Exact Inference Algorithms for Probabilistic Reasoning; BTE and CTE

COMPSCI 276, Fall 2014

Set 6: Rina Dechter

(Reading: Primary: Dechter chapter 5
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)

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

- Finding maximum a-posteriory hypothesis

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

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

- Finding maximum-expected-utility (MEU) decision

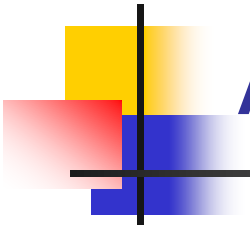
$$(d_1^*,\ldots,d_k^*) = \arg\max_{\bar{d}} \sum_{X/D} \mathbf{P}(\bar{x}, e)\mathbf{U}(\bar{x})$$

$D \subseteq X$: **decision variables**
$U(\bar{x})$: **utility function**

1998 roadmap

Outline; Road Map

Methods \ Tasks	CSP	SAT	Optimi-zation	Belief updating	MPE, MAP, MEU	Solving linear equalities/ inequalities
elimination	adaptive consistency join-tree	directional resolution	dynamic program-ming	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	DCDR, BDR-DP		loop-cutset		
approximate elimination	i-consistency	bounded (directional) resolution	mini-buckets	mini-buckets	mini-buckets	
approximate conditioning	greedylocal search (GSAT)	GSAT	gradient descent	stochastic simulation	gradient descent	
approximate (elimination + conditioning)	GSAT + partial path-consistency					

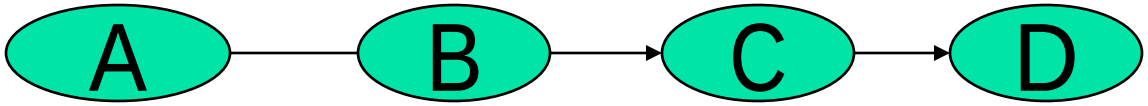


Agenda

- 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

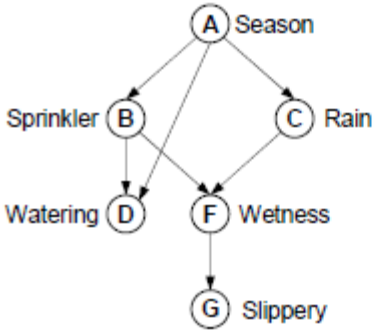
A simple network

Given:

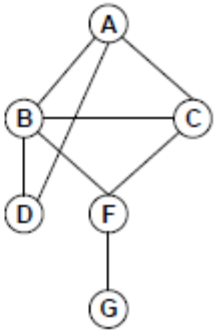


- How can we compute P(D)?, P(D|A=0)? P(A|D=0)?
- Brute force O(k^4)
- Maybe O(4k^2)

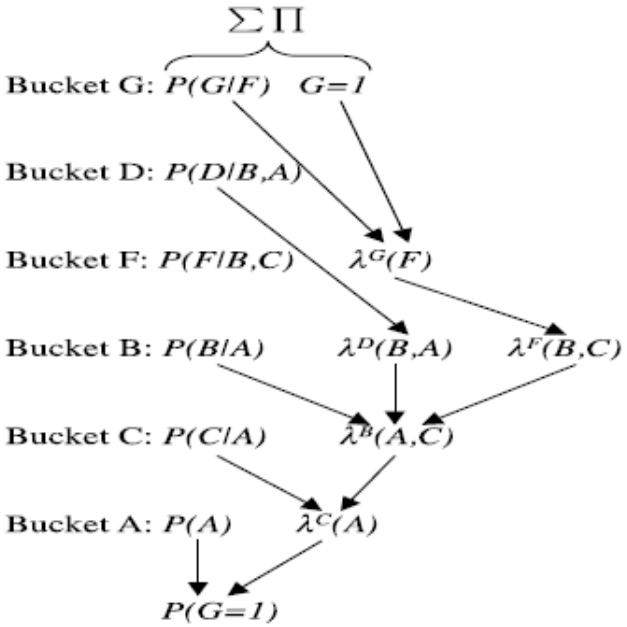
A Bayesian Network Processed by BE

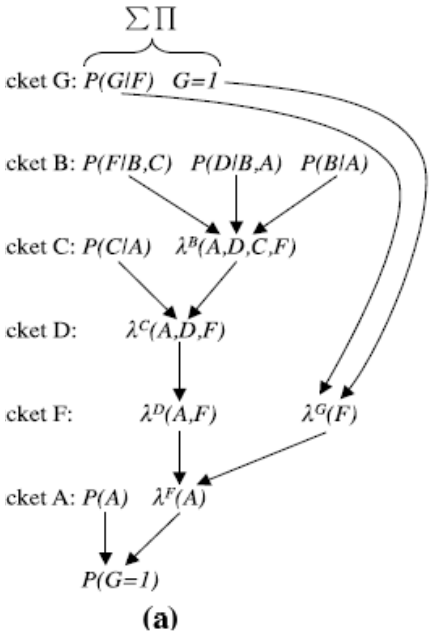


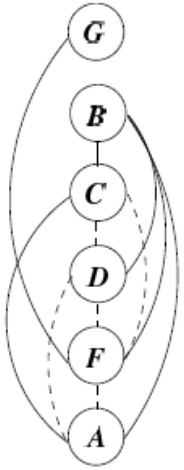
(a) Directed acyclic graph



(b) Moral graph





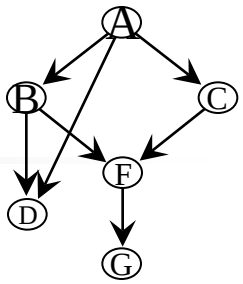


The bucket's output when processing along $d_2 = A, F, D, C, B, G$

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

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

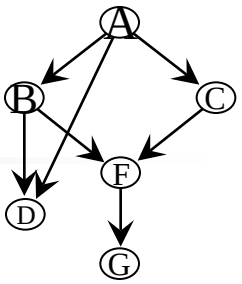
Bucket C: $P(C|A)$ $\lambda_{F\to C}(B,C)$

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

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

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: $P(G|F)$

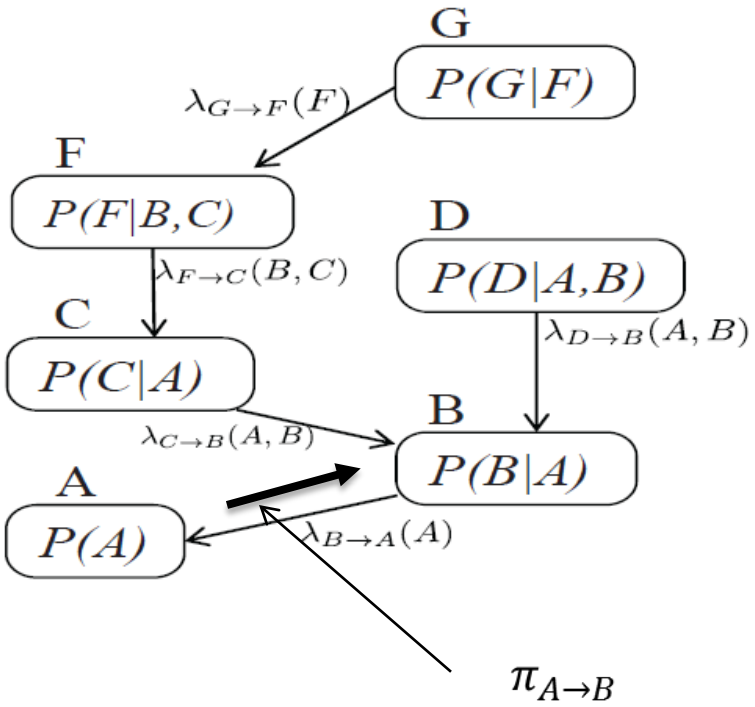
Bucket F: $P(F|B,C)$ $\lambda_{G\to F}(F)$

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

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

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

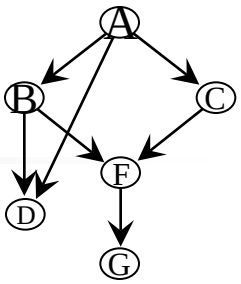
Bucket A: $P(A)$ $\lambda_{B\to A}(A)$



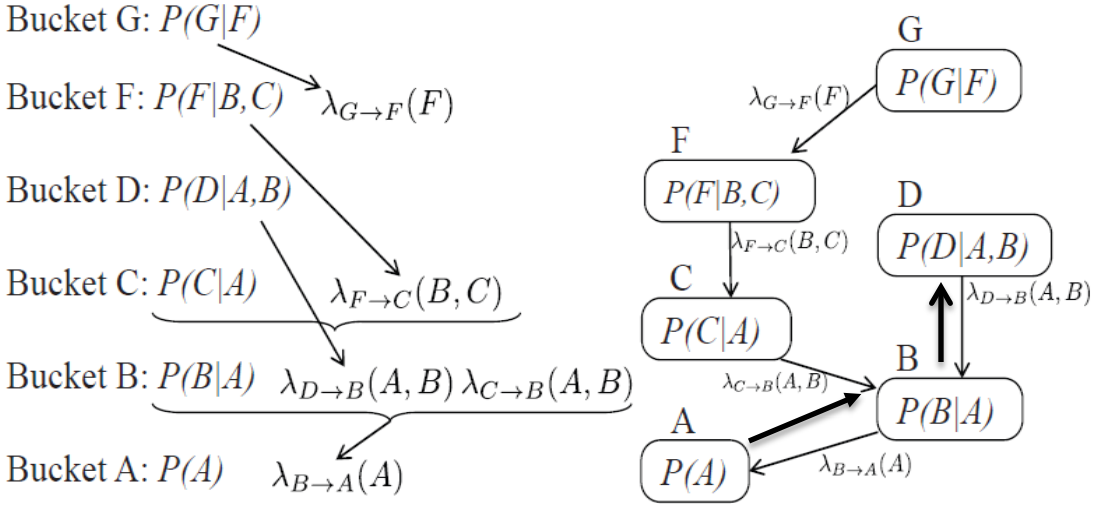
$$\pi_{A\to B}(a) = P(A),$$

$$P(B) = \sum P(B \mid A)P(A)\lambda_{C->B}(B,C)\lambda_{D->B}(A,B)$$

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)



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

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

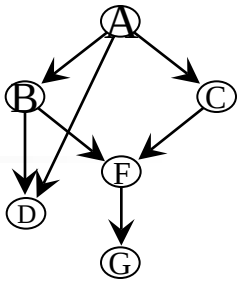
$$bel(d) = \alpha \sum_{a,b} P(d|a,b)\cdot \pi_{B\to D}(a,b).$$

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 π. [illegible] 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



Bucket G: $P(G/F)$ → $\pi_F^G(F)$

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

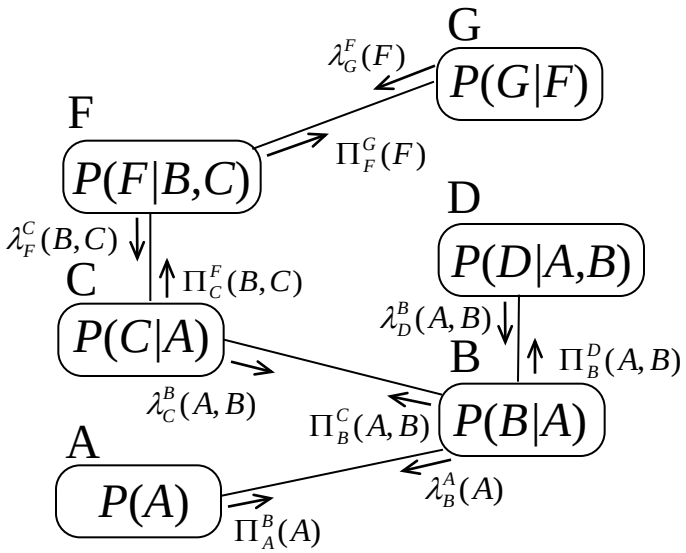
Bucket D: $P(D/A,B)$ → $\pi_B^D(A,B)$

Bucket C: $P(C/A)$ $\lambda_F^C(B,C)$ → $\pi_B^C(A,B)$

Bucket B: $P(B/A)$ $\lambda_D^B(A,B)$ $\lambda_C^B(A,B)$ → $\pi_A^B(A)$

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

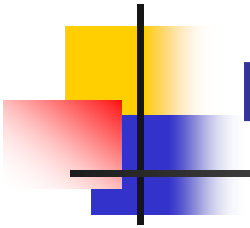
Each bucket can Compute its marginal probability



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

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_i to B_j iff the function created at bucket B_i is placed at bucket B_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

BTE

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 $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, ordering d. $X = \{X_1, ..., X_n\}$ and $F = \{f_1, ..., f_r\}$ Evidence $E = e$.

Output: Augmented buckets $\{B'_i\}$, containing the original functions and all the π and λ 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:** λ messages (BE) **do**
   for $i = n$ to 1, in reverse order of d process bucket B_i:
   The message $\lambda_{i \to j}$ from B_i to its parent B_j, is:
   λ [redacted] $\lambda_{k \to i}$
   endfor
3. **bottom-up phase:** π **messages**
   for $j = 1$ to n, process bucket B_j **do**:
   B_j takes $\pi_{k \to j}$ received from its parent B_k, and computes a message $\pi_{j \to i}$ for each child bucket B_i by
   [redacted] $\lambda_{r \to j}$
   endfor
4. **Output:** and answering singleton queries (e.g., deriving beliefs).
   Output augmented buckets $B'_1, ..., B'_n$, where each B'_i contains the original bucket functions and the λ and π messages it received.

Query answering

COMPUTING MARGINAL BELIEFS

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

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

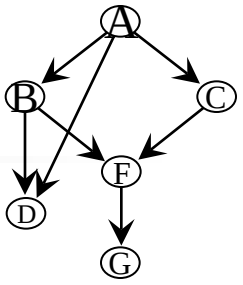
$$bel(B_i) \Leftarrow \prod_{f \in B'_i} f$$

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

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

Figure 6.4: Query answering

BTE: Allows Messages Both Ways



Bucket G: $P(G/F)$ → $\pi_F^G(F)$

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

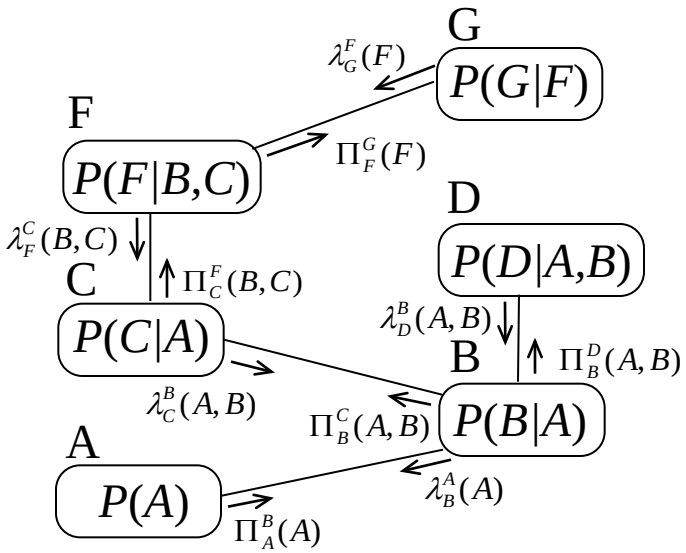
Bucket D: $P(D/A,B)$ → $\pi_B^D(A,B)$

Bucket C: $P(C/A)$ $\lambda_F^C(B,C)$ → $\pi_B^C(A,B)$

Bucket B: $P(B/A)$ $\lambda_D^B(A,B)$ $\lambda_C^B(A,B)$ → $\pi_A^B(A)$

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

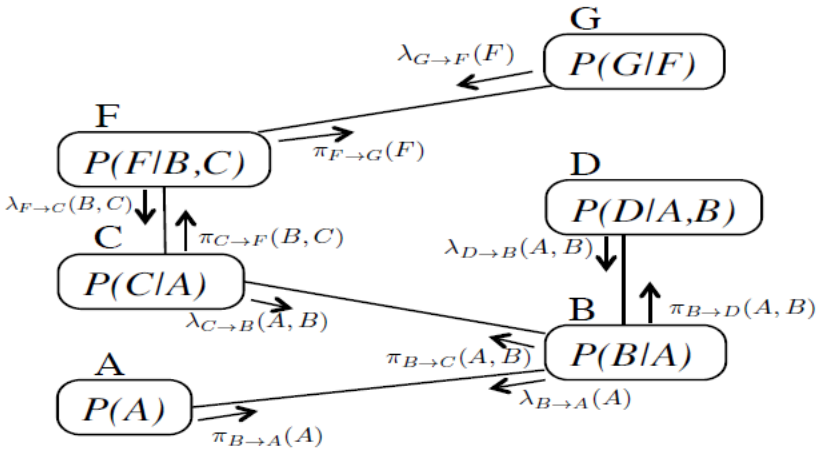
Each bucket can Compute its marginal probability



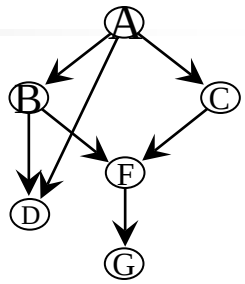
$$\pi_A^B(a) = P(a)$$
$$\pi_B^C(c,a) = P(b|a)\lambda_D^B(a,b)\pi_A^B(a)$$
$$\pi_B^D(a,b) = P(b|a)\lambda_C^B(a,b)\pi_A^B(a,b)$$
$$\pi_C^F(c,b) = \sum_a P(c|a)\pi_B^C(a,b)$$
$$\pi_F^G(f) = \sum_{b,c} P(f|b,c)\pi_C^F(c,b)$$

BTE: Allows messages both ways

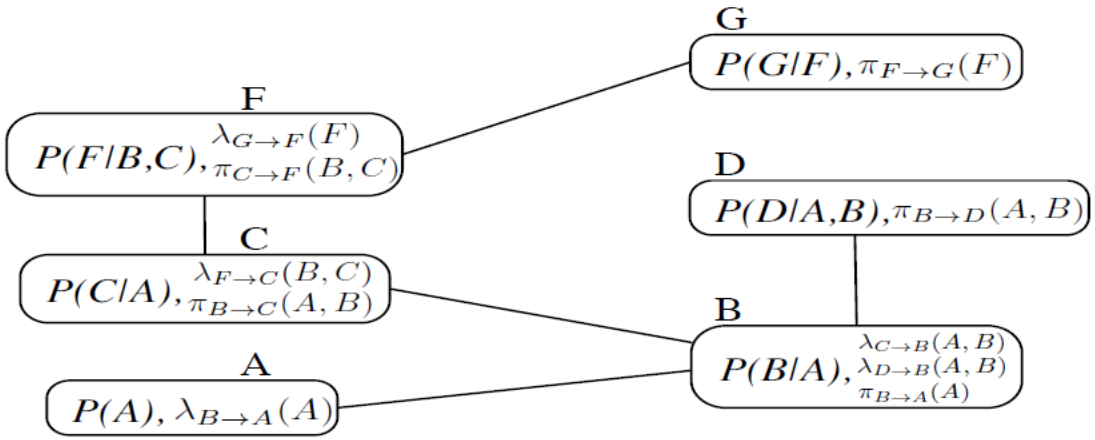
Initial buckets
+ messages



(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:

$$\mathcal{M}_Y = \sum \prod f, \tag{6.4}$$

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*

$$\prod_{f \in F_Y} f = \mathcal{M}_Y \tag{6.5}$$

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 imapness of trees)*

•Theorem 6.1.5 (Complexity of BTE) *Let w*(d) 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* $k^{(w*(d)+1)}$*), where deg is the maximum degree in the bucket-tree. The space complexity of BTE is O(n* $k^{w^*(d)}$*.)*

Asynchronous BTE: Bucket-tree Propagation (BTP)

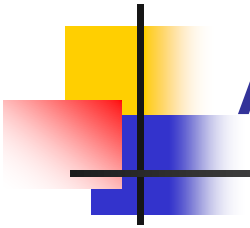
BUCKET-TREE PROPAGATION (BTP)

Input: A problem $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, ordering d. $X = \{X_1, ..., X_n\}$ and $F = \{f_1, ..., f_r\}$, $E = e$. 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
   $\lambda_{i \to j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \to i})$
3. **endfor**

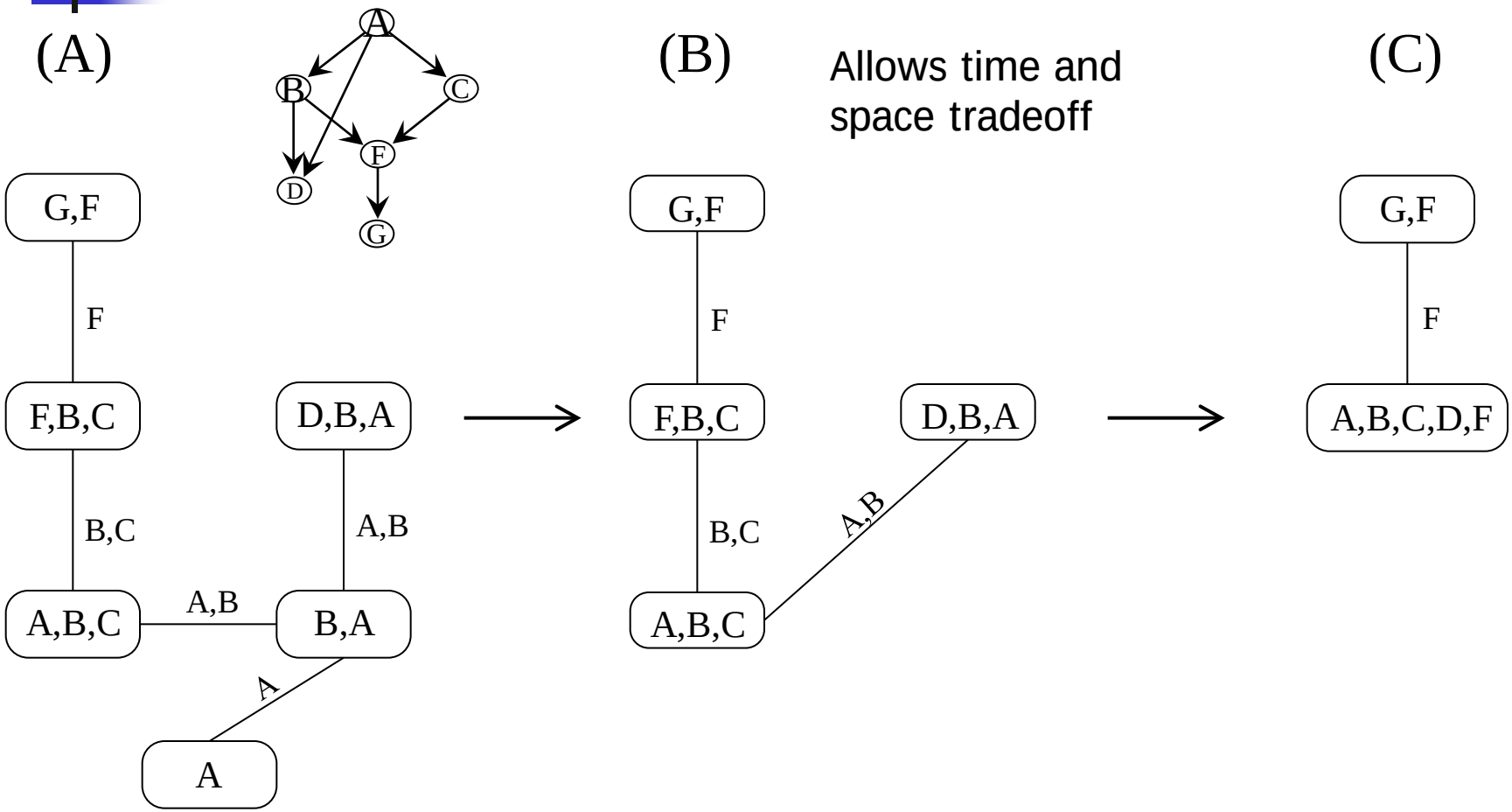
$$\lambda_{i \to j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \to i})$$



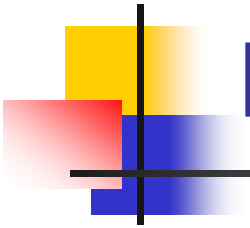
Agenda

- 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.

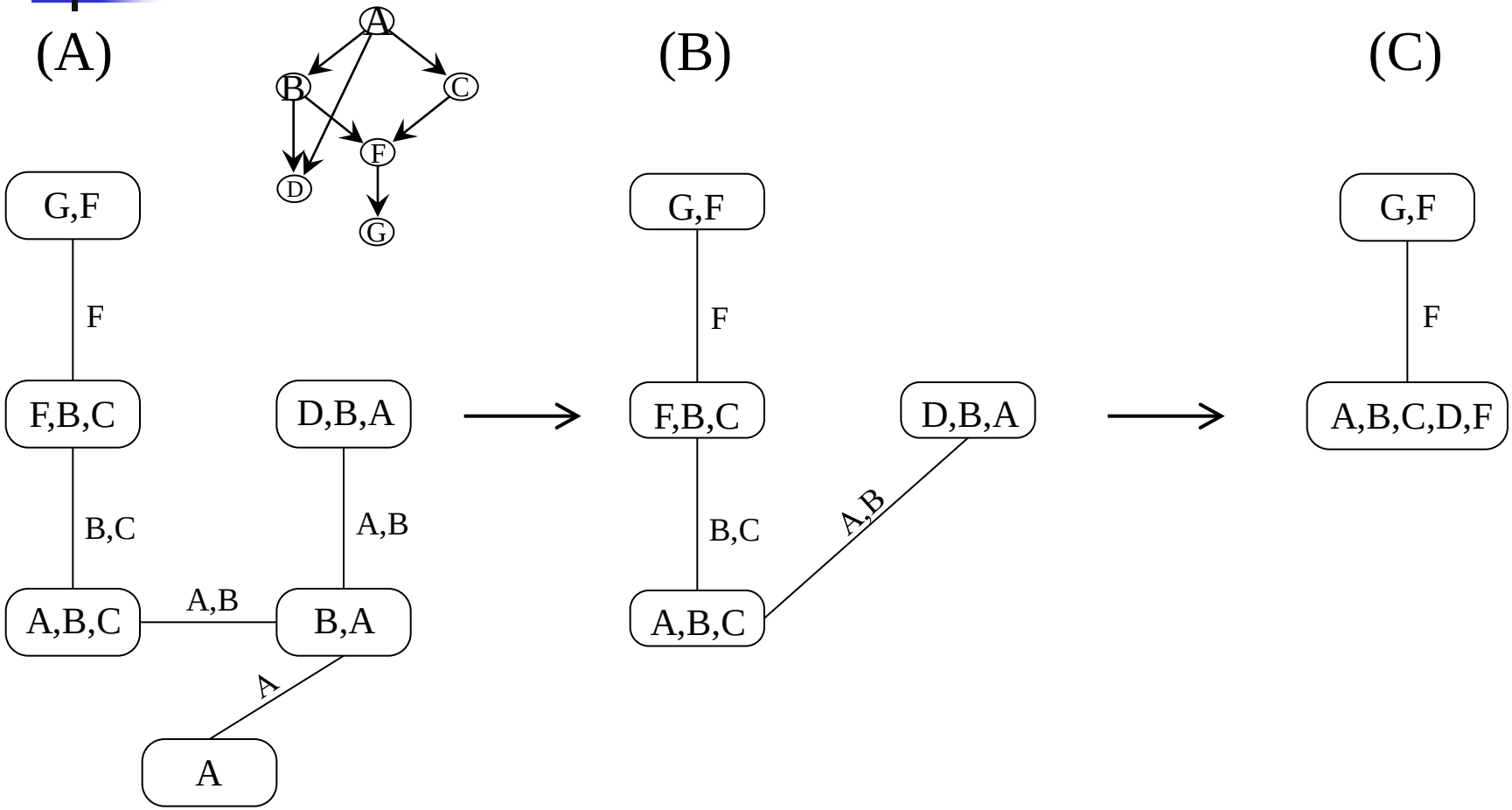
Tree-decompositions

A *tree decomposition* for a belief network $BN =< X,D,G,P >$ 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 :

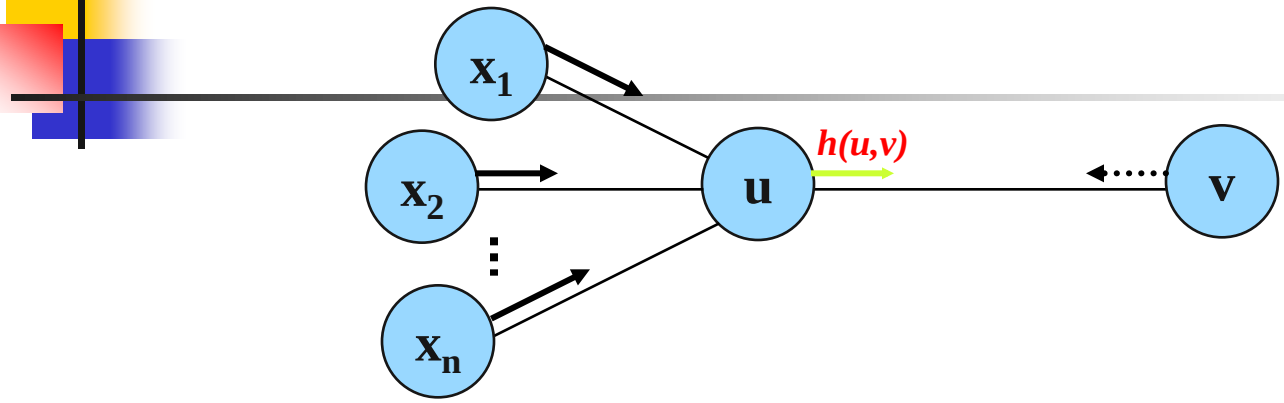
1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $scope(p_i) \subseteq \chi(v)$
2. For each variable $X_i \in X$ the set $\{v \in V | X_i \in \chi(v)\}$ forms a connected subtree (running intersection property)

Definition 6.2.8 (treewidth, separator-width, eliminator) *The* treewidth *[4] of a tree-decomposition* $< T, \chi, \psi >$ *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


(A)
A
B
C
F
D
G
G,F
F
F,B,C
B,C
A,B,C
A,B
D,B,A
A,B
B,A
A
(B)
G,F
F
F,B,C
B,C
A,B,C
A,B
D,B,A
(C)
G,F
F
A,B,C,D,F

The General Message Passing On a General Tree-Decomposition



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

For max-product
Just replace ∑
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 $< T, \chi, \psi >$ for a problem $M =< X, D, F, \prod\} >$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$. 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 $< T, \chi, \bar{\psi} >$ where for $u \in T$, $\chi(u), \bar{\psi}(u)$ is model explicit.

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

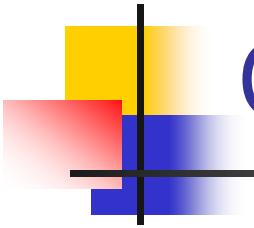
   for every edge (u, v) in the tree, **do**

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

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

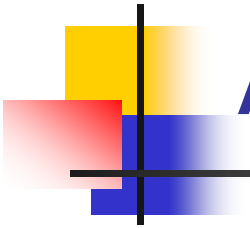
   endfor

   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 $< T, \chi, \bar{\psi} >$, where $\bar{\psi}(v) \Leftarrow \psi(v) \cup_{u \in neighbor(v)} \{m_{u \to v}\}$.



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.*



Agenda

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

Acyclic Networks

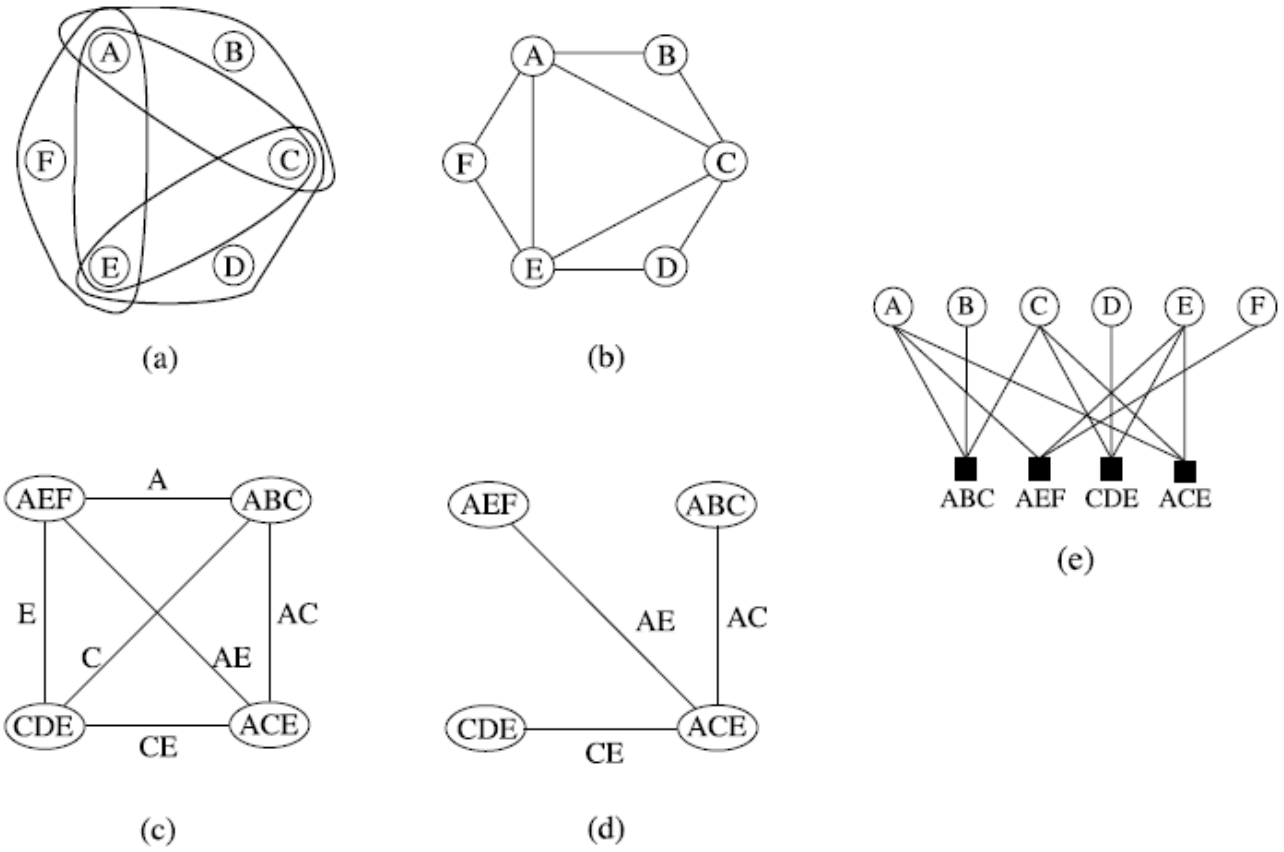


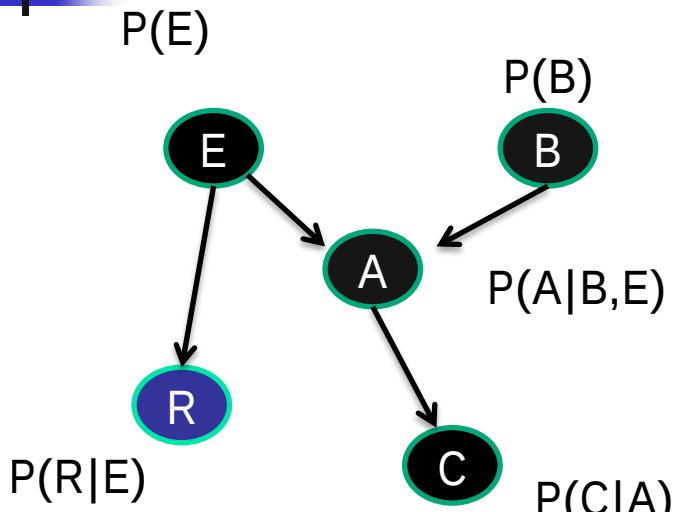
Figure 5.1: (a)Hyper, (b)Primal, (c)Dual and (d)Join-tree of a graphical model having scopes ABC, AEF, CDE and ACE. (e) the factor graph

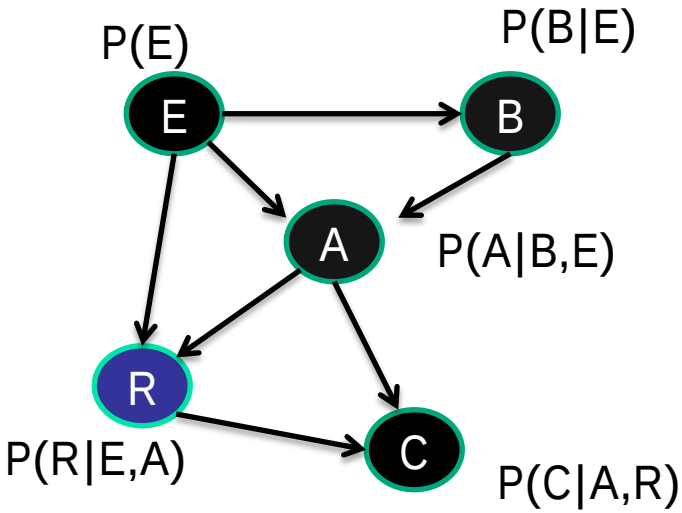
Polytrees and Acyclic Networks

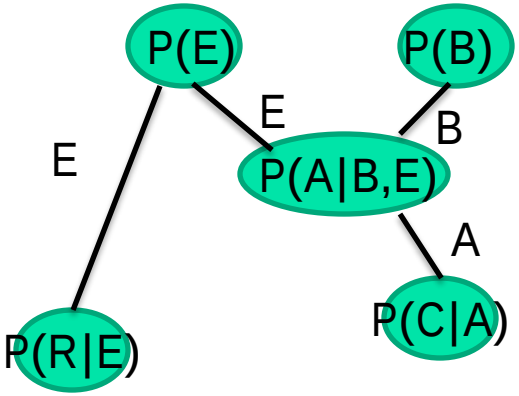
- **Polytree:** a BN whose undirected skeleton is a tree
- **Acyclic network**: A network is acyclic if it has a tree-decomposition where each node has a single original CPT.
- **Dual network**: each scope-cpt is a node and each arc is denoted by intersection.
- **Acylic network (alternative definition):**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

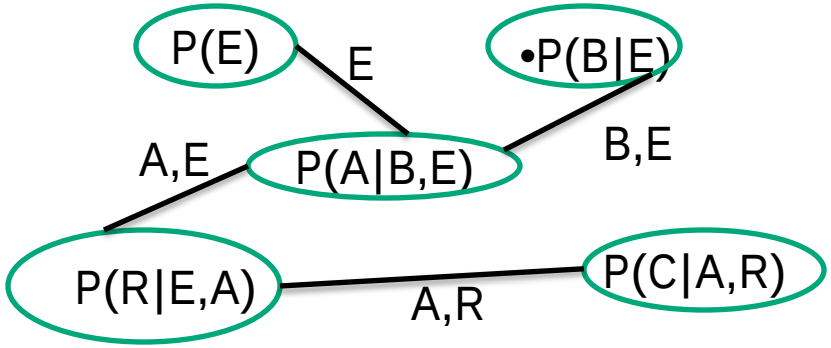
Definition 6.4.4 (belief propagation (BP)) *Given a ploytree and a directed dual graph which is a poly-tree decomposition, The belief propagation algorithm (BP) is algorithm CTE, whose messages down the polytree-decomposition are called λ and up are called π.*

Acyclic-Networks: Belief Propagation is Easy









A Glimpse into Pearl's BP

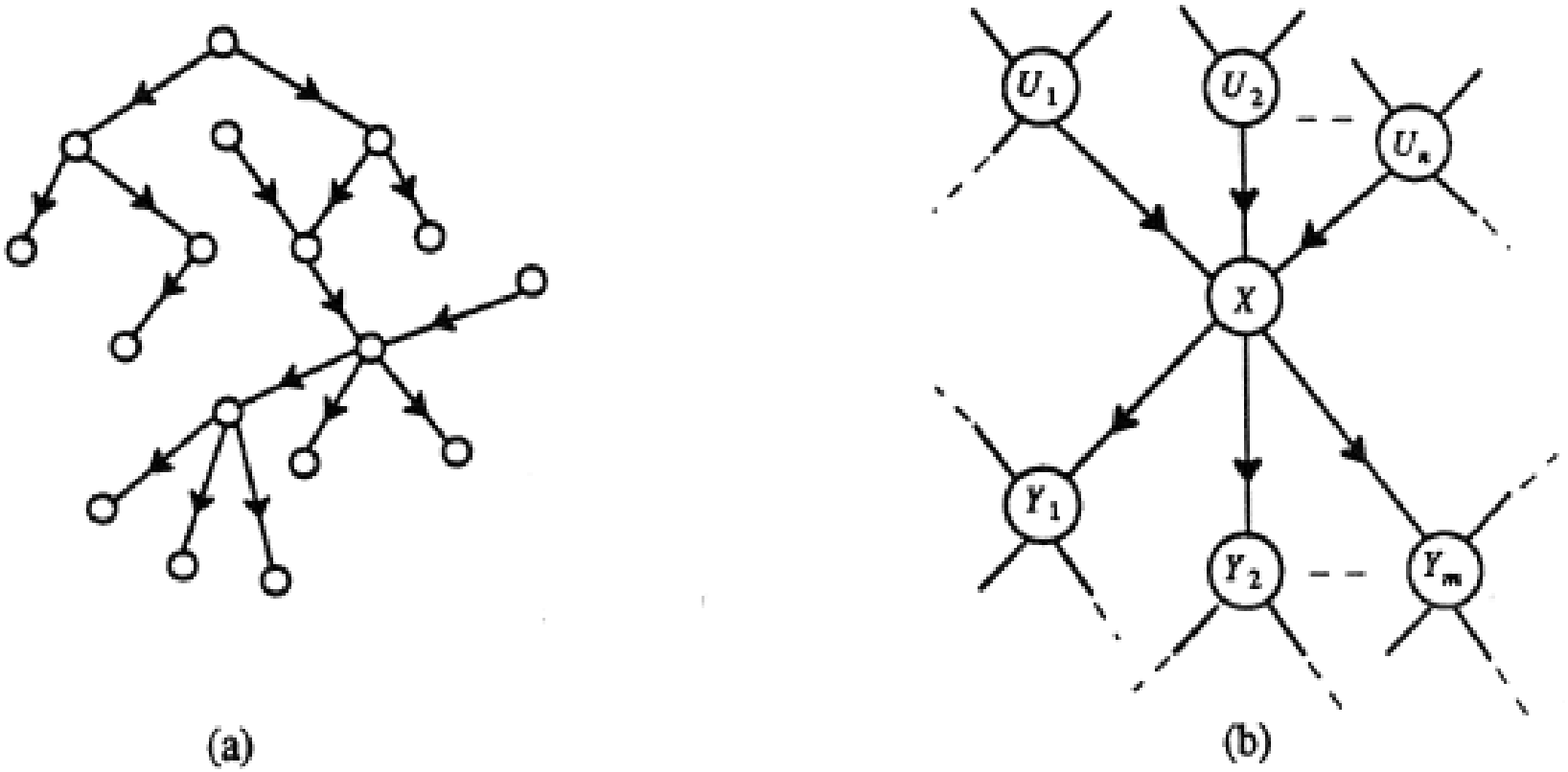


Figure 4.18. (a) A fragment of a polytree and (b) the parents and children of a typical node X.

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.

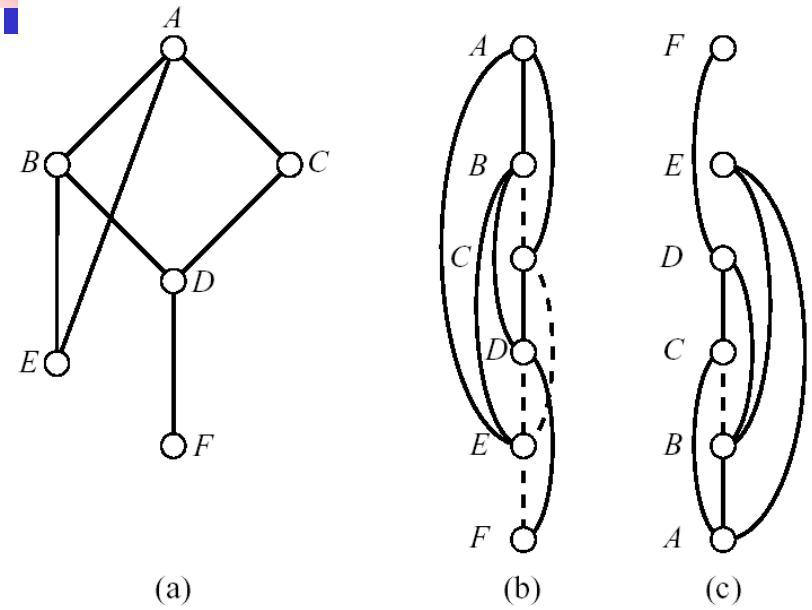
JOIN-TREE CLUSTERING (JTC)

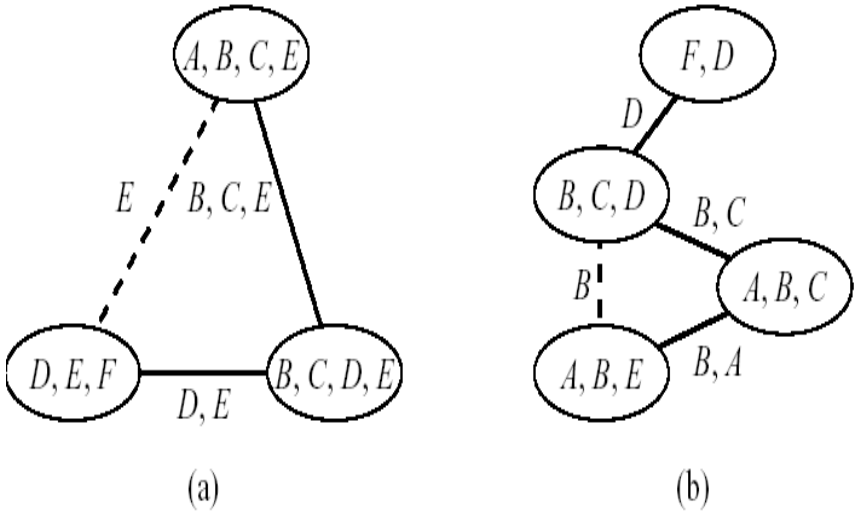
Input: A graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, $\mathbf{X} = \{X_1, ..., X_n\}$, $\mathbf{F} = \{f_1, ..., f_r\}$.
Its scopes $S = S_1, ..., S_r$ and its primal graph is $G = (X, E)$.
Output: A join-tree decomposition $< T, \chi, \psi >$ for $\mathcal{M}$

1. Select a variable ordering, $d = (X_1, ..., X_n)$.
2. **Triangulation** (create the induced graph along d and call it G^*):
   for $j = n$ to 1 by -1 do
   $E \leftarrow E \cup \{(i,k) | i < j,\ k < j,\ (i,j) \in E, (k,j) \in E\}$
3. **Create a join-tree of the induced graph** (G^*, d) as follows:
   a. Identify all maximal cliques in the chordal graph.
   Let $C = \{C_1, ..., C_t\}$ be all such cliques, where C_i is the cluster of bucket i.
   b. Create a tree T of cliques:
   Connect each C_i to a C_j $(j < i)$ with whom it shares largest subset of variables.
4. Create ψ_i: Partition input function in cluster-node whose variables contain its scope.
5. Return a tree-decomposition $< T, \chi, \psi >$, where T is generated in step 3,
$\chi(i) = C_i$ and $\psi(i)$ is determined in step 4.

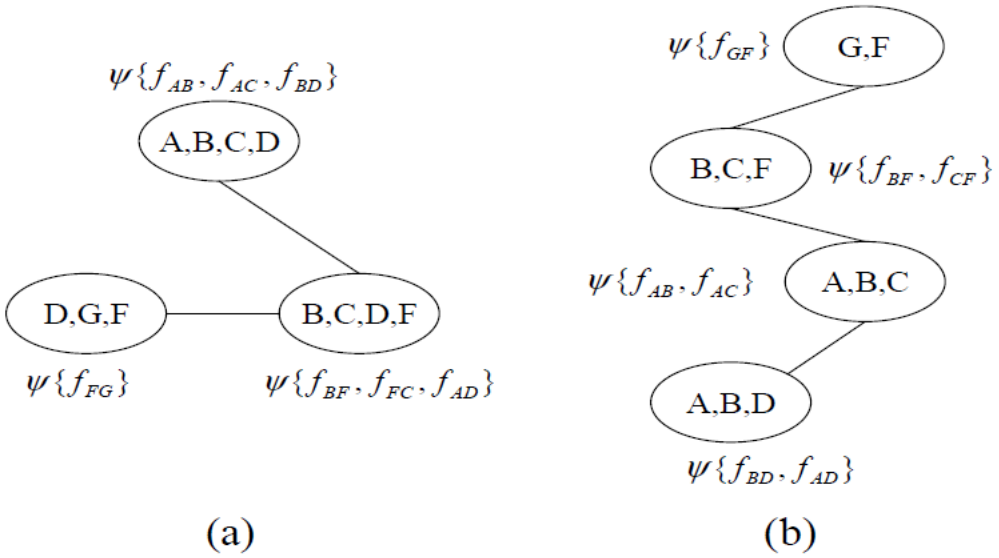
Figure 5.12: Join-tree clustering.

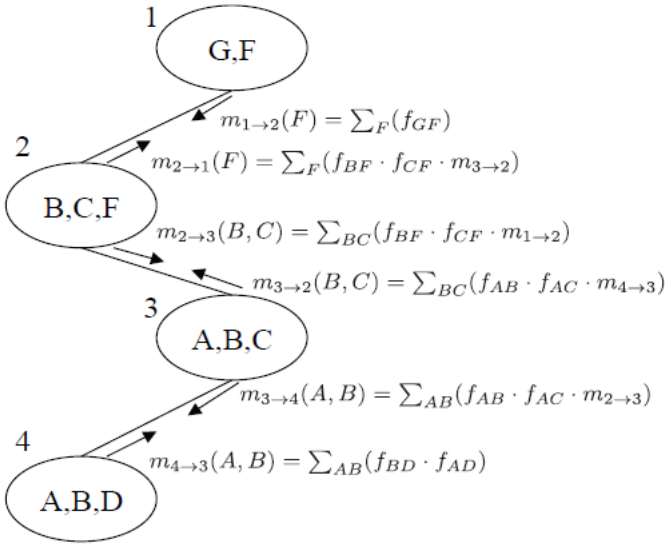
Examples of tree-clustering





Tree-clustering and message-passing





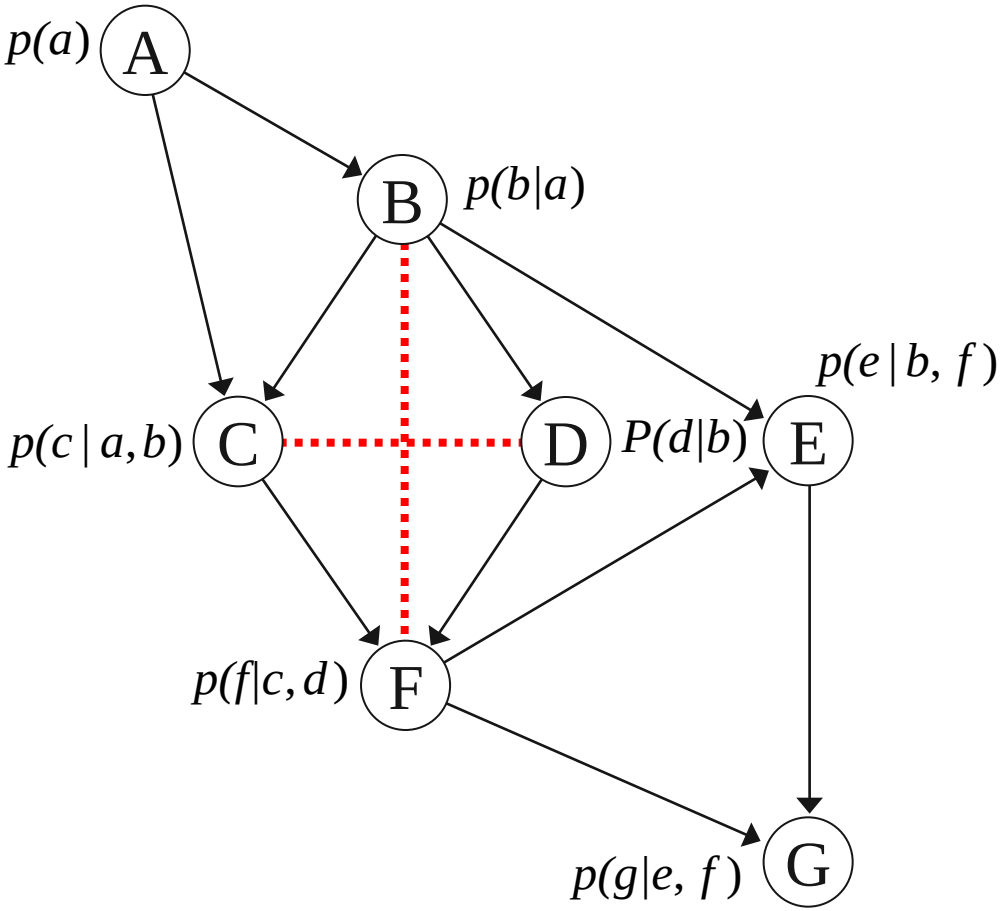
Cluster-Tree Elimination - Properties

- 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: $O (N \times d^{sep})$

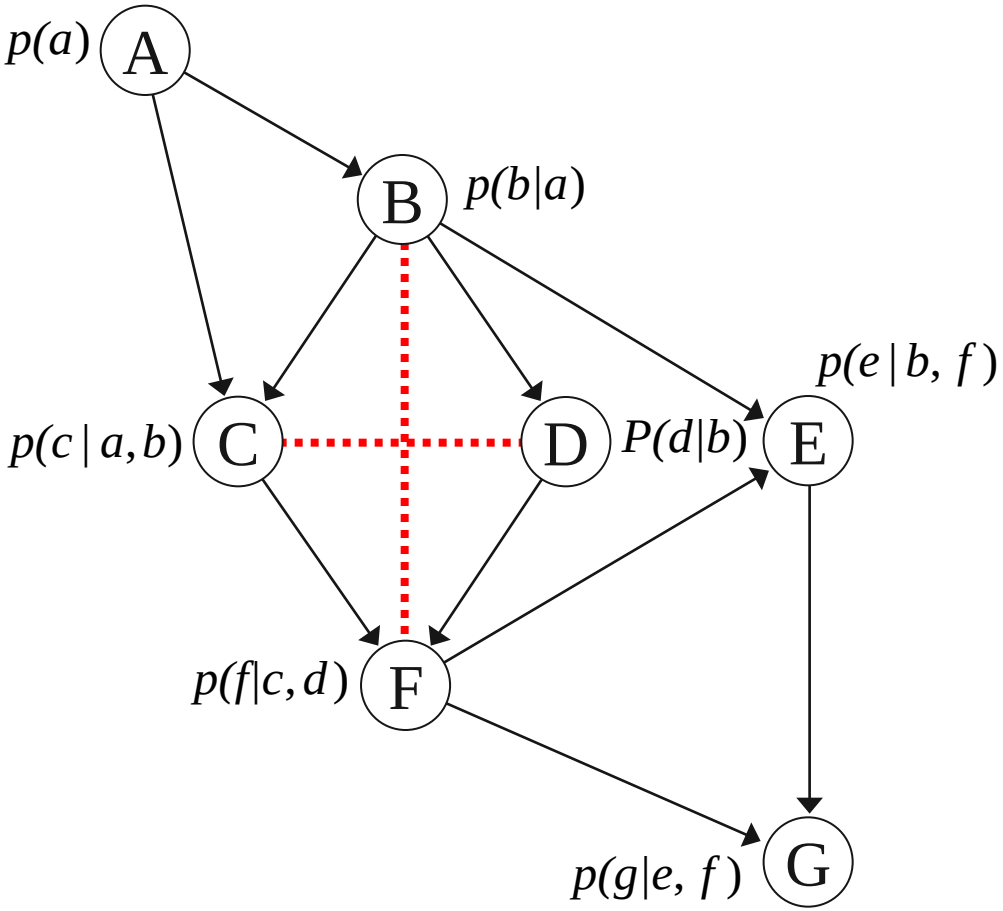
where

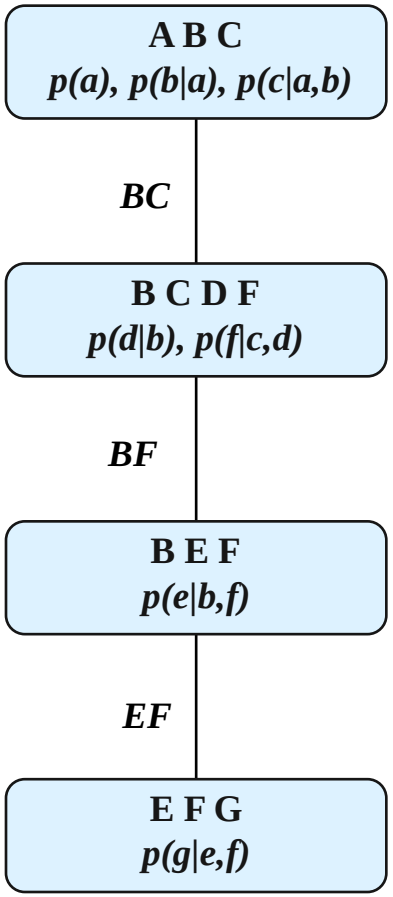
deg = the maximum degree of a node
n = number of variables (= number of CPTs)
N = number of nodes in the tree decomposition
d = the maximum domain size of a variable
*w** = the induced width
sep = the separator size

Tree Decomposition for belief updating



Tree Decomposition for belief updating





Example

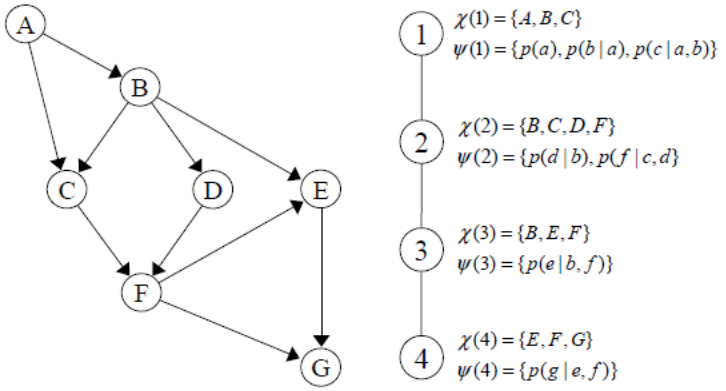


Figure 6.12: [Execution of CTE-BU]: *a*) A belief network; *b*) A join-tree decomposition; *c*)Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

Example

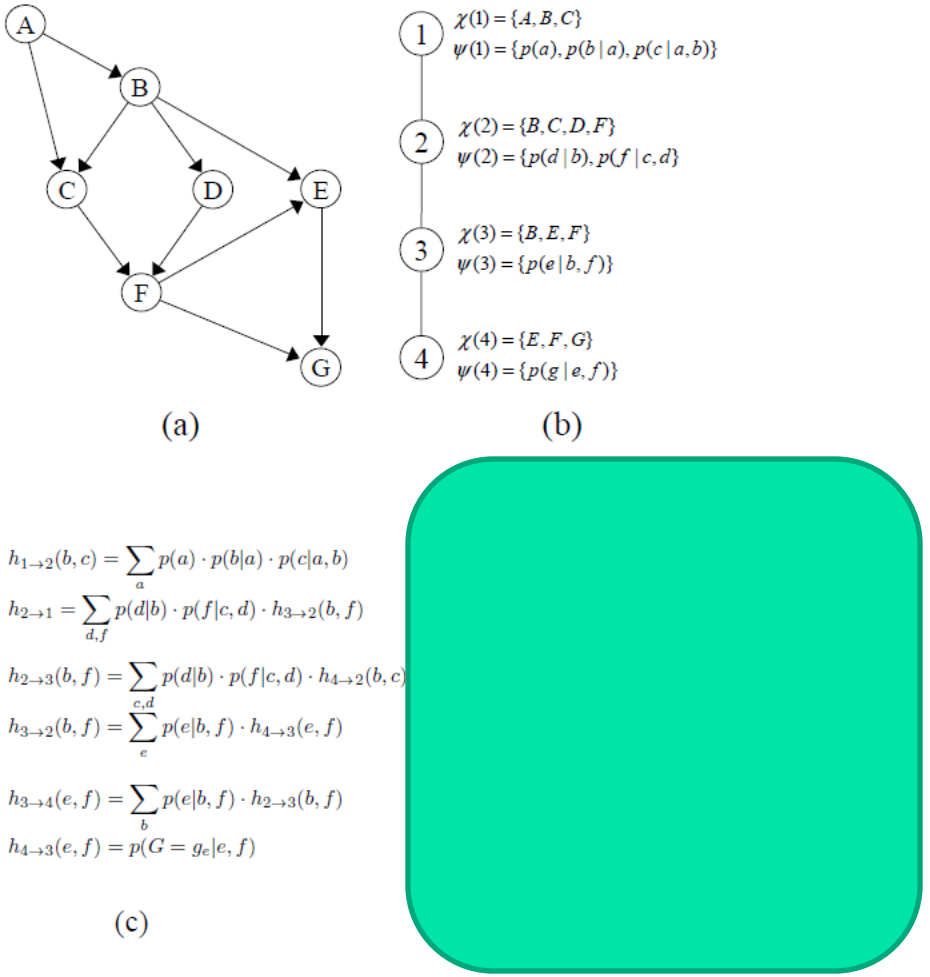


Figure 6.12: [Execution of CTE-BU]: *a*) A belief network; *b*) A join-tree decomposition; *c*)Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

Example

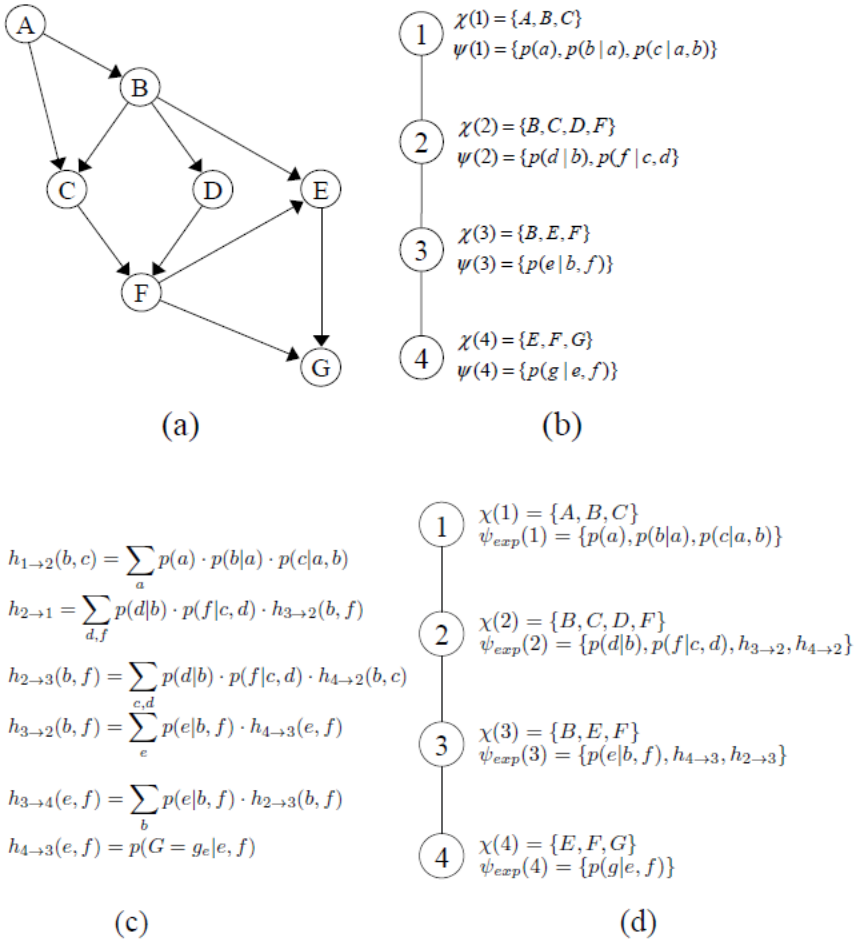
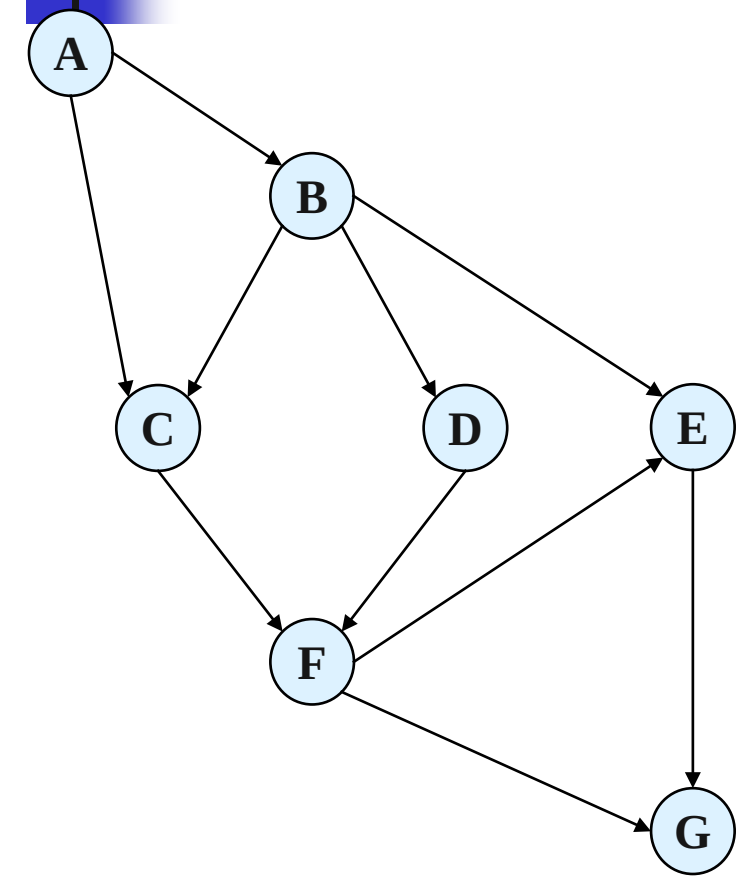
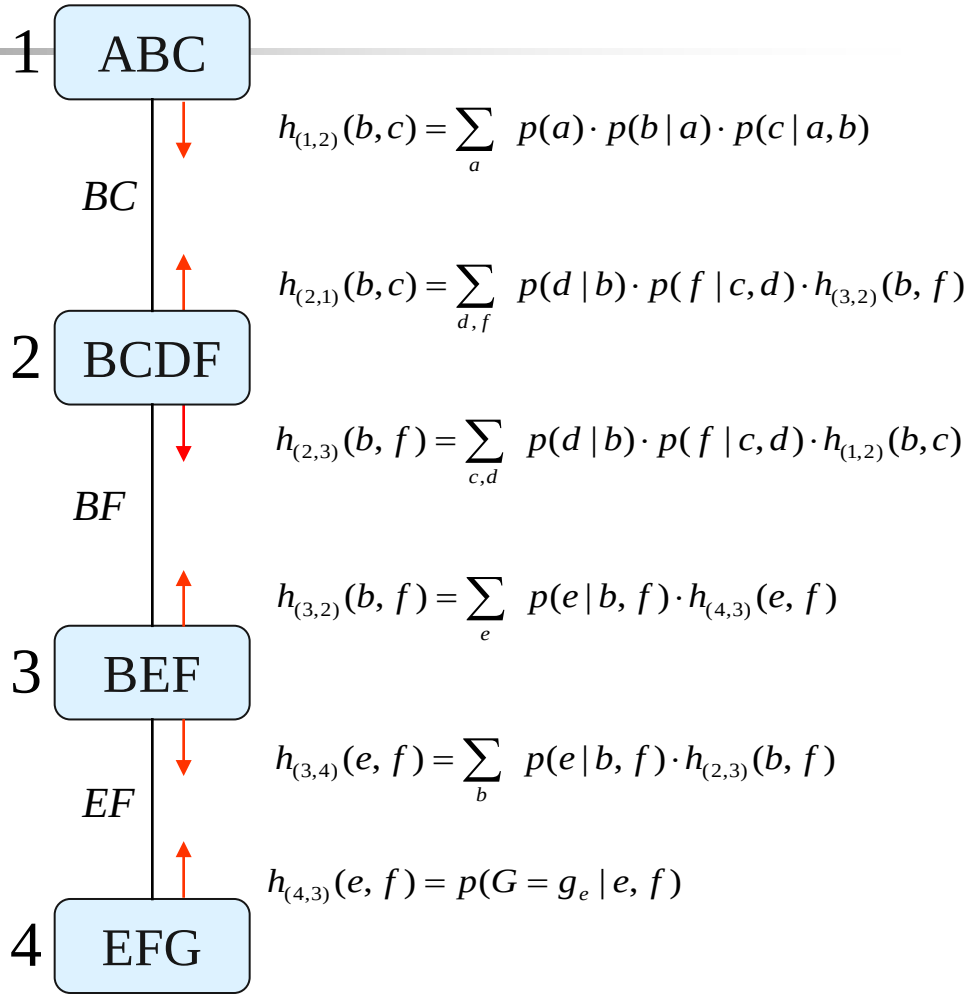


Figure 6.12: [Execution of CTE-BU]: *a*) A belief network; *b*) A join-tree decomposition; *c*)Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition

CTE: Cluster Tree Elimination





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

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

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

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

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

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

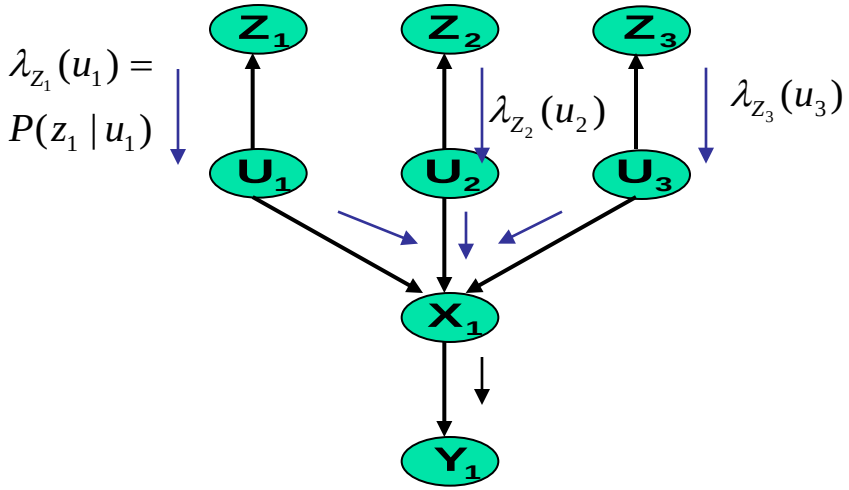
Time: *O (exp(w+1))*

Space: *O (exp(sep))*

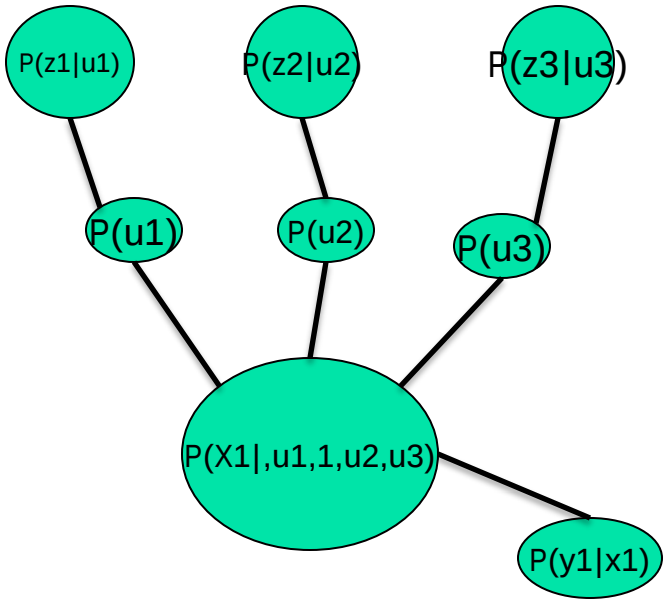
For each cluster P(X|e) is computed, also P(e)

Belief Propagation is Easy on Polytrees: Pearl's Belief Propagation

A polytree: a tree with Larger families



A polytree decomposition



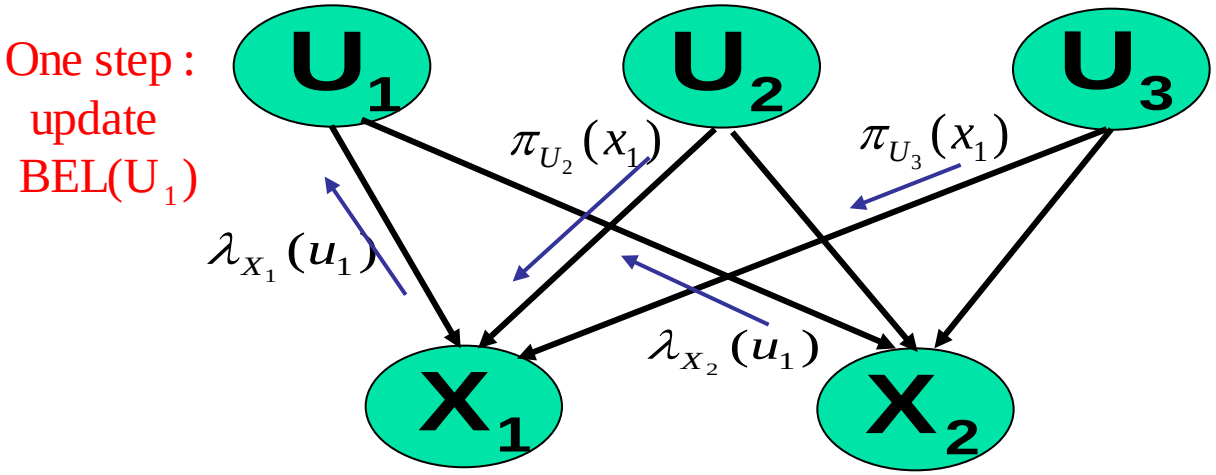
Running CTE = running Pearl's BP over the dual graph

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

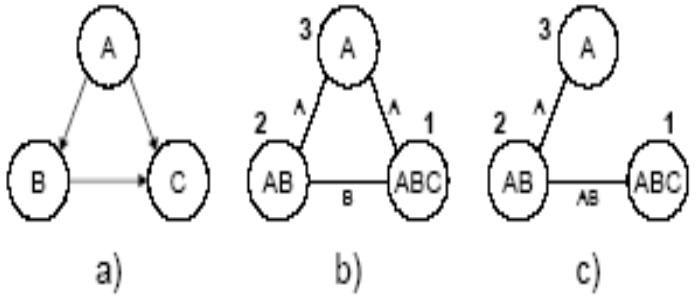


Figure 6.13: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

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:**

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

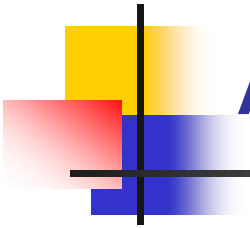
  Endfor

- **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,

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

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



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 (Dechter, 7.1, 7.2)

The idea of cutset-conditioning

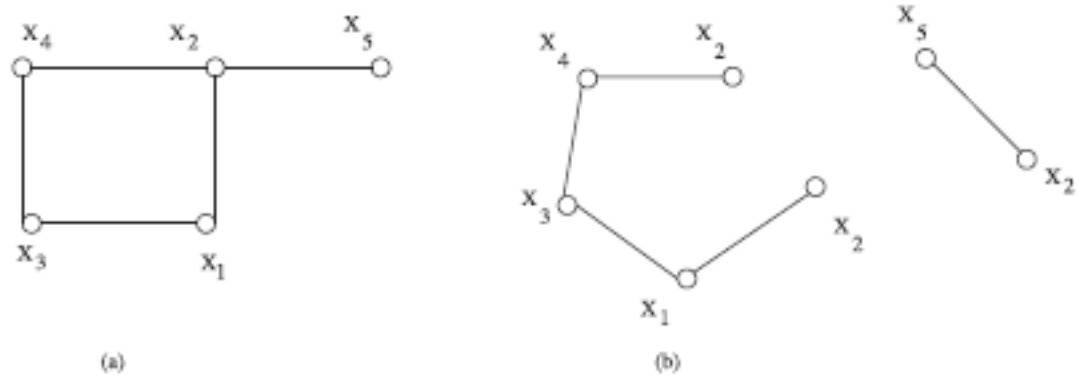
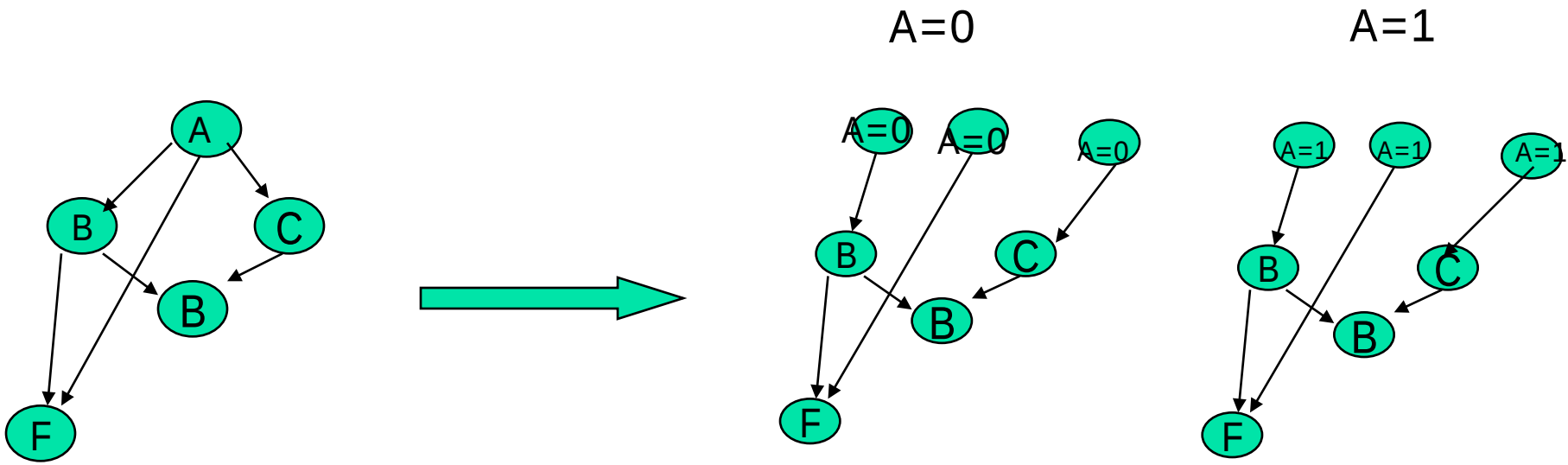


Figure 7.1: An instantiated variable cuts its own cycles.

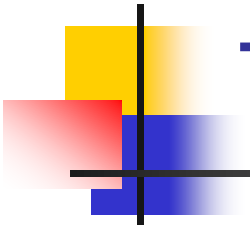
Loop-cutset decomposition

- 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



The Idea of Cutset-Conditioning

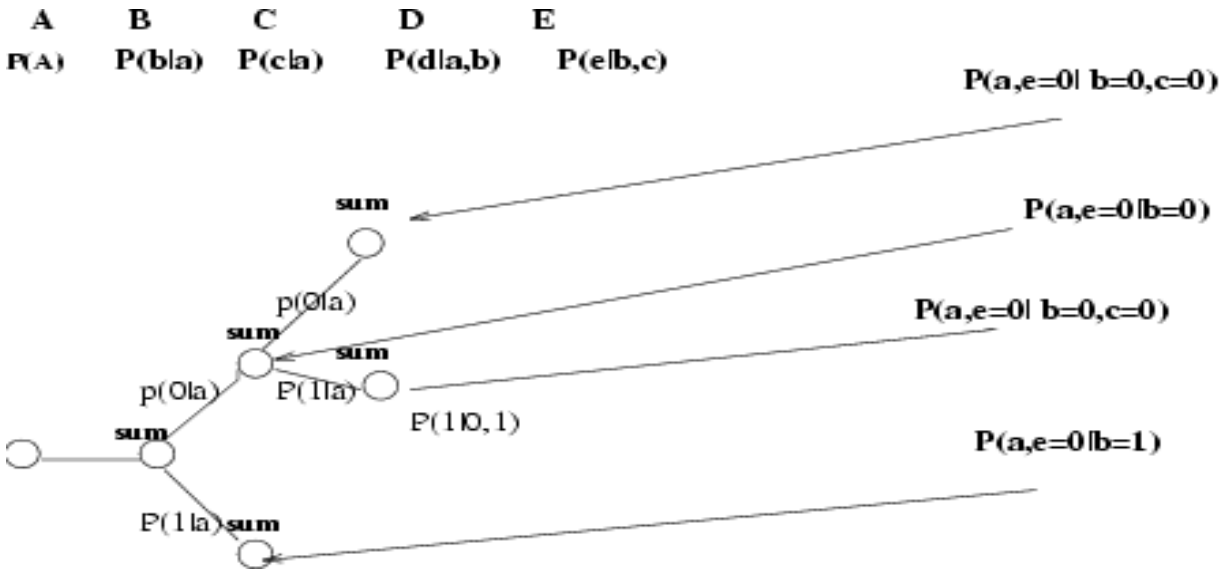
We observed that when variables are assigned connectivity reduces.
The magnitude of saving is reflected through the “conditioned-induced graph”

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

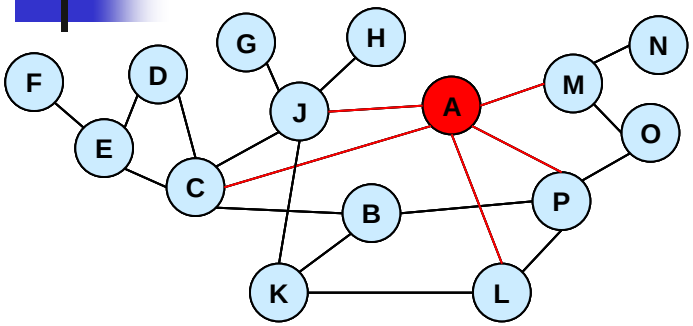
Conditioning+Elimination

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



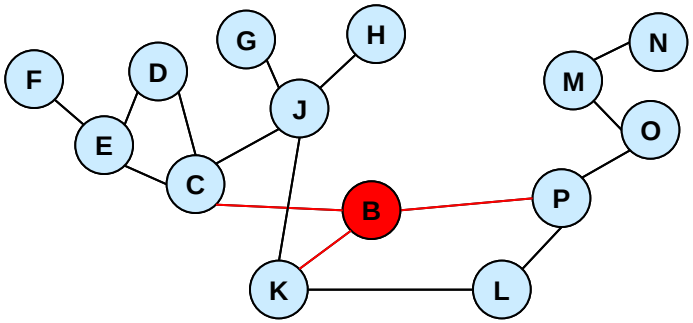
Idea: conditioning until w^* of a (sub)problem gets small

Conditioning and Cycle cutset

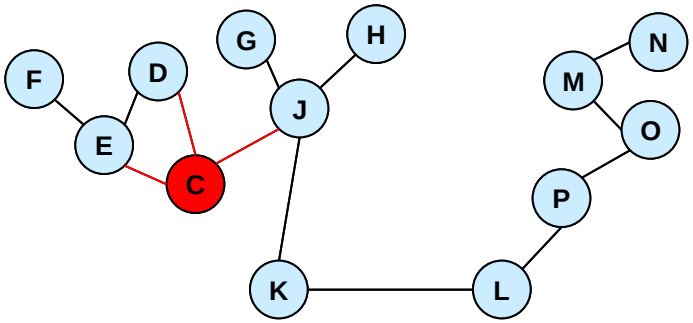


Cycle cutset = {A,B,C}

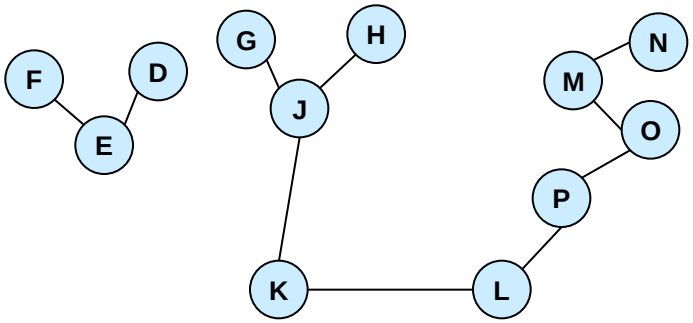
A



B



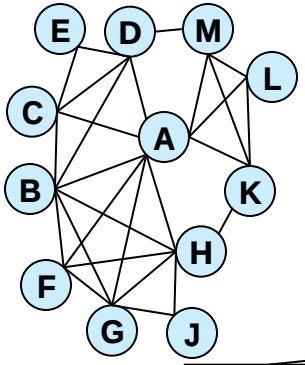
C



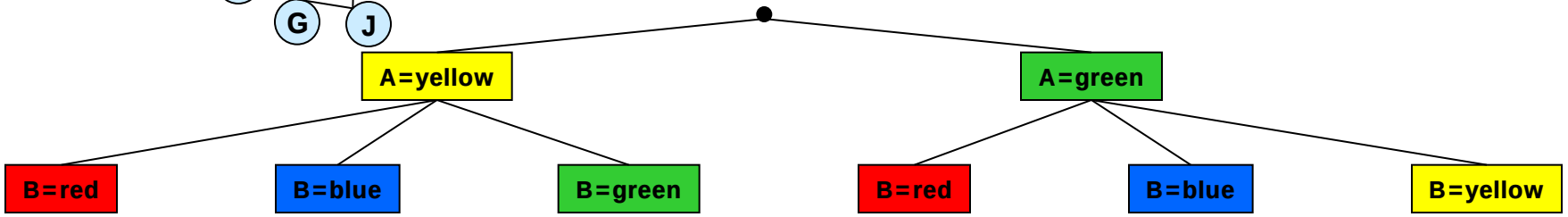
1-cutset = {A,B,C}, size 3

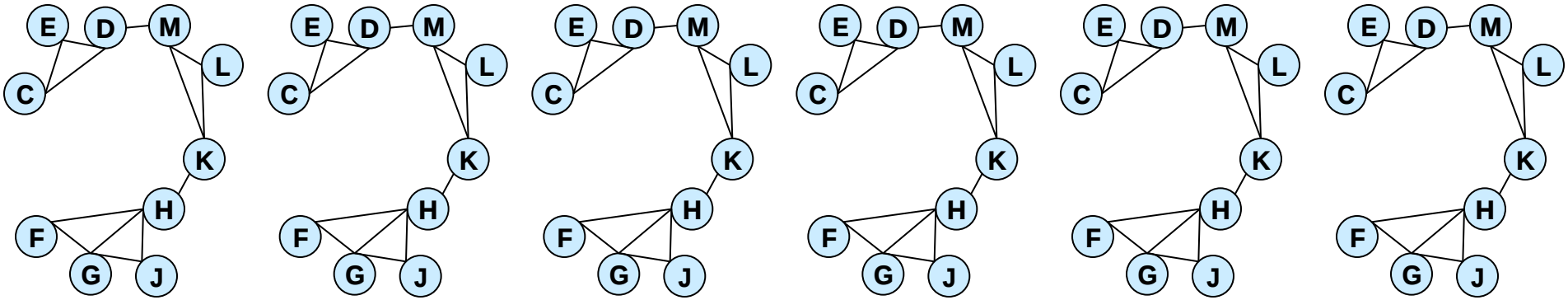
Search over the Cutset (cont)

Graph Coloring problem



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





2-cutset = {A,B}, size =2

q-cutset, minimal

Definition 7.3 ***q*-cutset, minimal.** Given a graph G, a subset of nodes is called a *q-cutset* for an integer q iff when removed, the resulting graph has an induced-width less than or equal to q. A minimal *q-cutset* of a graph has a smallest size among all q-cutsets of the graph. A cycle-cutset is a 1-cutset of a graph.

Finding a minimal q-cutset is clearly a hard task [A. Becker and Geiger, 1999; Bar-Yehuda *et al.*, 1998; Becker *et al.*, 2000; Bidyuk and Dechter, 2004]. However, like in the special case of a cycle-cutset we can settle for a non-minimal q-cutset relative to a given variable ordering. Namely,

Example 7.4 Consider as another example the contsaint graph of a graph coloring problem given in Figure 7.3a. The search space over a 2-cutset, and the induced-graph of the conditioned instances are depicted in 7.3b.

VEC: Variable Elimination with Conditioning; or, q-cutset algorithms

- VEC-bel:
- Identify a q-cutset, C, of the network
- For each assignment to C=c solve by CTE or BE the conditioned sub-problem.
- Accumulate probability.
- Time complexity: exp(|C|+q)
- Space complexity: exp(q)

Algorithm VEC (Variable-elimination with conditioning)

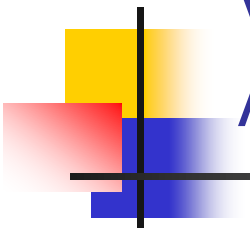
ALGORITHM VEC-EVIDENCE

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

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

Initialize: $\lambda = 0$.

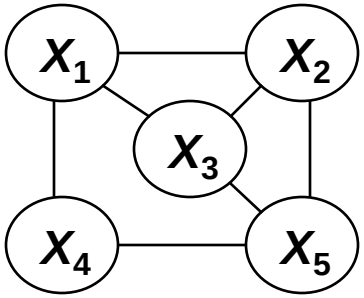
1. For every assignment $C = c$, do
   - $\lambda_1 \leftarrow$ The output of BE-bel with $c \cup e$ as observations.
   - $\lambda \leftarrow \lambda + \lambda_1$. (update the sum).
2. **Return** $P(e) = \alpha \cdot \lambda$ (α is a normalization constant.)



VEC and ALT-VEC: Alternate conditioning and Elimination

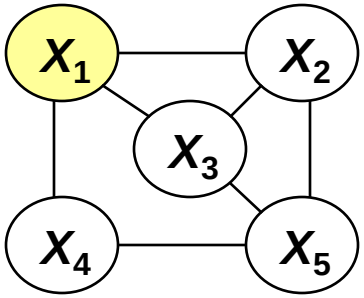
- VEC (q-cutset-conditioining) can also alternate search and elimination, yielding ALT-VEC.
- A time-space tradeoff

Search Basic Step: Conditioning

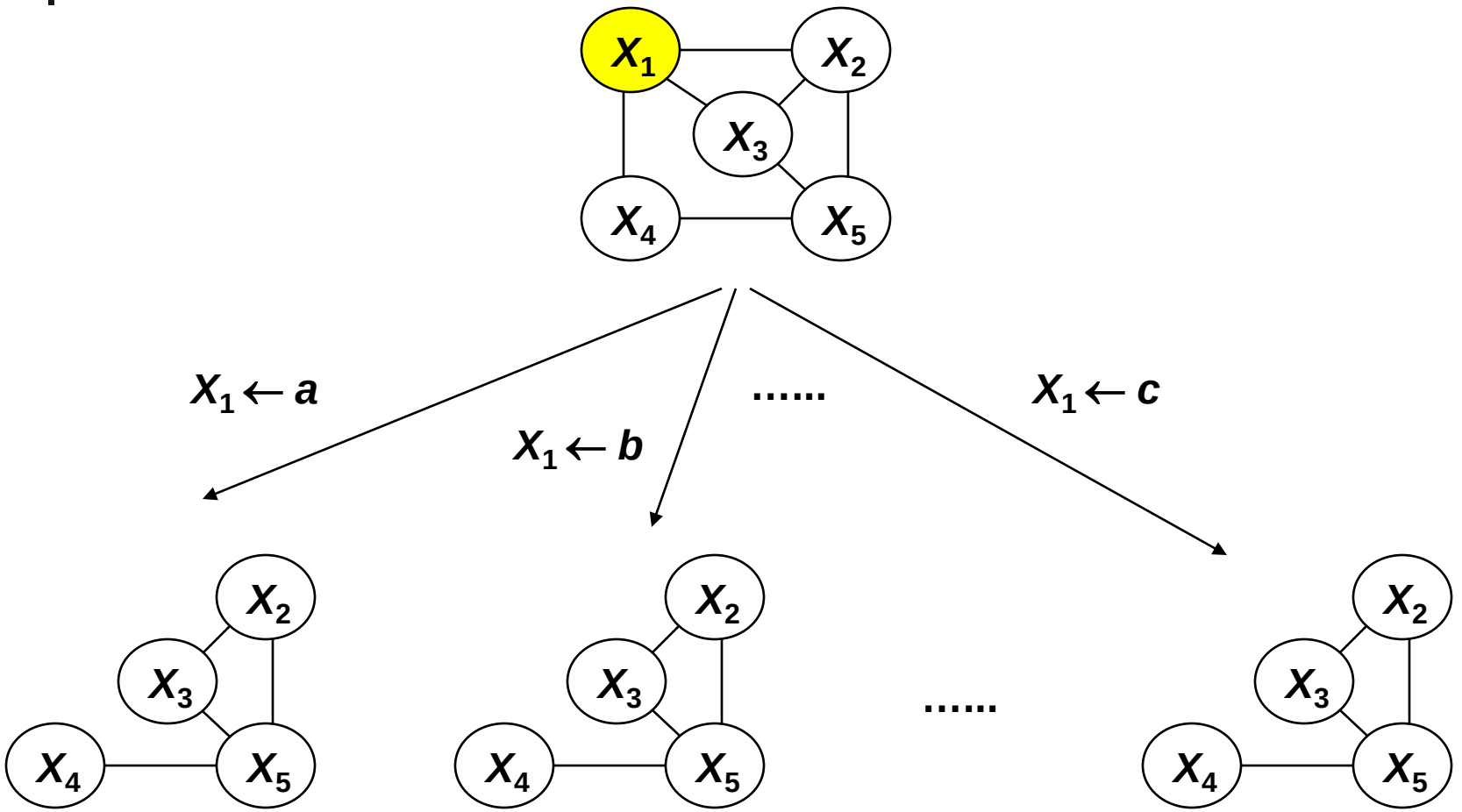


Search Basic Step: Conditioning

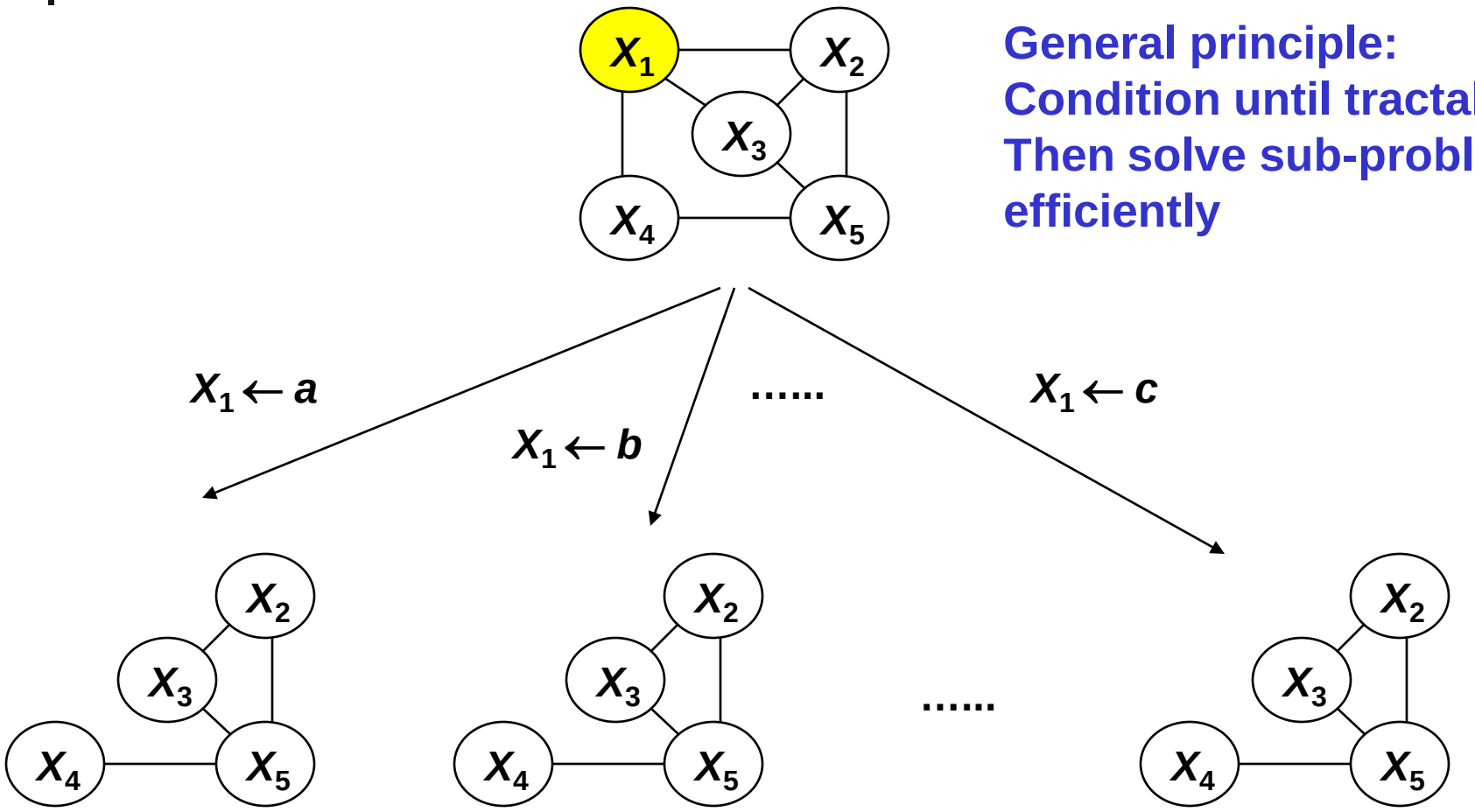
- **Select a variable**



Search Basic Step: Conditioning

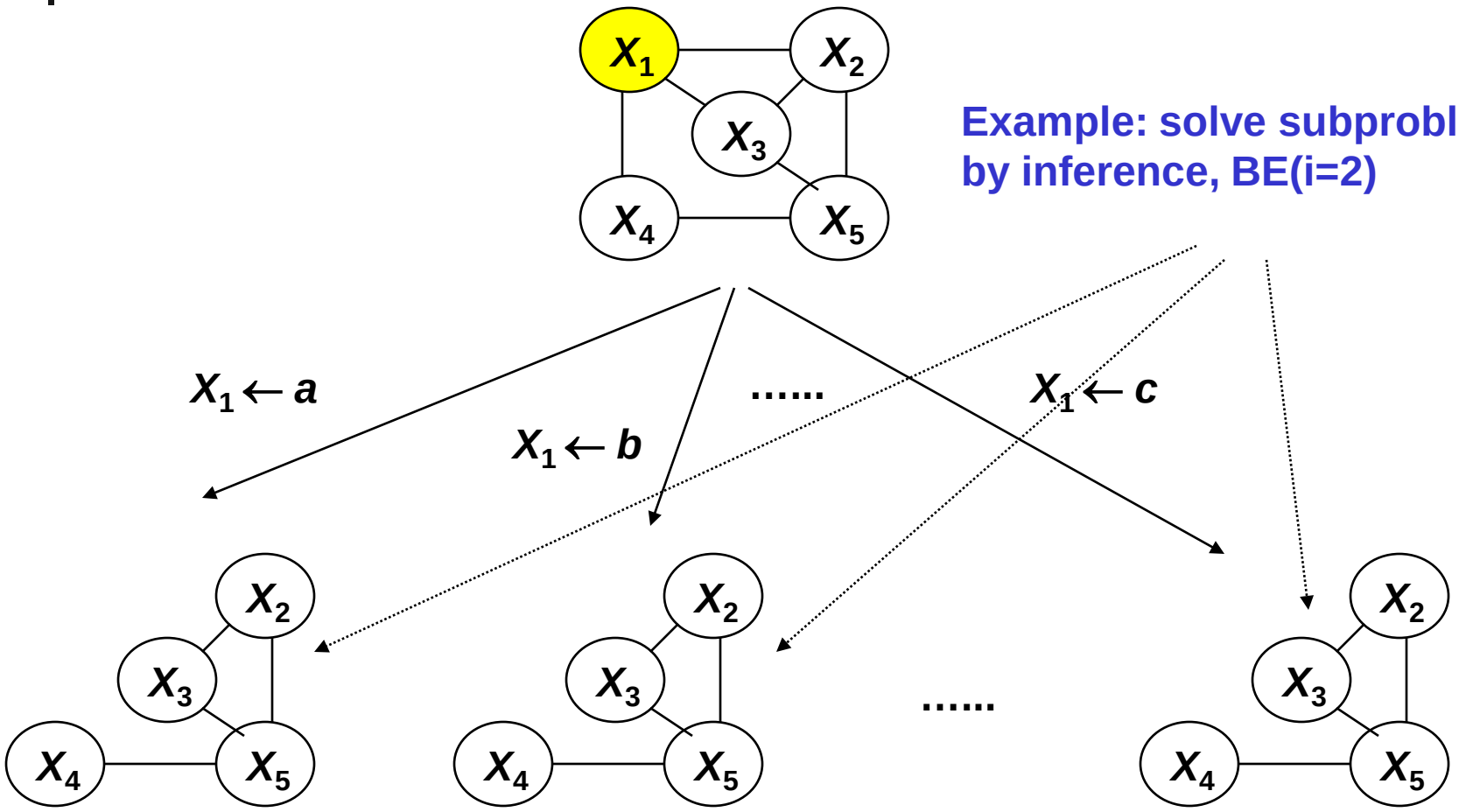


Search Basic Step:
Variable Branching by Conditioning



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

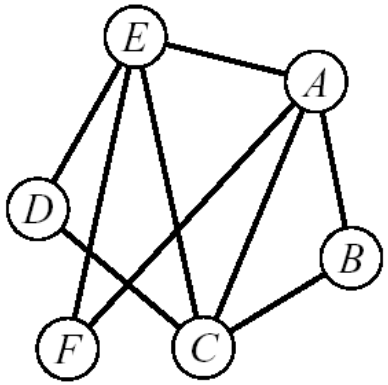
Search Basic Step:
Variable Branching by Conditioning



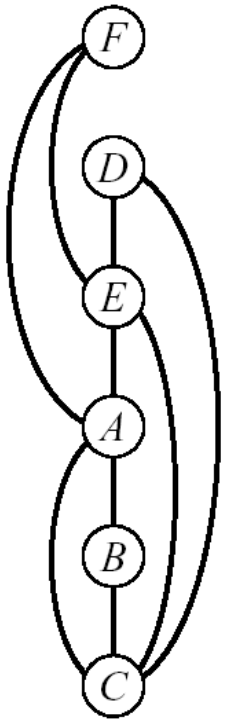
Example: solve subproblem by inference, BE(i=2)

The Cycle-Cutset Scheme:
Condition Until Treeness

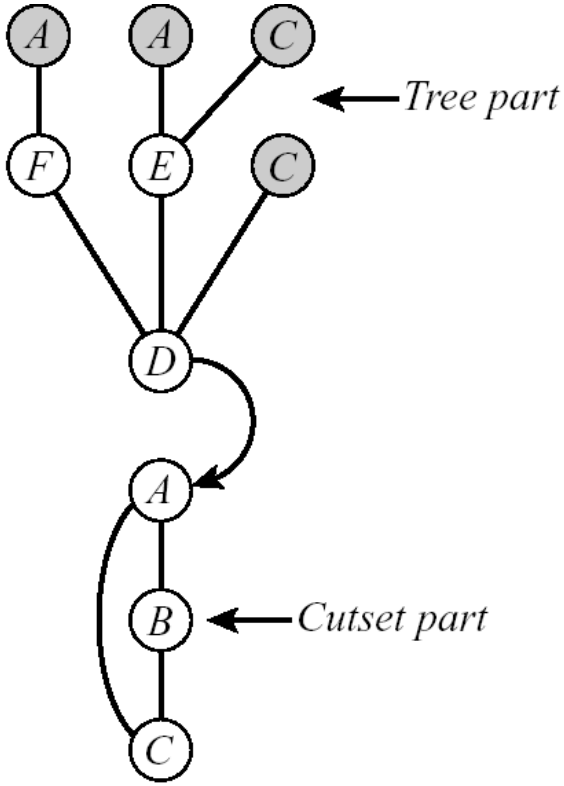
- **Cycle-cutset**
- **i-cutset**
- **C(i)-size of i-cutset**



(a)



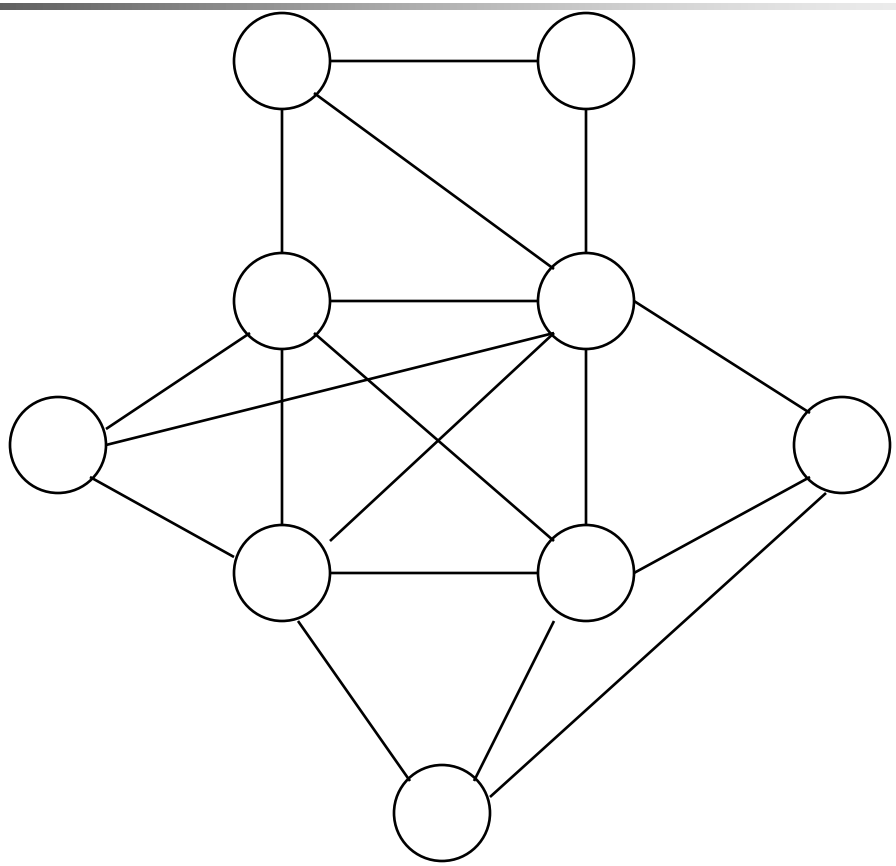
(b)



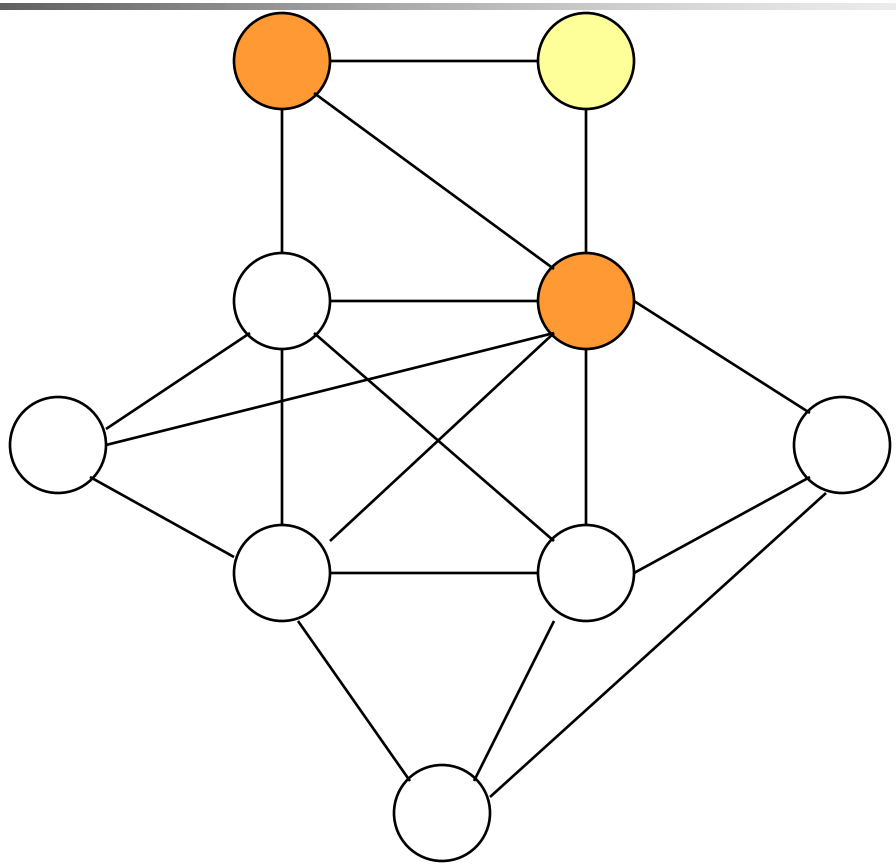
(c)

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

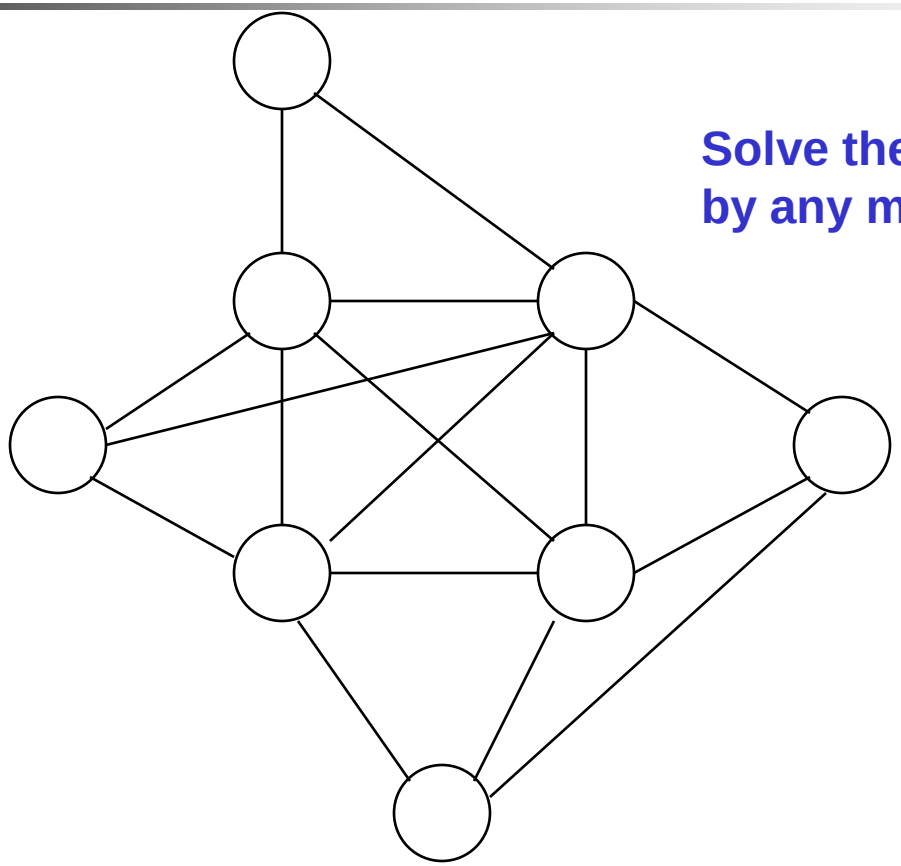
Eliminate First



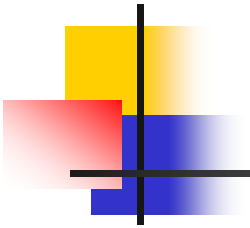
Eliminate First



Eliminate First



Solve the rest of the problem by any means

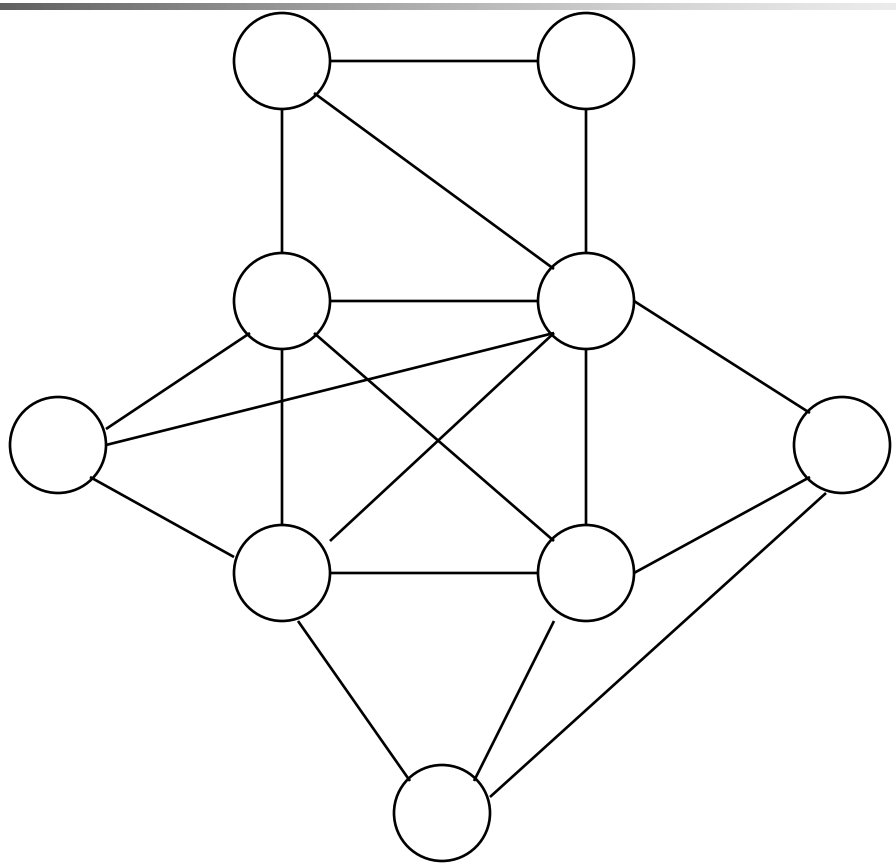


Hybrids Variants

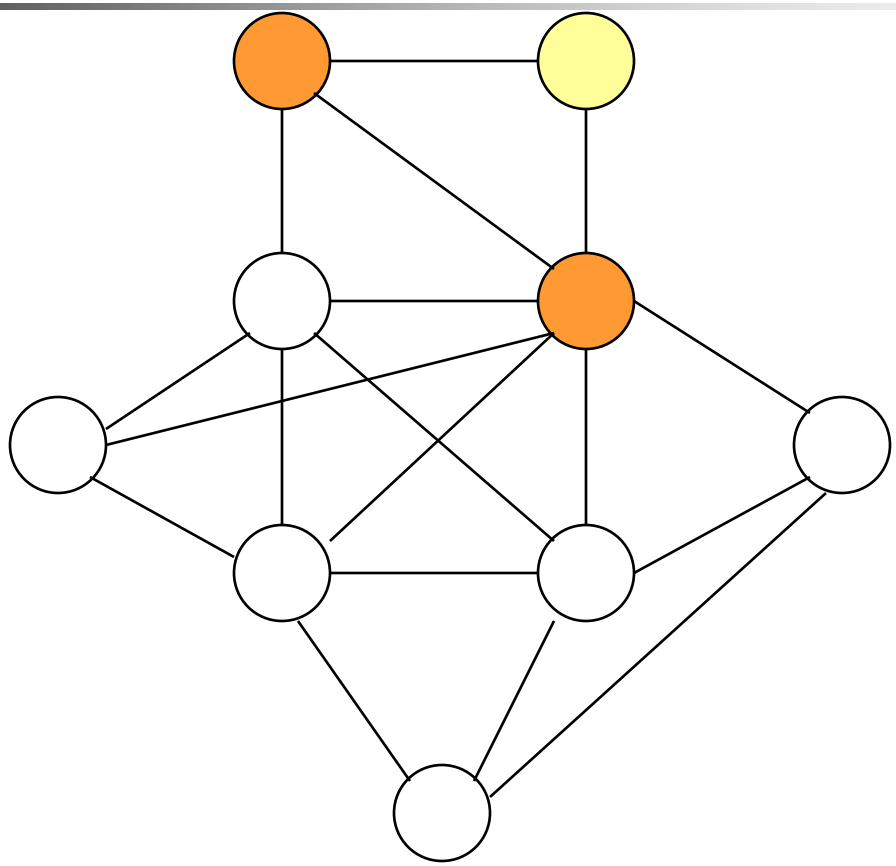
- Condition, condition, condition ... and then only eliminate (w-cutset, cycle-cutset)
- Eliminate, eliminate, eliminate ... and then only search
- Interleave conditioning and elimination (elim-cond(i), VE+C)

Interleaving Conditioning and Elimination

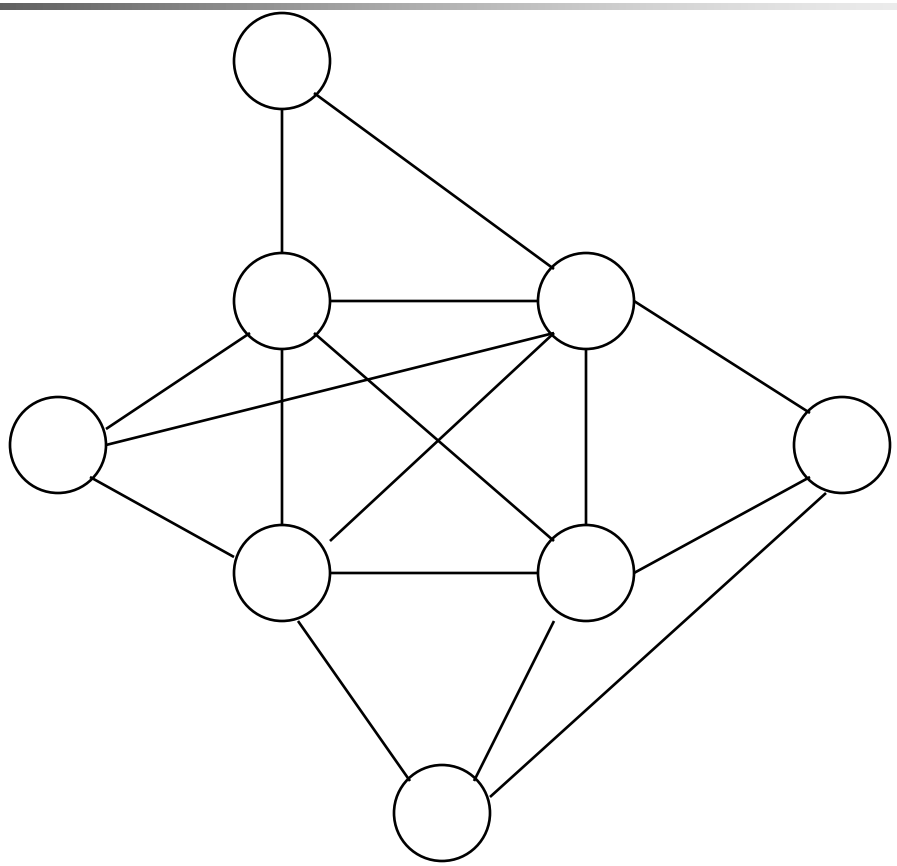
(Larrosa & Dechter, CP'02)



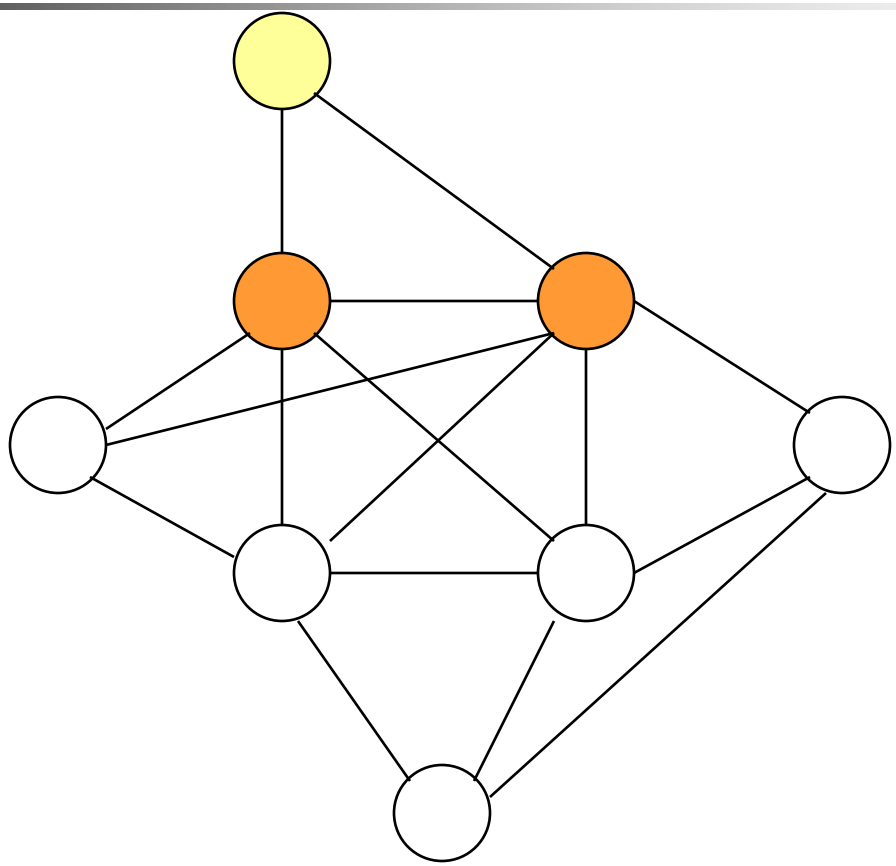
Interleaving Conditioning and Elimination



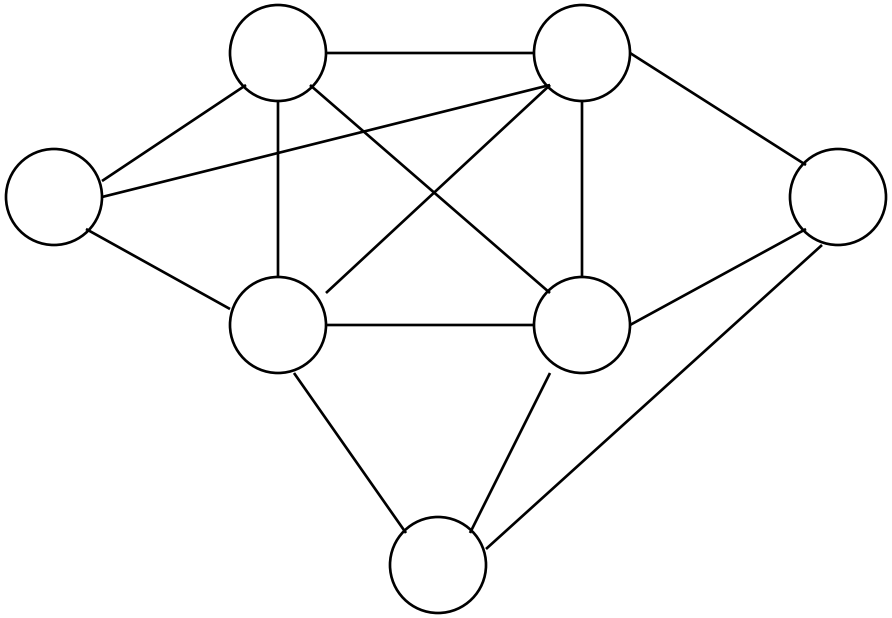
Interleaving Conditioning and Elimination



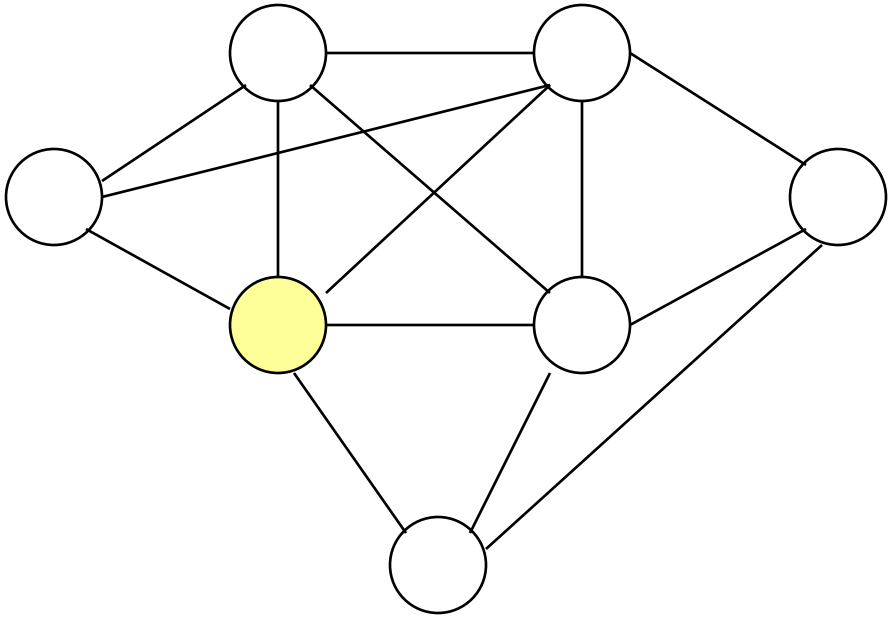
Interleaving Conditioning and Elimination



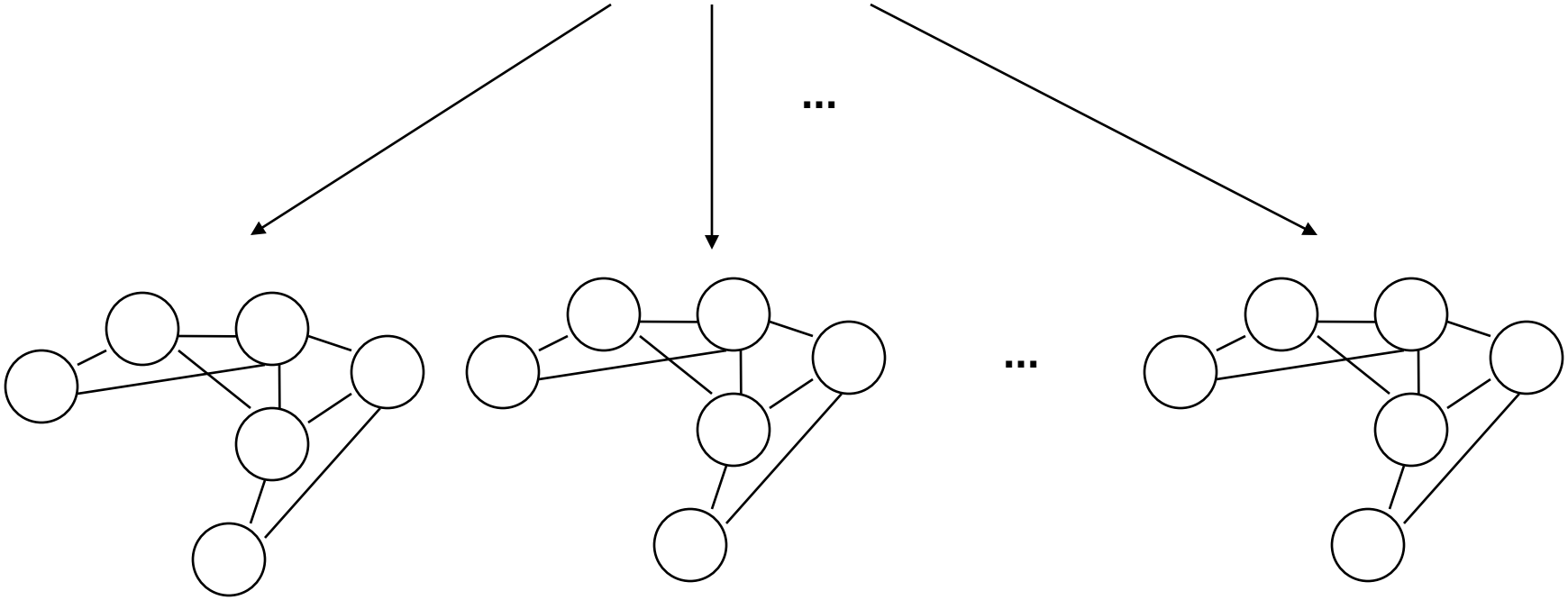
Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



What hybrid should we use?

- q=1? (loop-cutset?)
- q=0? (Full search?)
- q=w* (Full inference)?
- q 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 w* and q-cutstes

Theorem 7.7 *Given graph G, and denoting by c_q^* its minimal q-cutset then,*

$$1 + c_1^* \geq 2 + c_2^* \geq ...q + c_q^*, ... \geq w^* + c_{w^*}^* = w^*.$$

Proof. Let's assume that we have a q-cutset of size c_q. Then if we remove it from the graph the result is a graph having a tree decomposition whose treewidth is bounded by q. Let's T be this decomposition where each cluter has size $q+1$ or less. If we now take the q-cutset variables and add them back to every cluster of T, we will get a tree decomposition of the whole graph (exercise: show that) whose treewidth is $c_q + q$. Therefore, we showed that for *every* c_q-size q-cutset, there is a tree decomposition whose treewidth is $c_q + q$. In particular, for an optimal q-cutset of size c_q^* we have that $w*$, the treewidth obeys, $w* \leq c_q^* + q$. This does not complete the proof because we only showed that for every q, $w* \leq c_q^* + q$. But, if we remove even a single node from a minimal q-cutset whose size is c_q^*, we get a $q+1$ cutset by definition, whose size is $c_q^* - 1$. Therefore, $c_{q+1}^* \leq c_q^* - 1$. Adding q to both sides of the last inequality we get that for every $1 \leq q \leq w^*$, $q + c_q^* \geq q + 1 + c_{q+1}^*$, which completes the proof. □