a variable



# Search Basic Step: Conditioning

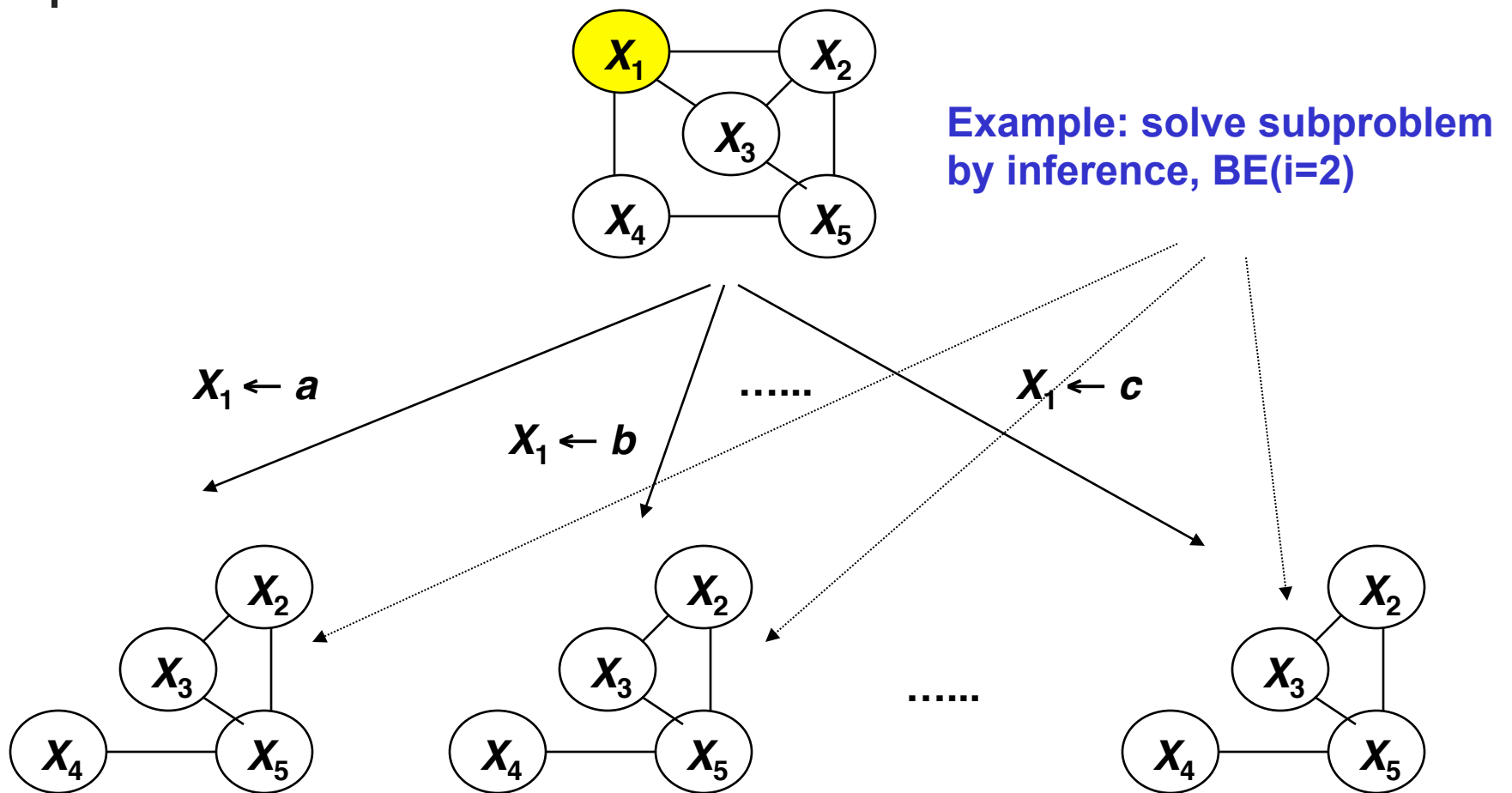


# Search Basic Step: Variable Branching by Conditioning



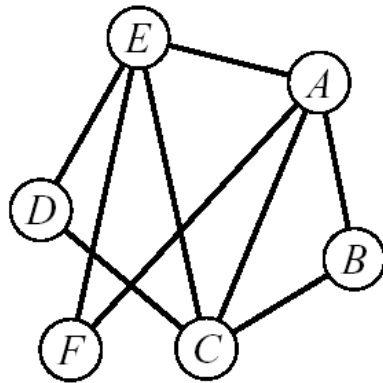
**General principle:  
Condition until tractable  
Then solve sub-problems  
efficiently**

# Search Basic Step: Variable Branching by Conditioning

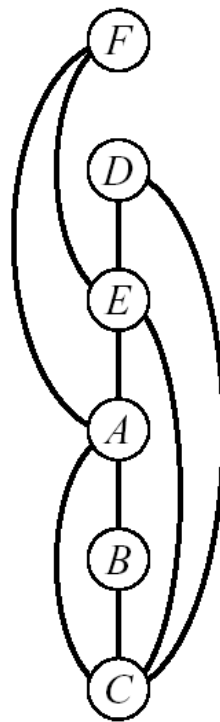


# The Cycle-Cutset Scheme: Condition Until Treeness

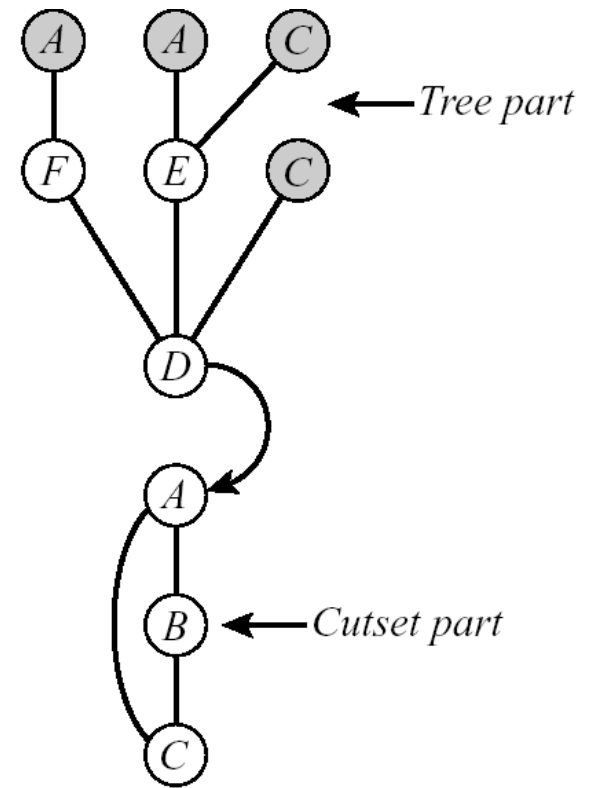
- Cycle-cutset
- $i$ -cutset
- $C(i)$ -size of  $i$ -cutset



(a)



(b)



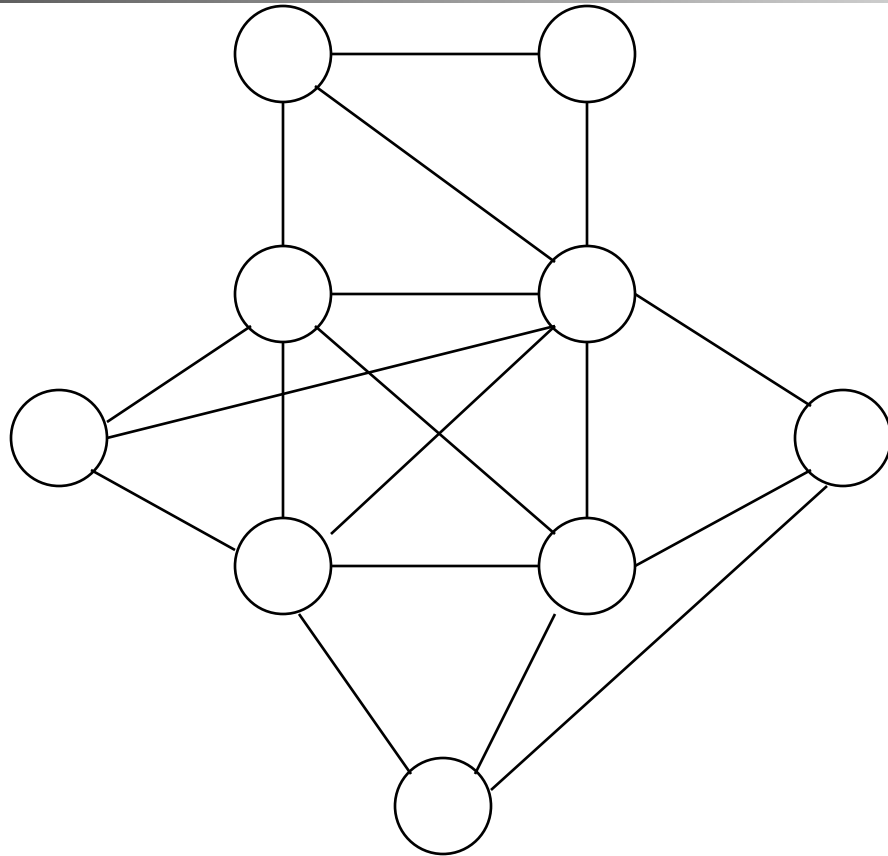
(c)

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

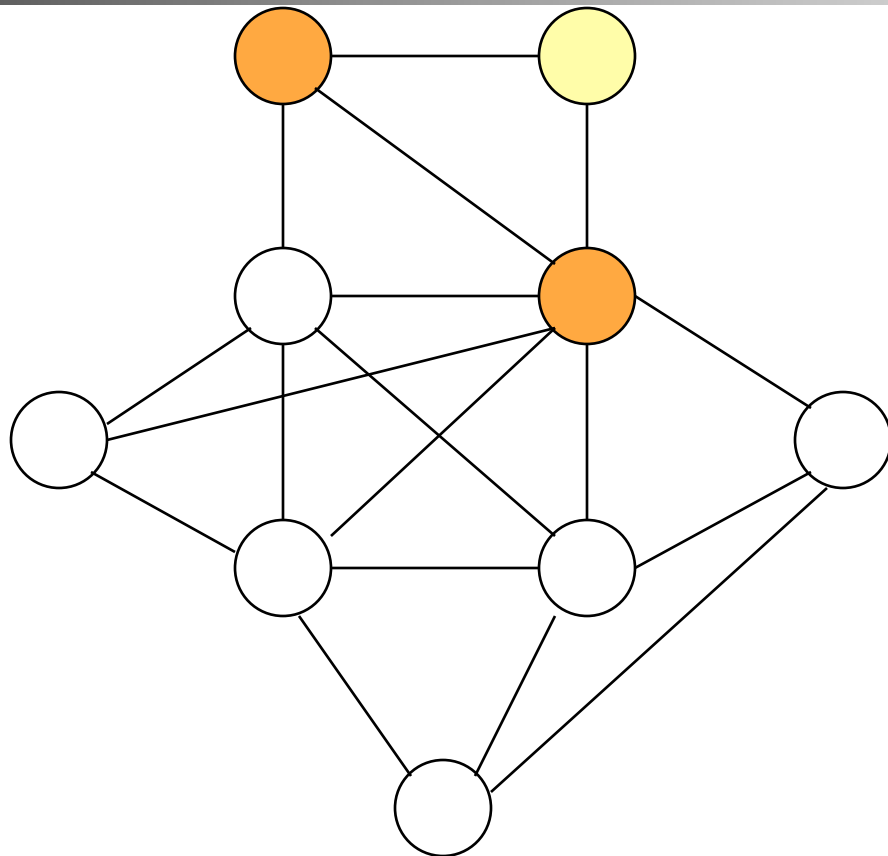


# Eliminate First

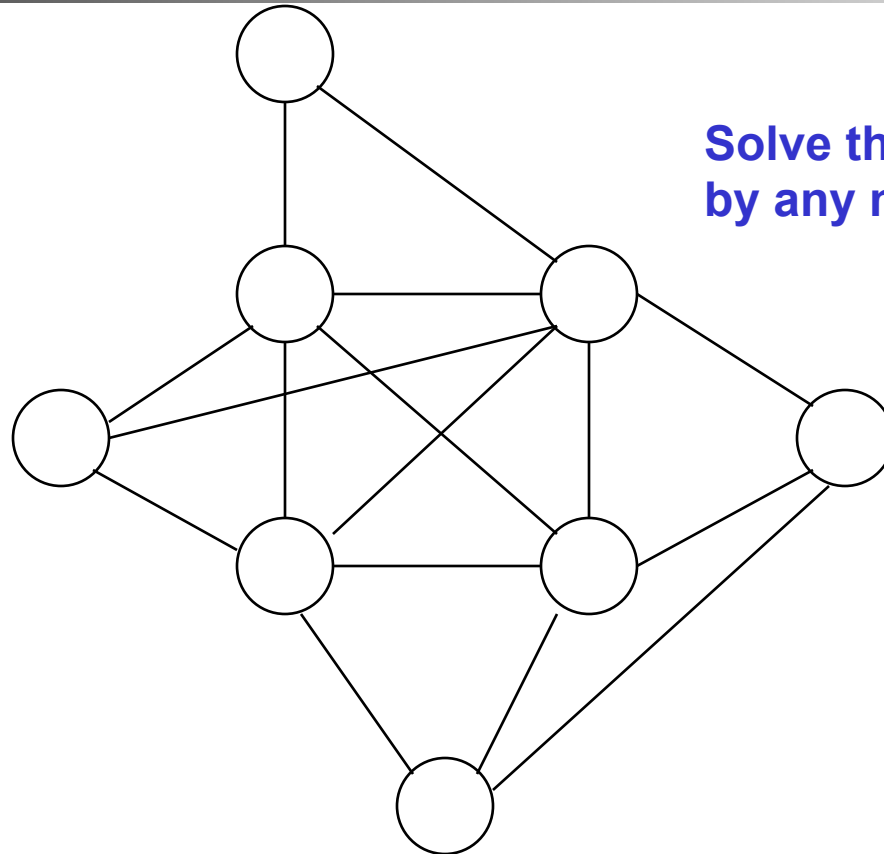
---



# Eliminate First



# Eliminate First



**Solve the rest of the problem  
by any means**





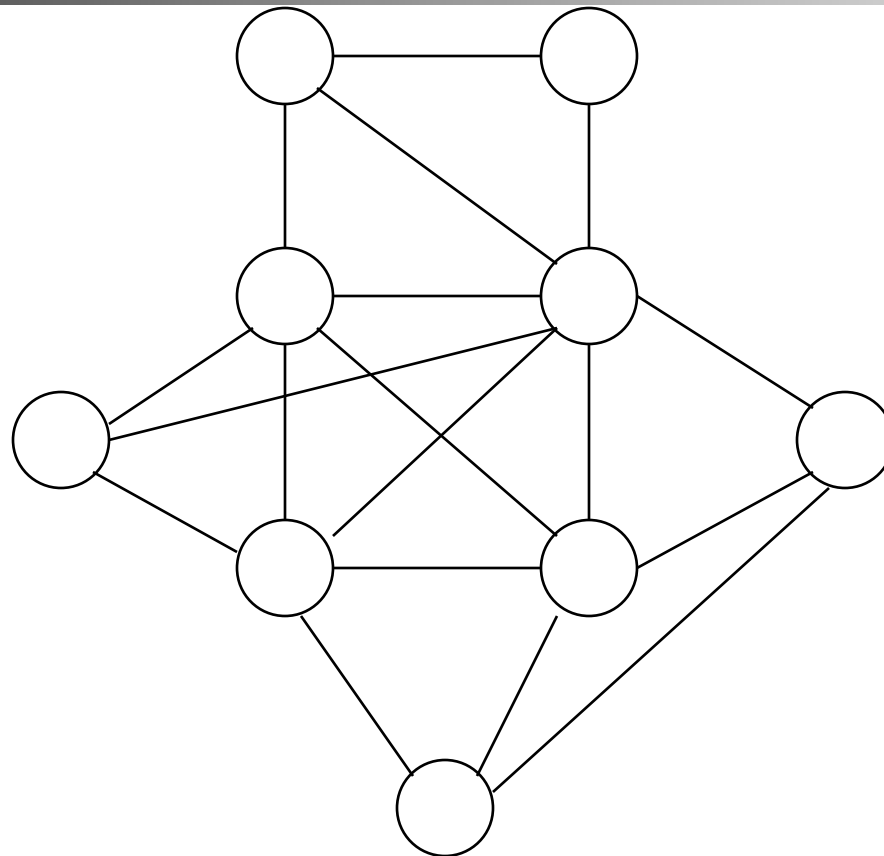
# Hybrids Variants

---

- **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

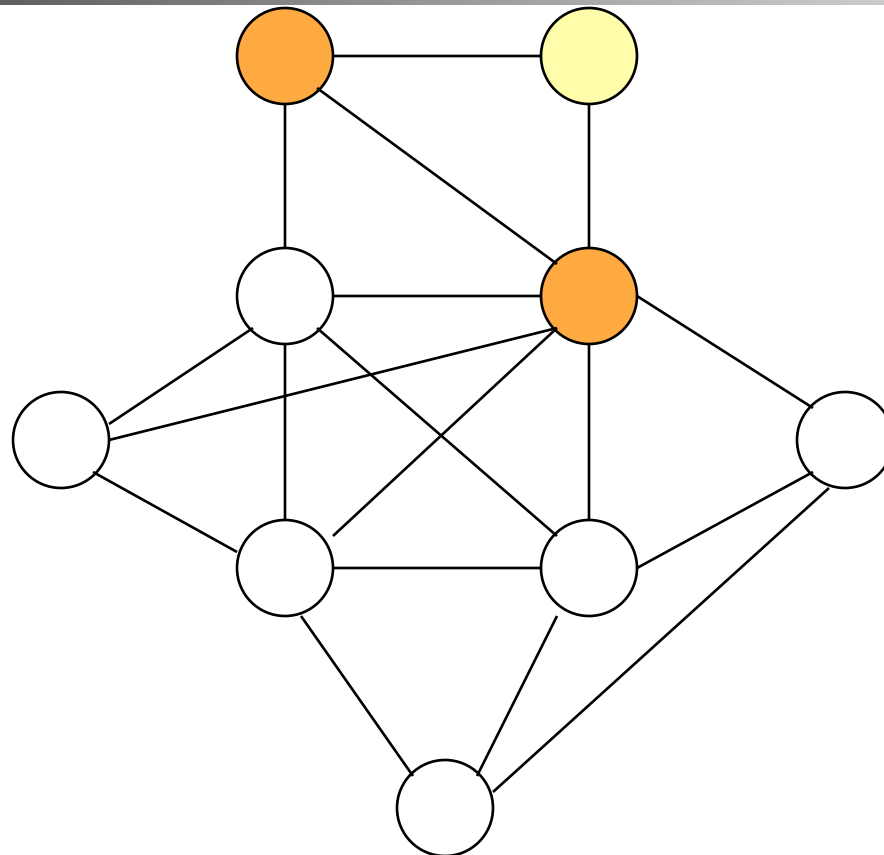
(Larrosa & Dechter, CP'02)





# Interleaving Conditioning and Elimination

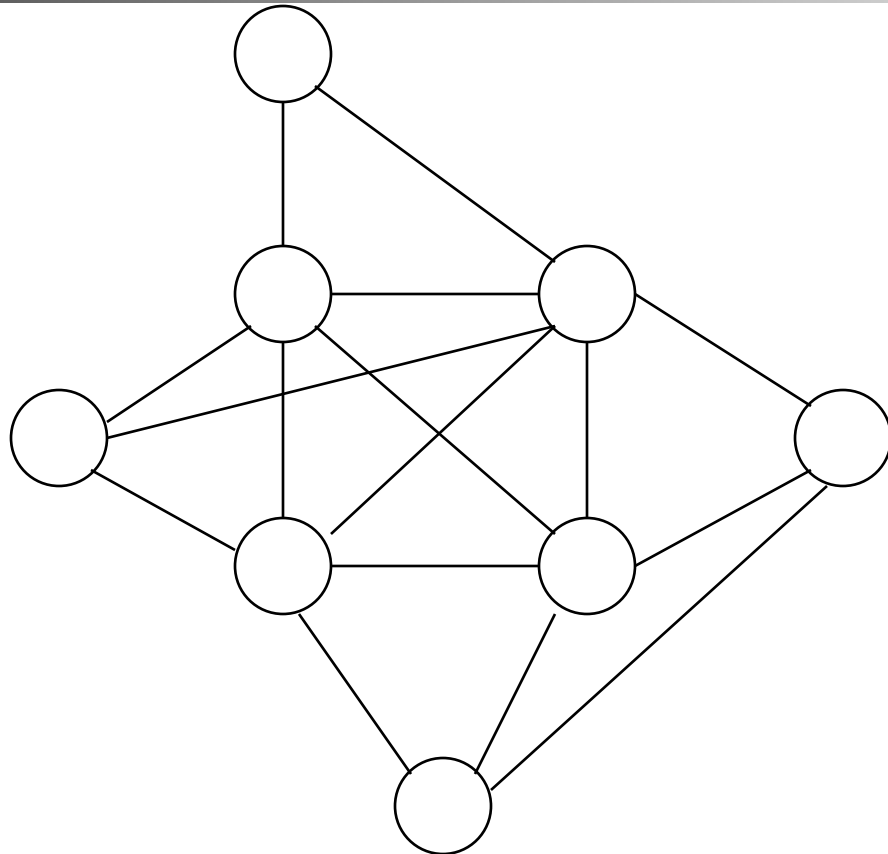
---





# Interleaving Conditioning and Elimination

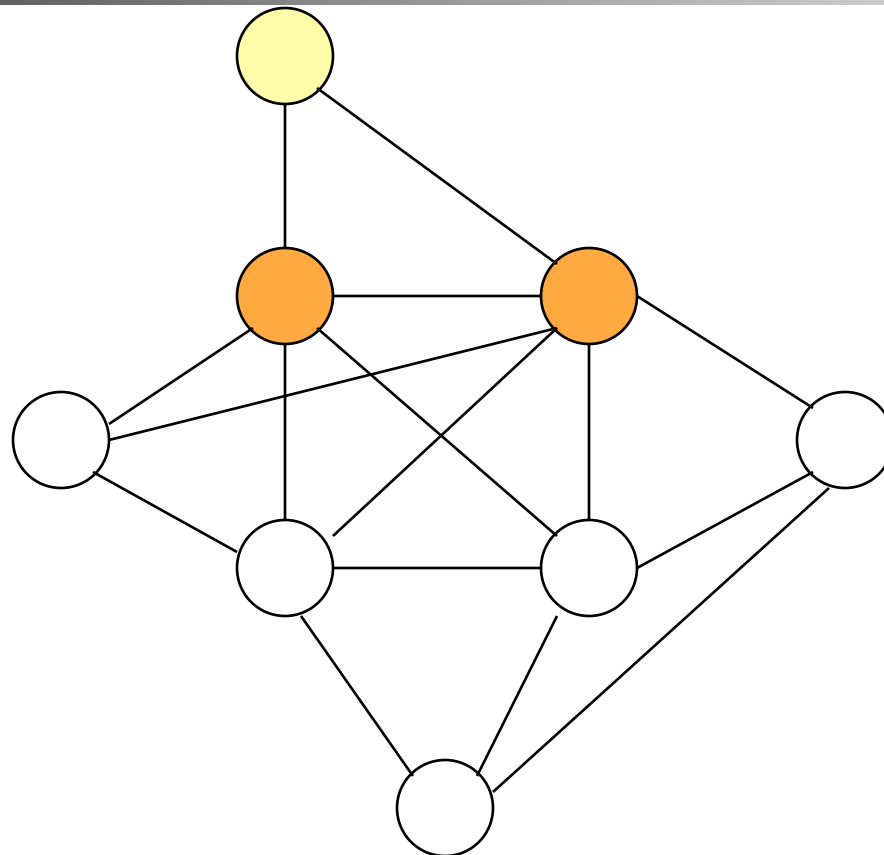
---





# Interleaving Conditioning and Elimination

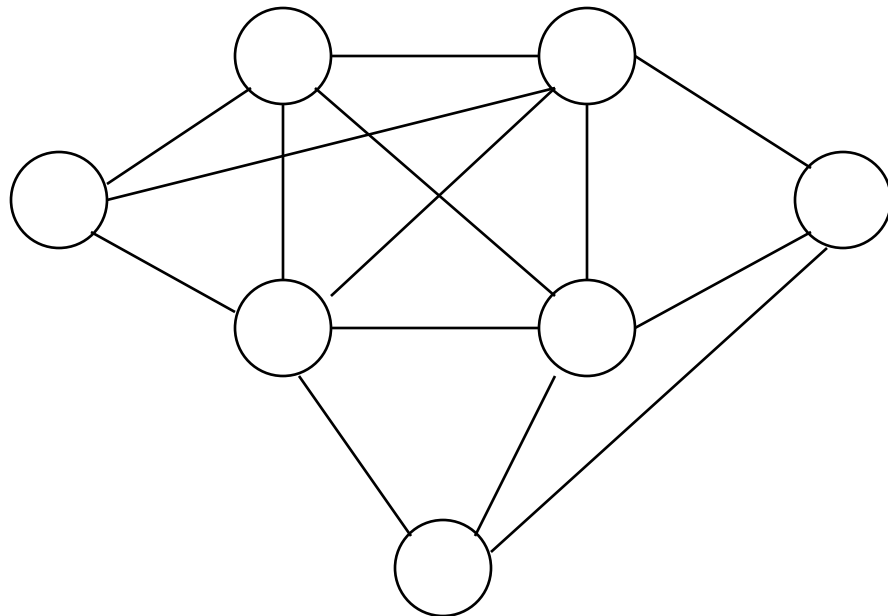
---





# Interleaving Conditioning and Elimination

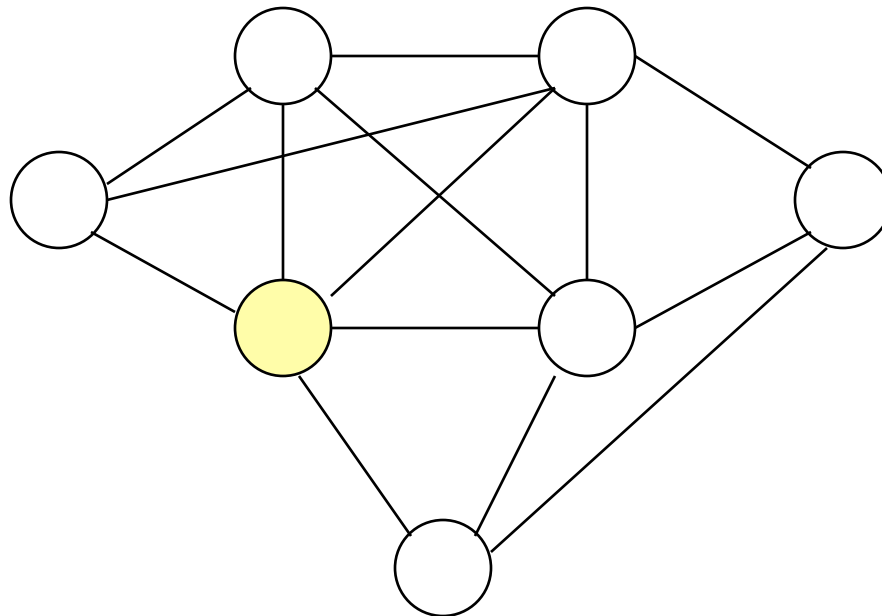
---

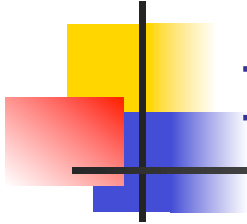




# Interleaving Conditioning and Elimination

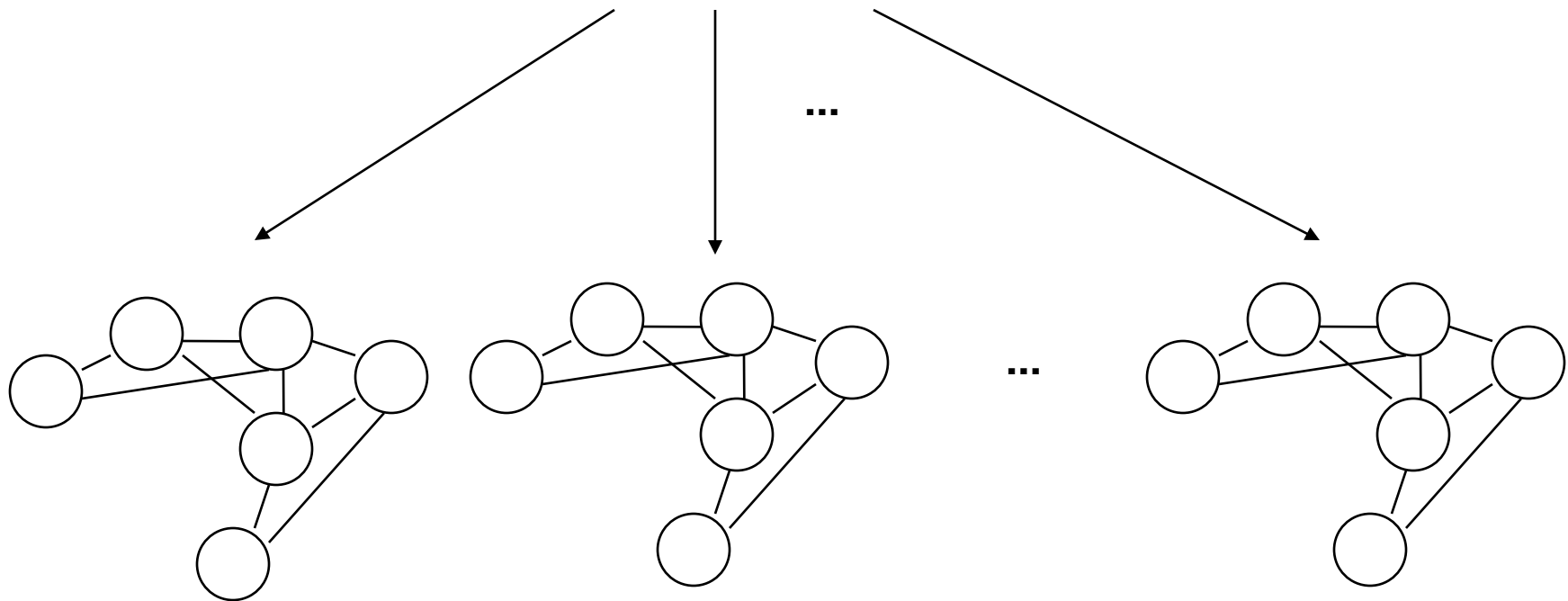
---





# Interleaving Conditioning and Elimination

---







# 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?

# Properties of conditioning + elimination

**Definition 5.6.1 (cycle-cutset,  $w$ -cutset)** *Given a graph  $G$ , a subset of nodes is called a  $w$ -cutset iff when removed from the graph the resulting graph has an induced-width less than or equal to  $w$ . A minimal  $w$ -cutset of a graph has a smallest size among all  $w$ -cutsets of the graph. A cycle-cutset is a 1-cutset of a graph.*

A cycle-cutset is known by the name a *feedback vertex set* and it is known that finding the minimal such set is NP-complete [41]. However, we can always settle for approximations, provided by greedy schemes. Cutset-decomposition schemes call for a new optimization task on graphs:

**Definition 5.6.2 (finding a minimal  $w$ -cutset)** *Given a graph  $G = (V, E)$  and a constant  $w$ , find a smallest subset of nodes  $U$ , such that when removed, the resulting graph has induced-width less than or equal  $w$ .*



# Tradeoff between cutset and elimination

---

It can be shown that the size of the smallest cycle-cutset (1-cutset),  $c_1^*$  and the smallest induced width,  $w^*$ , obey the inequality  $c_1^* \geq w^* - 1$ . Therefore,  $1 + c_1^* \geq w^*$ . In general, if  $c_w^*$  is the size of a minimal  $w$ -cutset then,

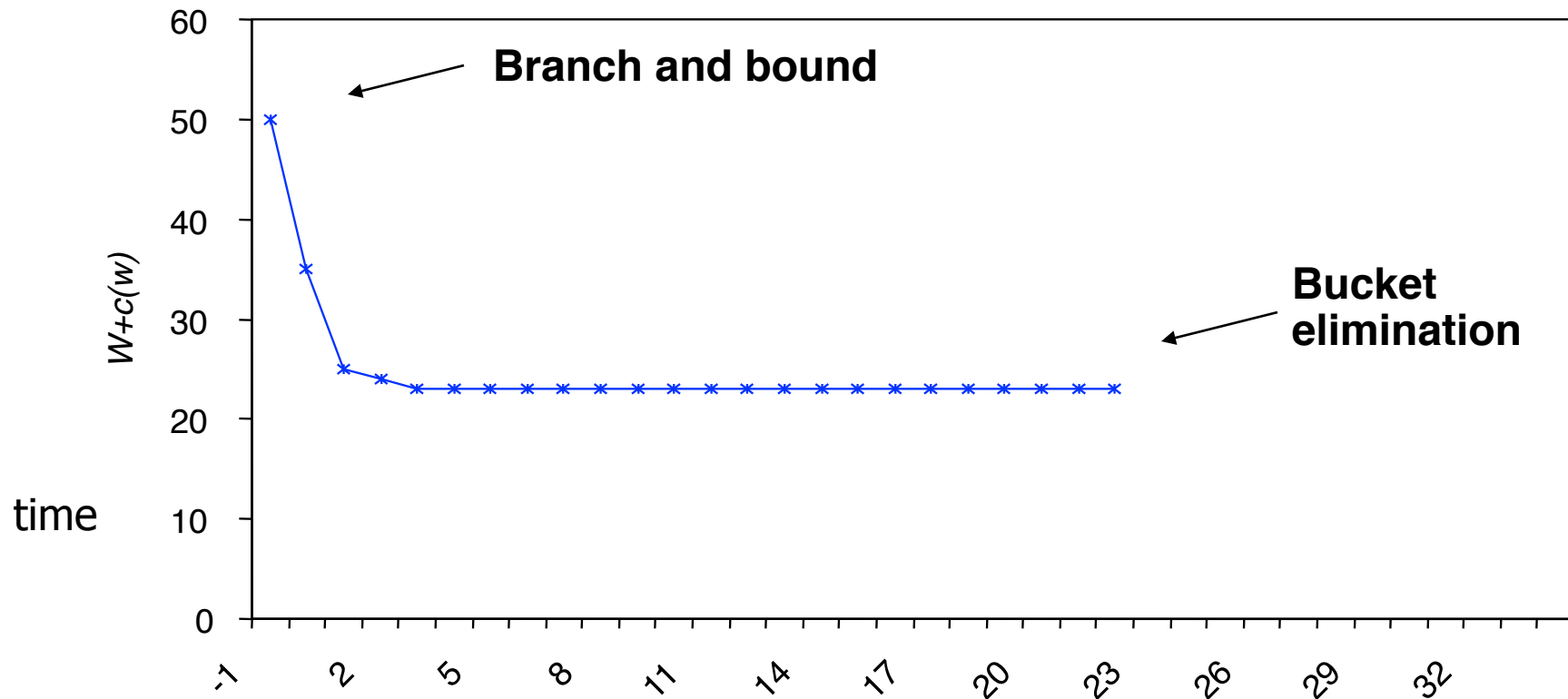
## Theorem 5.6.3

$$1 + c_1^* \geq 2 + c_2^* \geq \dots b + c_b^*, \dots \geq w^* + c_{w^*}^* = w^*$$

# 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,  $w^* \approx 23$ )



W-cutset time  $O(\exp(w+\text{cutset-size}))$   
Space  $O(\exp(w))$