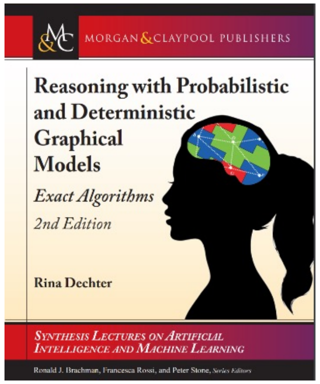
Slides Set 6:

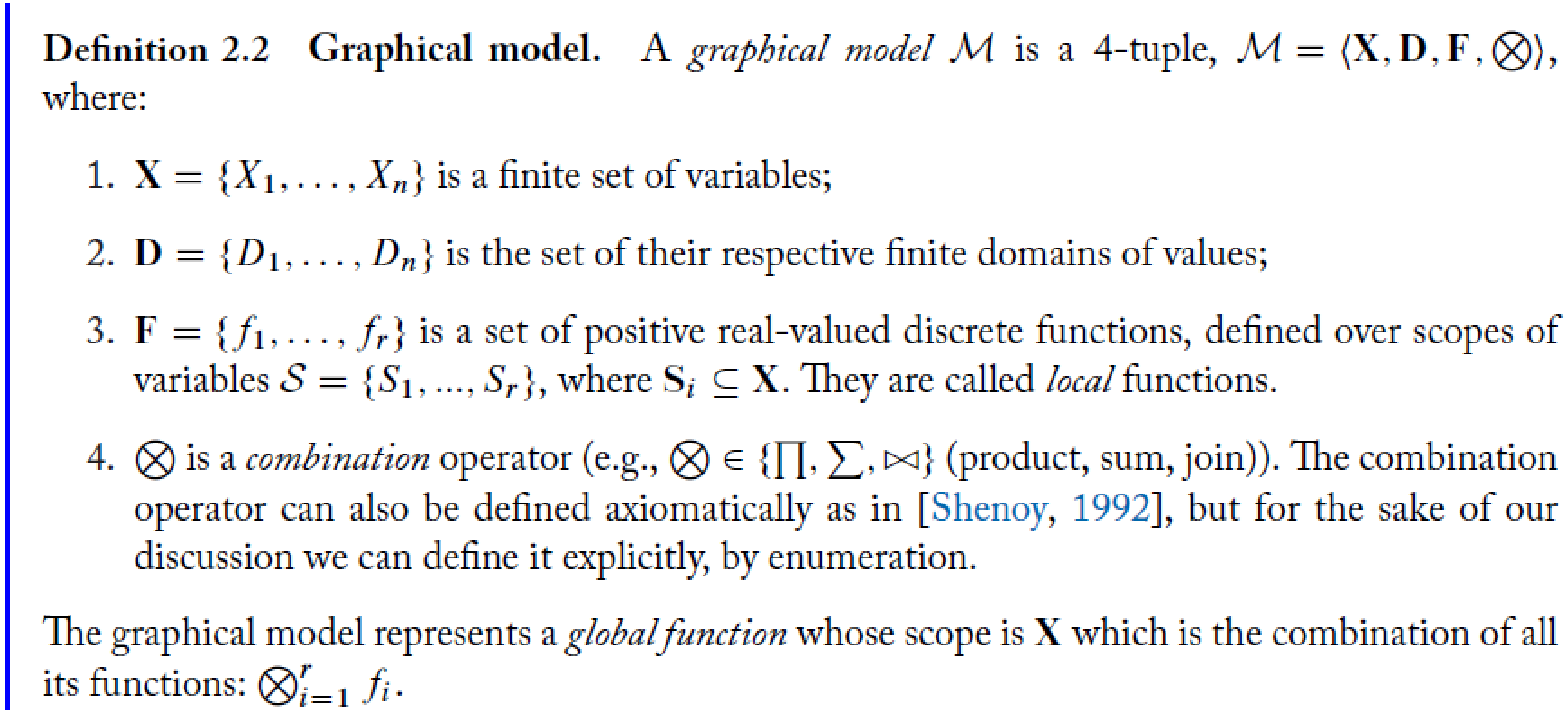
Exact Inference Algorithms Tree-Decomposition Schemes

Rina Dechter

(Dechter chapter 5, Darwiche chapter 6-7)


MORGAN & CLAYPOOL PUBLISHERS
Reasoning with Probabilistic and Deterministic Graphical Models
Exact Algorithms
2nd Edition
Rina Dechter
Synthesis Lectures on Artificial Intelligence and Machine Learning
Ronald J. Brachman, Francesca Rossi, and Peter Stone, Series Editors

General Graphical Models

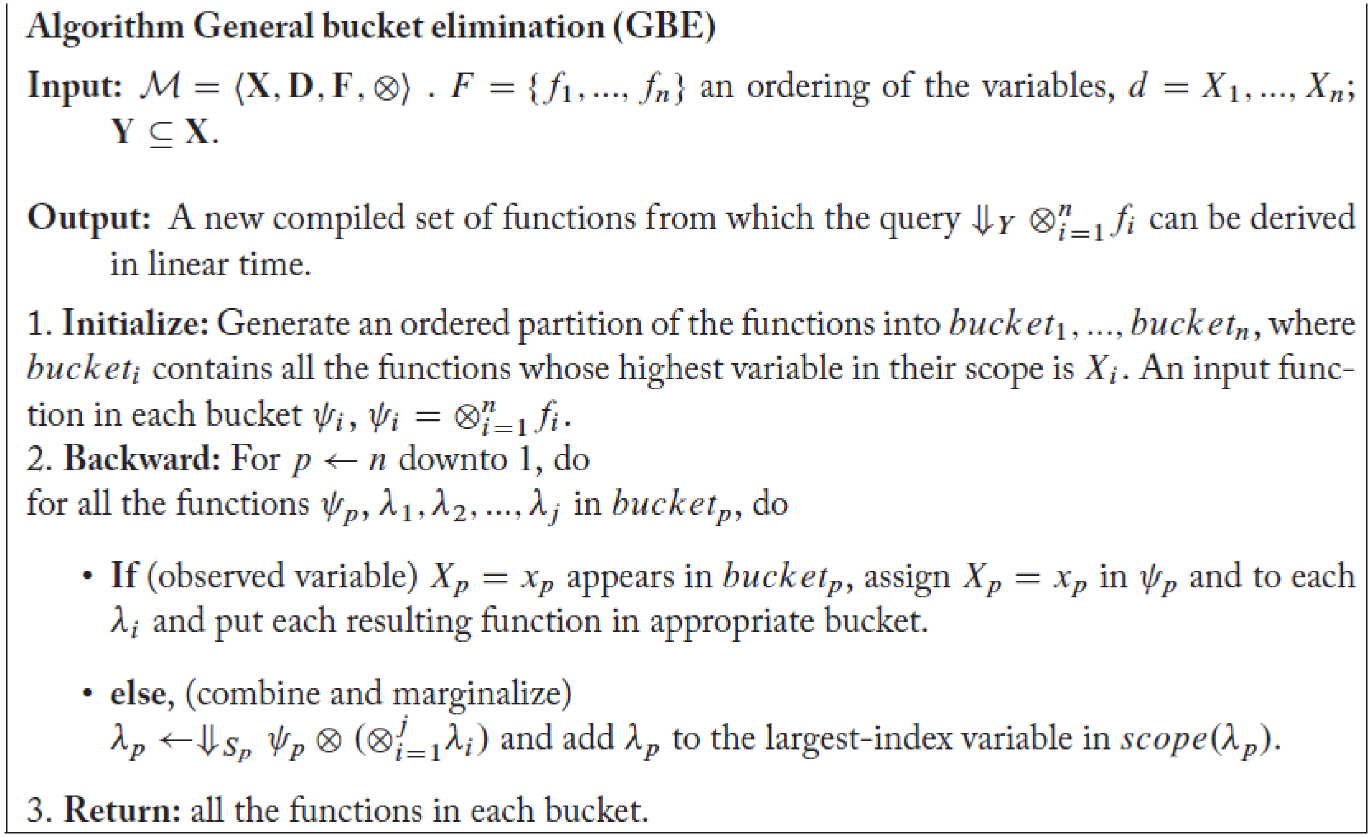


Definition 2.2 Graphical model. A *graphical model* $\mathcal{M}$ is a 4-tuple, $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \bigotimes \rangle$, where:

1. $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a finite set of variables;
2. $\mathbf{D} = \{D_1, \ldots, D_n\}$ is the set of their respective finite domains of values;
3. $\mathbf{F} = \{f_1, \ldots, f_r\}$ is a set of positive real-valued discrete functions, defined over scopes of variables $\mathcal{S} = \{S_1, ..., S_r\}$, where $\mathbf{S}_i \subseteq \mathbf{X}$. They are called *local* functions.
4. $\bigotimes$ is a *combination* operator (e.g., $\bigotimes \in \{\prod, \sum, \bowtie\}$ (product, sum, join)). The combination operator can also be defined axiomatically as in [Shenoy, 1992], but for the sake of our discussion we can define it explicitly, by enumeration.

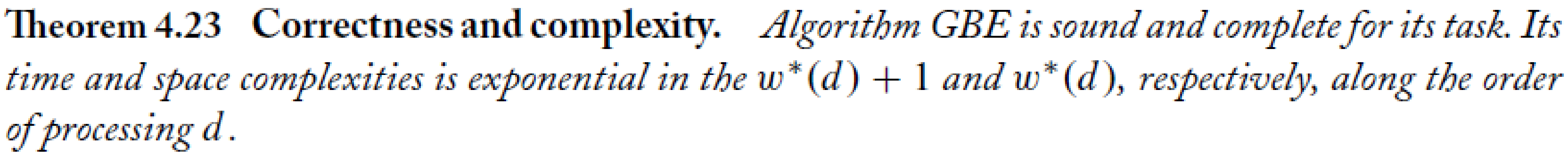
The graphical model represents a *global function* whose scope is $\mathbf{X}$ which is the combination of all its functions: $\bigotimes_{i=1}^{r} f_i$.

General Bucket Elimination



Algorithm General bucket elimination (GBE)

```
Input: $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes \rangle$ . $F = \{f_1, ..., f_n\}$ an ordering of the variables, $d = X_1, ..., X_n$;
       $\mathbf{Y} \subseteq \mathbf{X}$.

Output: A new compiled set of functions from which the query $\Downarrow_Y \otimes_{i=1}^{n} f_i$ can be derived
       in linear time.

1. Initialize: Generate an ordered partition of the functions into $bucket_1, ..., bucket_n$, where
$bucket_i$ contains all the functions whose highest variable in their scope is $X_i$. An input func-
tion in each bucket $\psi_i$, $\psi_i = \otimes_{i=1}^{n} f_i$.
2. Backward: For $p \leftarrow n$ downto 1, do
for all the functions $\psi_p, \lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$, do

    • If (observed variable) $X_p = x_p$ appears in $bucket_p$, assign $X_p = x_p$ in $\psi_p$ and to each
      $\lambda_i$ and put each resulting function in appropriate bucket.

    • else, (combine and marginalize)
      $\lambda_p \leftarrow \Downarrow_{S_p} \psi_p \otimes (\otimes_{i=1}^{j} \lambda_i)$ and add $\lambda_p$ to the largest-index variable in $scope(\lambda_p)$.

3. Return: all the functions in each bucket.
```



Theorem 4.23 Correctness and complexity. *Algorithm GBE is sound and complete for its task. Its time and space complexities is exponential in the* $w^*(d) + 1$ *and* $w^*(d)$, *respectively, along the order of processing* d.

Outline

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm
- Generating join-trees, the treewidth
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks
- Conditioning with elimination (Dechter, 7.1, 7.2)

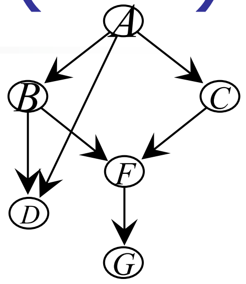
Outline

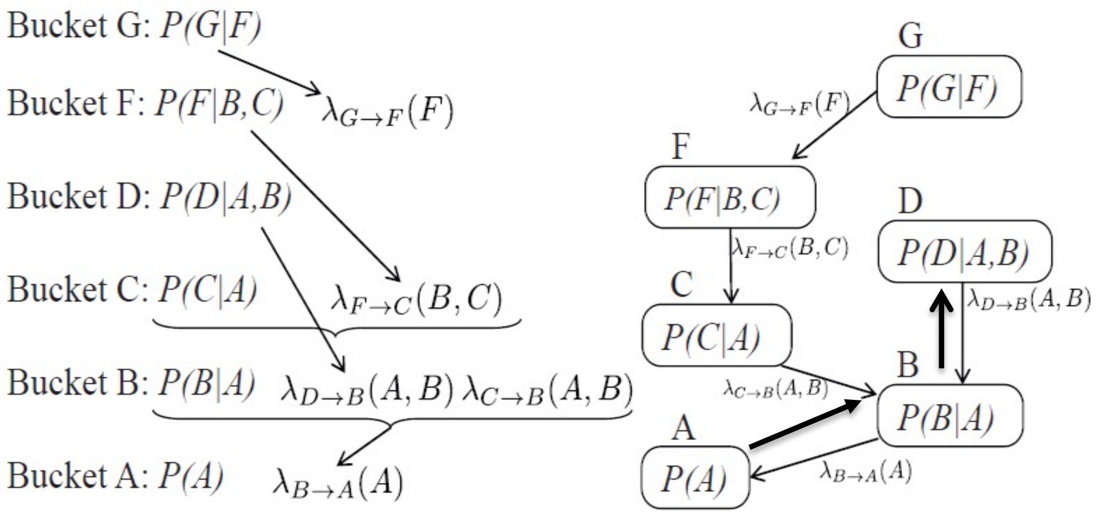
- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm
- Generating join-trees, the treewidth
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks

From BE to Bucket-Tree Elimination(BTE)

First, observe the BE operates on a tree.

Second, What if we want the marginal on D?





P(D)?

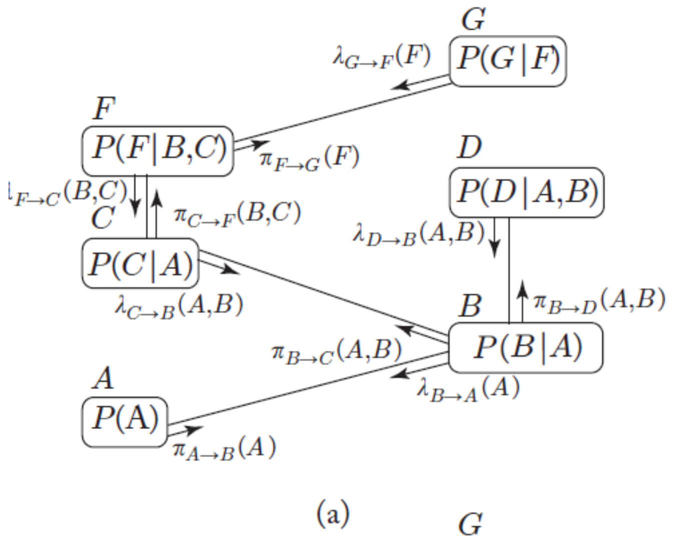
$$\pi_{A\to B}(a) = P(A),$$

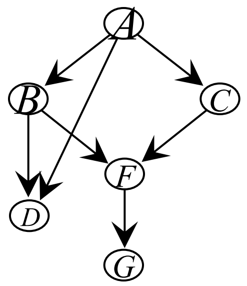
$$\pi_{B\to D}(a,b) = p(b|a)\cdot \pi_{A\to B}(a)\cdot \lambda_{C\to B}(b)$$

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

BTE: Allows Messages Both Ways

Initial buckets
+ messages

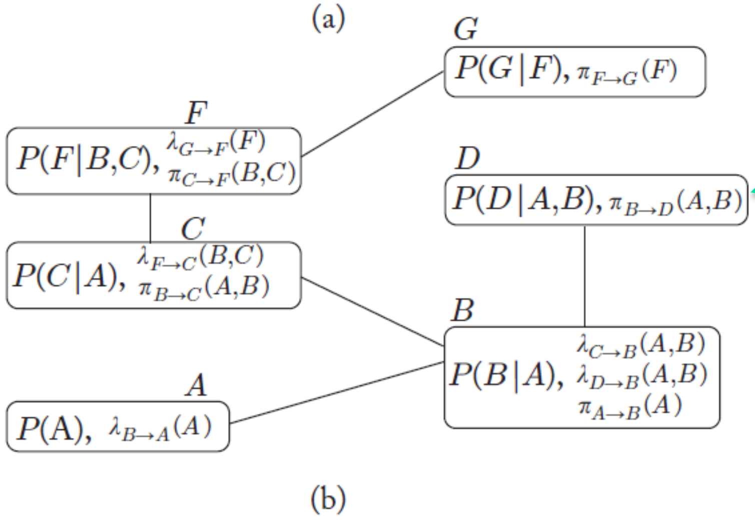




Output buckets

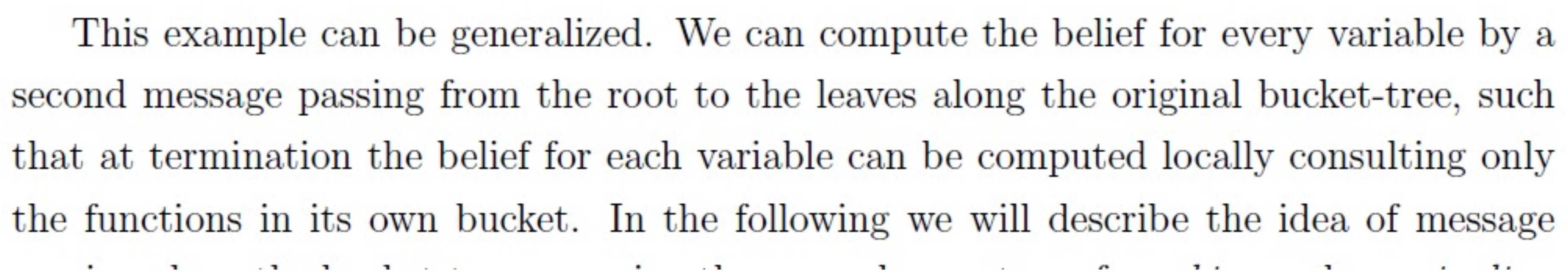
$$P(F) = \sum_{b,c} P(F|b,c)\pi_{C\to F}(b,c)\,\lambda_{G\to F}(F)$$

$$P(D) = \sum_{a,b} P(D|a,b)\,\pi_{B\to D}(a,b)$$

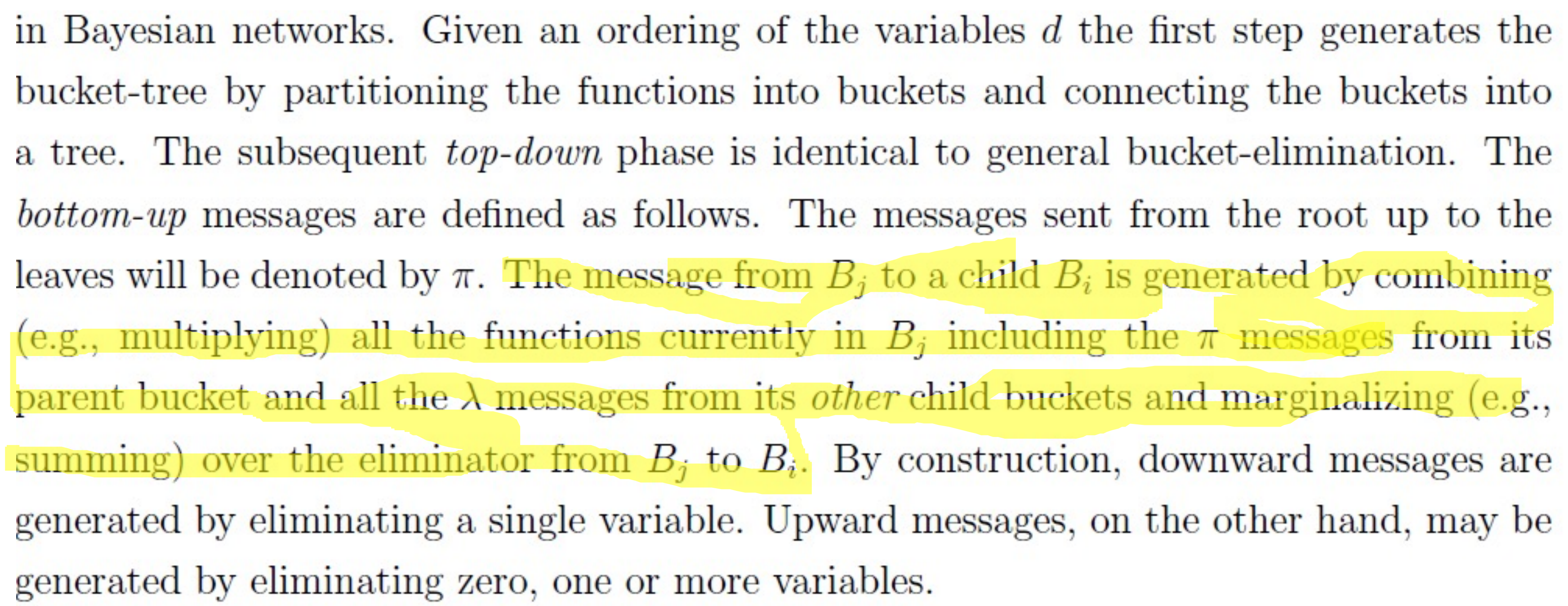


G
$P(G|F), \pi_{F\to G}(F)$
F
$P(F|B,C), \lambda_{G\to F}(F), \pi_{C\to F}(B,C)$
D
$P(D|A,B), \pi_{B\to D}(A,B)$
C
$P(C|A), \lambda_{F\to C}(B,C), \pi_{B\to C}(A,B)$
B
$P(B|A), \lambda_{C\to B}(A,B), \lambda_{D\to B}(A,B), \pi_{A\to B}(A)$
A
$P(A), \lambda_{B\to A}(A)$
(b)

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

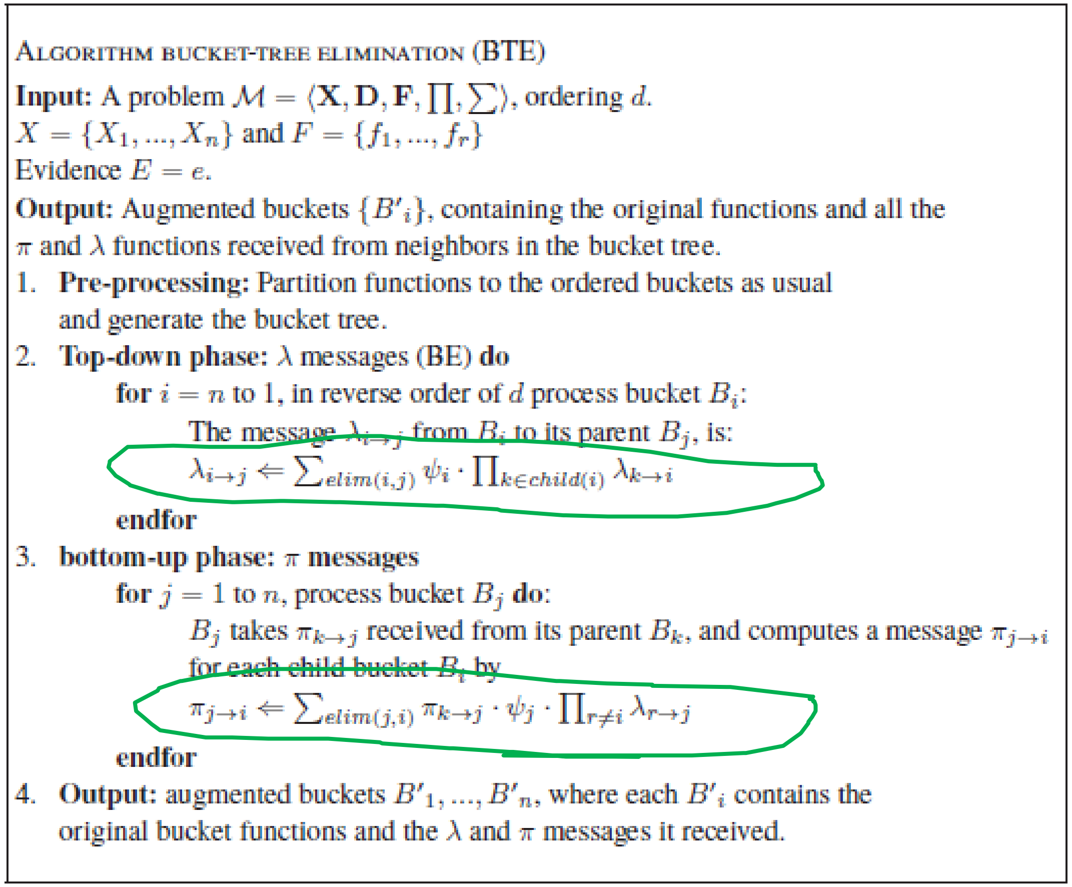


in Bayesian networks. Given an ordering of the variables d the first step generates the bucket-tree by partitioning the functions into buckets and connecting the buckets into a tree. The subsequent *top-down* phase is identical to general bucket-elimination. The *bottom-up* messages are defined as follows. The messages sent from the root up to the leaves will be denoted by π. 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 π messages from its parent bucket and all the λ 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

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

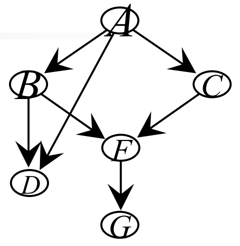
elim(i,j) = scope(B_i)– scope(B_j)



```
ALGORITHM BUCKET-TREE ELIMINATION (BTE)
Input: A problem $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod, \sum \rangle$, 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
$\pi$ 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 $\lambda_{i \to j}$ from $B_i$ to its parent $B_j$, is:
           $\lambda_{i \to j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot \prod_{k \in child(i)} \lambda_{k \to i}$
       endfor
3. bottom-up phase: $\pi$ 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
           $\pi_{j \to i} \Leftarrow \sum_{elim(j,i)} \pi_{k \to j} \cdot \psi_j \cdot \prod_{r \neq i} \lambda_{r \to j}$
       endfor
4. Output: augmented buckets $B'_1, ..., B'_n$, where each $B'_i$ contains the
   original bucket functions and the $\lambda$ and $\pi$ messages it received.
```

Figure 5.3: Algorithm bucket-tree elimination.

Bucket-Tree Construction From the Graph



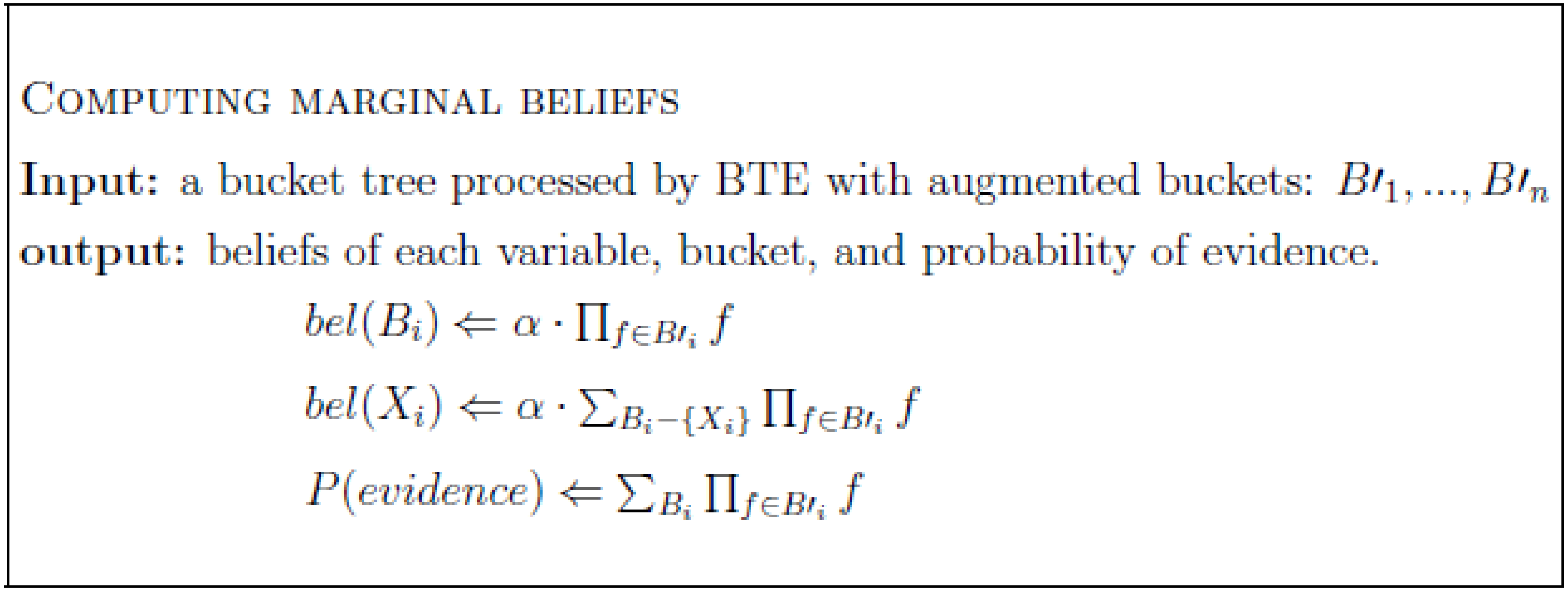
1. Pick a (good) variable ordering, d.
2. Generate the induced ordered graph
3. From top to bottom, each bucket of X is mapped to pairs (variables, functions)
4. The variables are the clique of X, the functions are those placed in the bucket
5. Connect the bucket of X to earlier bucket of Y if Y is the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

Example: Create bucket tree for ordering A,B,C,D,F,G

Asynchronous BTE: Bucket-tree Propagation (BTP)

```
BUCKET-TREE PROPAGATION (BTP)
Input: A problem $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod, \sum \rangle$, ordering $d$. $X = \{X_1, \ldots, X_n\}$ and
    $F = \{f_1, \ldots, f_r\}$, $\mathbf{E} = \mathbf{e}$. 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. Assume functions assigned with the evidence.
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
           $\boxed{\lambda_{i \to j} \Leftarrow \sum_{elim(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \to i})}$
3. Output: augmented buckets $B'_1, \ldots, B'_n$, where each $B'_i$ contains the
   original bucket functions and the $\lambda$ 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.

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

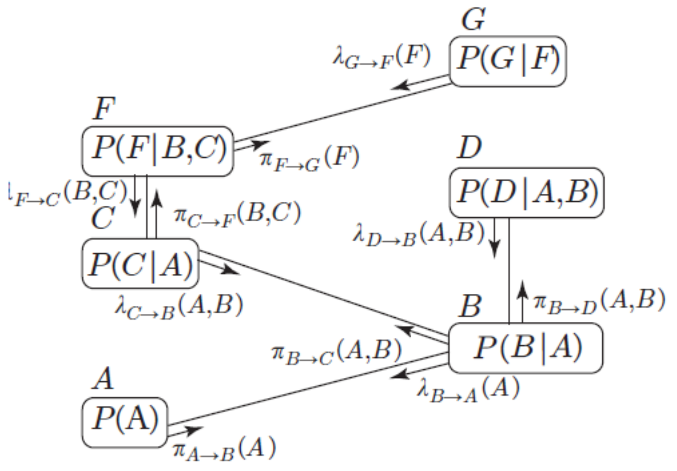
$$bel(X_i) \Leftarrow \alpha \cdot \sum_{B_i - \{X_i\}} \prod_{f \in B'_i} f$$

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

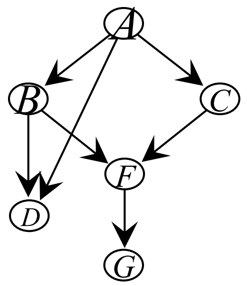
Figure 5.4: Query answering.

BTE: Allows Messages Both Ways

Initial buckets
+ messages



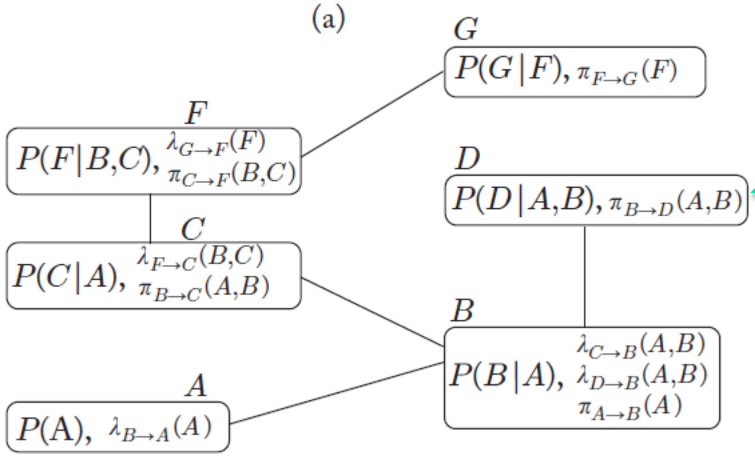
(a)



Output buckets

$$P(F) = \sum_{b,c} P(F|b,c)\pi_{C\to F}(b,c)\,\lambda_{G\to F}(F)$$

$$P(D) = \sum_{a,b} P(D|a,b)\,\pi_{B\to D}(a,b)$$



G
P(G|F), $\pi_{F\to G}(F)$
F
P(F|B,C), $\lambda_{G\to F}(F)$, $\pi_{C\to F}(B,C)$
D
P(D|A,B), $\pi_{B\to D}(A,B)$
C
P(C|A), $\lambda_{F\to C}(B,C)$, $\pi_{B\to C}(A,B)$
B
P(B|A), $\lambda_{C\to B}(A,B)$, $\lambda_{D\to B}(A,B)$, $\pi_{A\to B}(A)$
A
P(A), $\lambda_{B\to A}(A)$

(b)

Complexity of BTE/BTP on Trees

Theorem 5.6 Complexity of BTE. *Let $w^*(d)$ be the induced width of (G^*, d) where G is the primal graph of $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod, \sum \rangle$, r be the number of functions in $\mathbf{F}$ and k be the maximum domain size. The time complexity of BTE is $O(r \cdot deg \cdot k^{w^*(d)+1})$, where deg is the maximum degree of a node in the bucket tree. The space complexity of BTE is $O(n \cdot k^{w^*(d)})$.*

Proposition 5.8 BTE on trees *For tree graphical models, algorithms BTE and BTP are time and space $O(nk^2)$ and $O(nk)$, respectively, when k bound the domain size and n bounds the number of variables.*

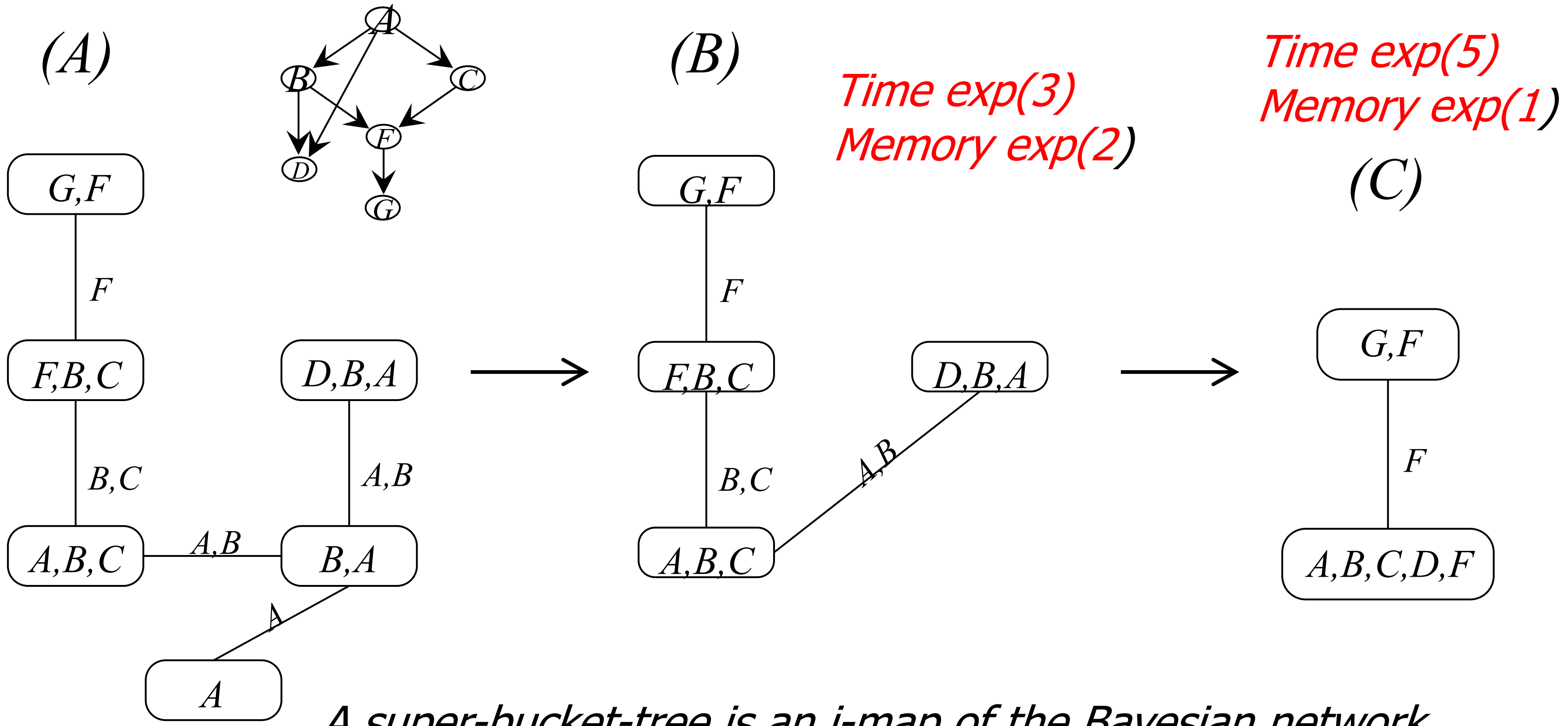
This will be extended to acyclic graphical models shortly

Outline

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm (also called, junction-tree algorithm)
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks

From Buckets to Tree-Clusters

- Merge non-maximal buckets into maximal clusters.
- Connect clusters into a tree: connect each cluster to one with which it shares a largest subset of variables.
- Separators are variable-intersection on adjacent clusters.



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

Acyclic Graphical Models

- **Dual network**: Each scope of a CPT is a node and each arc is denoted by intersection.
- **Acylic network:** when the dual graph is a tree or has a join-tree
- Tree-clustering converts a network into an acyclic one.

From Acyclic Networks

Sometime the dual graph seems to not be a tree, but it is in fact, a tree. This is because some of its arcs are redundant and can be removed while not violating the original independency relationships that is captured by the graph. [redacted]

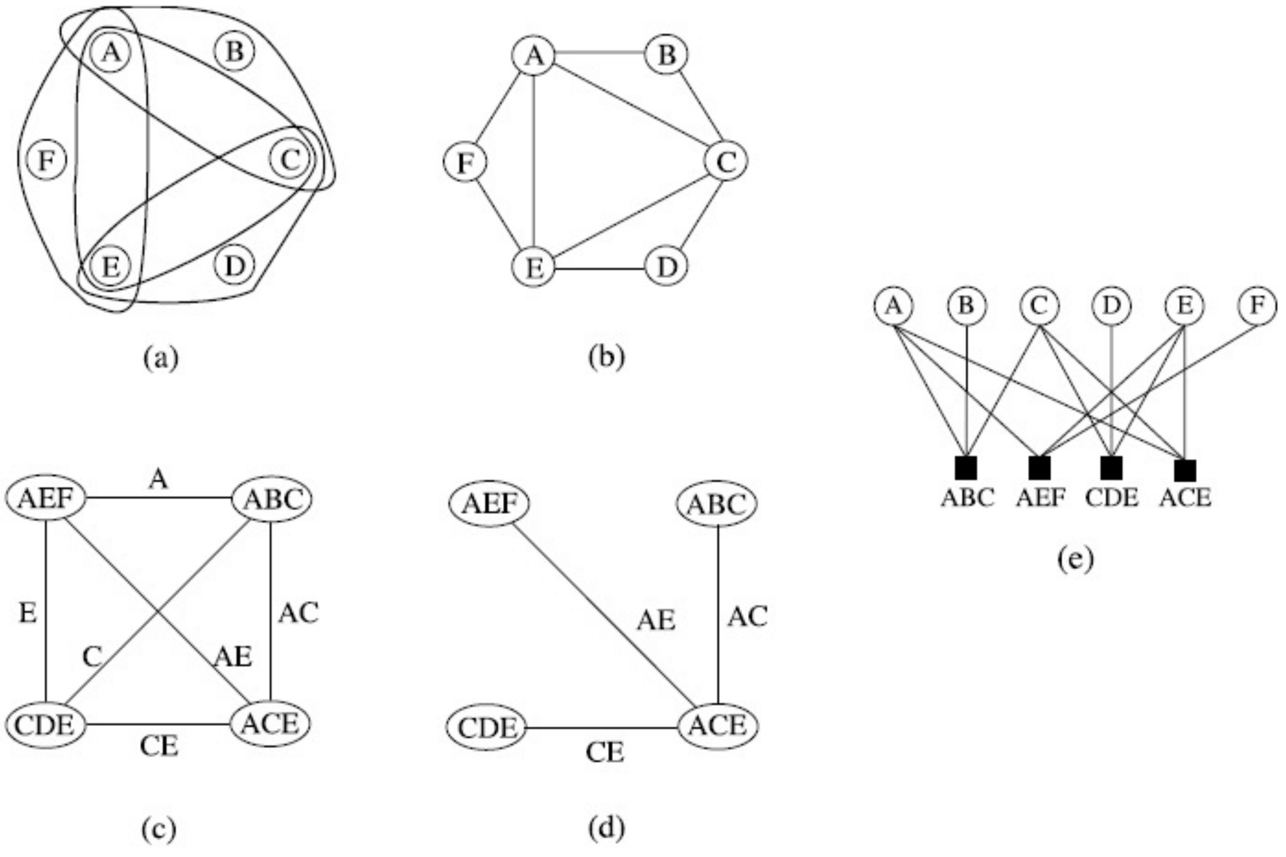
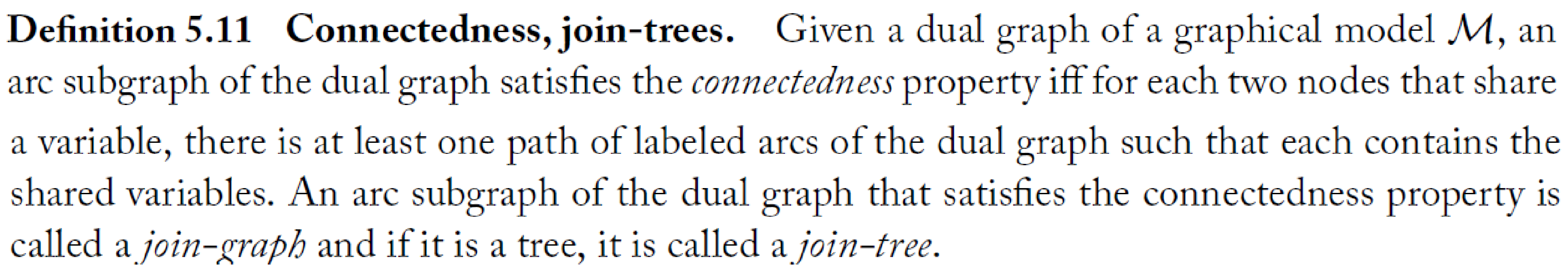


Figure 5.1: (a)Hyper, (b)Primal, (c)Dual and (d)Join-tree of a graphical model having scopes ABC, AEF, CDE and ACE. (e) the factor graph

Connectedness and Ascyclic Dual Graphs
(The Running Intersection Property)


Definition 5.11 Connectedness, join-trees. Given a dual graph of a graphical model $\mathcal{M}$, an arc subgraph of the dual graph satisfies the *connectedness* property iff for each two nodes that share a variable, there is at least one path of labeled arcs of the dual graph such that each contains the shared variables. An arc subgraph of the dual graph that satisfies the connectedness property is called a *join-graph* and if it is a tree, it is called a *join-tree*.

Definition: A graphical model whose dual graph has a join-tree is acyclic

Theorem: BTE is time and space linear on acyclic graphical models

Tree-decomposition: If we transform a general model into an acyclic one it can then be solved by a BTE/BTP scheme. Also known as tree-clustering

Tree Decompositions

A *tree decomposition* for a graphical model $< X,D,P >$ 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 :

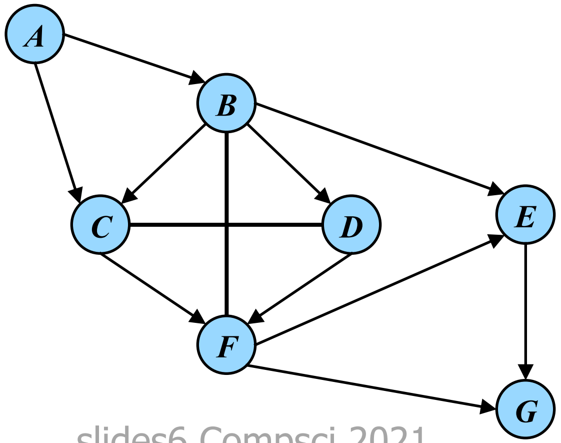
1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $scope(p_i) \subseteq \chi(v)$
2. For each variable $X_i \in X$ the set $\{v \in V | X_i \in \chi(v)\}$ forms a connected subtree (running intersection property)

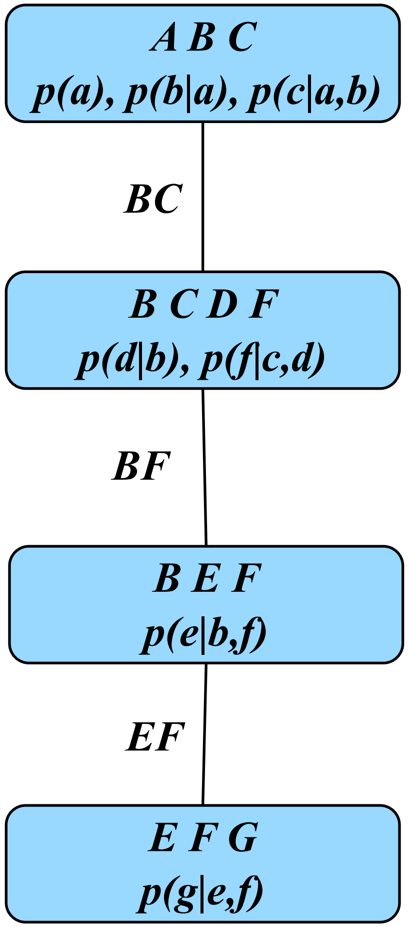
Tree Decompositions

A *tree decomposition* for a graphical model $< X,D,P >$ 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)

Connectedness, or
Running intersection property





Tree decomposition

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)

Treewidth: maximum number of variables in a node of Tree-decomposition – 1
Seperator-width: maximum intersection between adjacent nodes
Eliminator: elim(u,v) = χ(u) - χ(v)

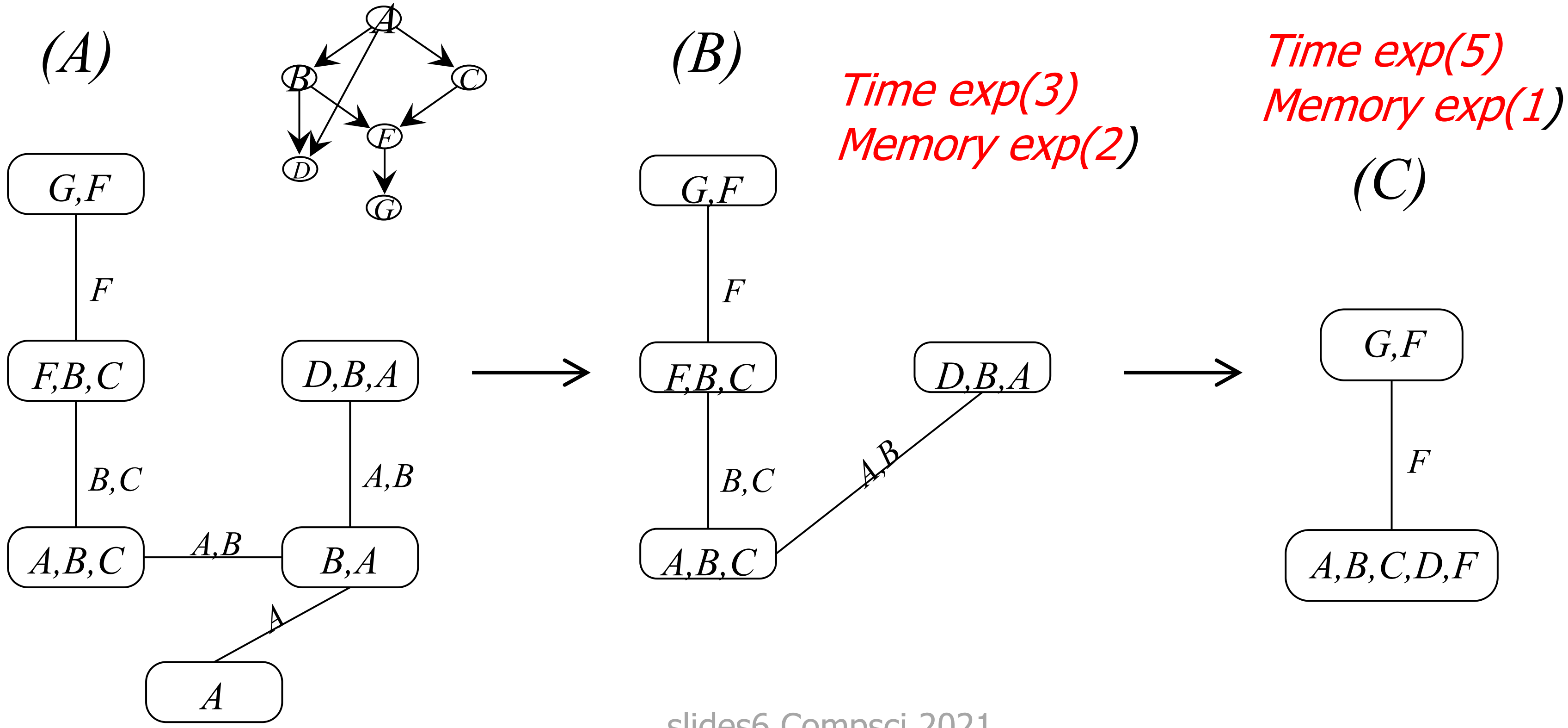
Generating Tree-Decompositions

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

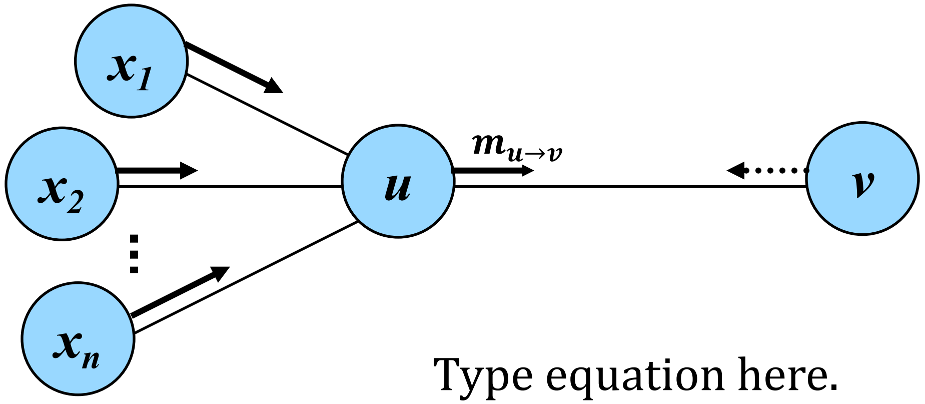
A bucket-tree of a graphical model is a tree-decomposition of the model

From Buckets to Clusters

- 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 variable-intersection on adjacent clusters.



Message Passing on a Tree Decomposition



Type equation here.

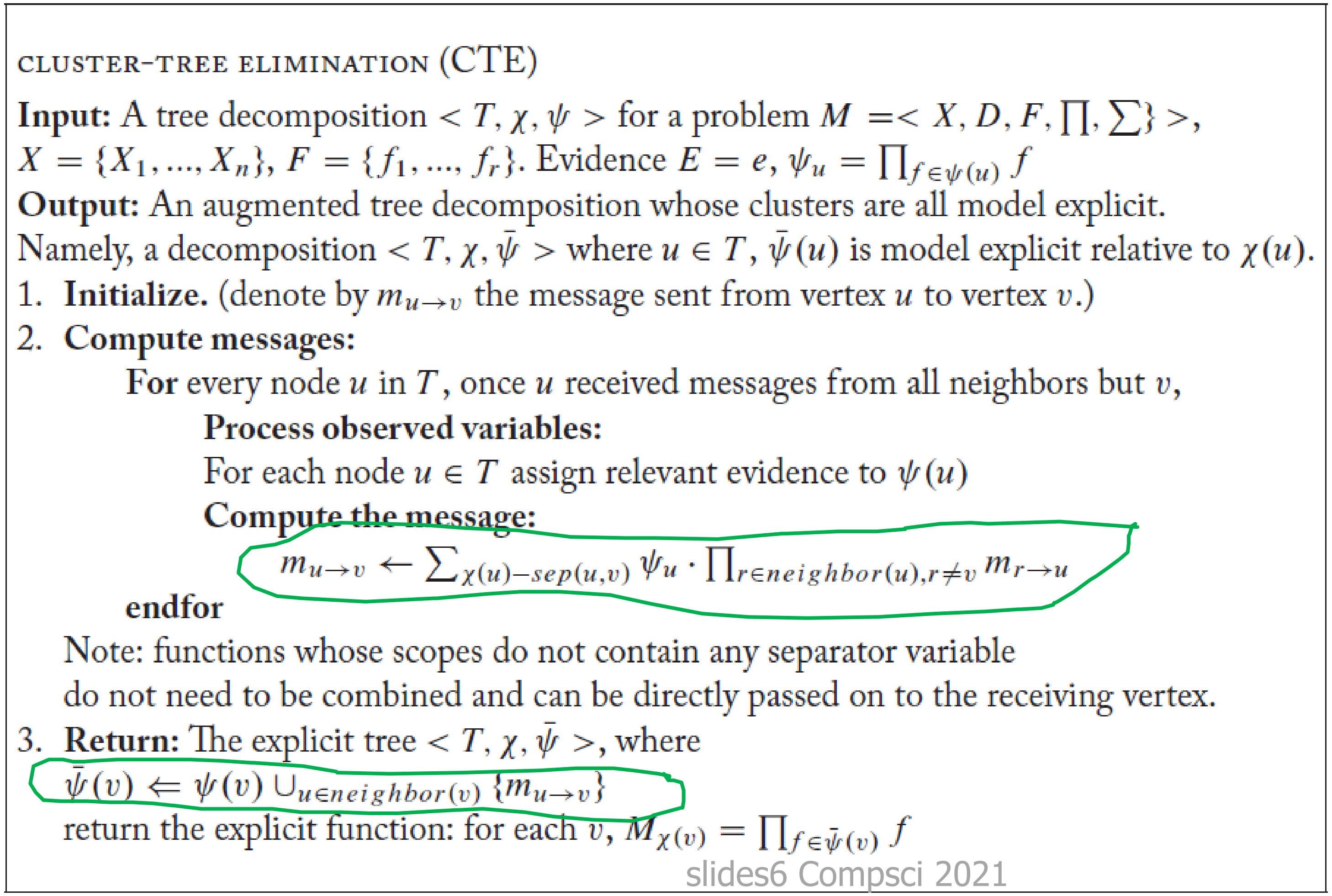
For max-product
Just replace $\sum$
With max.

Cluster(u) = $\psi(u) \cup \{m_{X_{1\to u}}, m_{X_{1\to u}}, m_{X_{2\to u}}, \dots m_{X_{n\to u}}\}$

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

$$\boldsymbol{m_{u\to v}} = \sum\nolimits_{elim(u,v)} \psi(u) \prod\nolimits_{r\, \in neighbor(u), r\neq v} \{\boldsymbol{m_{r\to u}}\}$$

Cluster-Tree Elimination



```
CLUSTER-TREE ELIMINATION (CTE)
Input: A tree decomposition $< T, \chi, \psi >$ for a problem $M =< X, D, F, \prod, \sum \} >$,
$X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$. Evidence $E = e$, $\psi_u = \prod_{f \in \psi(u)} f$
Output: An augmented tree decomposition whose clusters are all model explicit.
Namely, a decomposition $< T, \chi, \bar{\psi} >$ where $u \in T$, $\bar{\psi}(u)$ is model explicit relative to $\chi(u)$.
1. Initialize. (denote by $m_{u \to v}$ the message sent from vertex $u$ to vertex $v$.)
2. Compute messages:
        For every node $u$ in $T$, once $u$ received messages from all neighbors but $v$,
            Process observed variables:
            For each node $u \in T$ assign relevant evidence to $\psi(u)$
            Compute the message:
                $\boxed{m_{u \to v} \leftarrow \sum_{\chi(u) - sep(u,v)} \psi_u \cdot \prod_{r \in neighbor(u), r \neq v} m_{r \to u}}$
        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 $< T, \chi, \bar{\psi} >$, where
   $\boxed{\bar{\psi}(v) \Leftarrow \psi(v) \cup_{u \in neighbor(v)} \{m_{u \to v}\}}$
   return the explicit function: for each $v$, $M_{\chi(v)} = \prod_{f \in \bar{\psi}(v)} f$
```

Properties of CTE

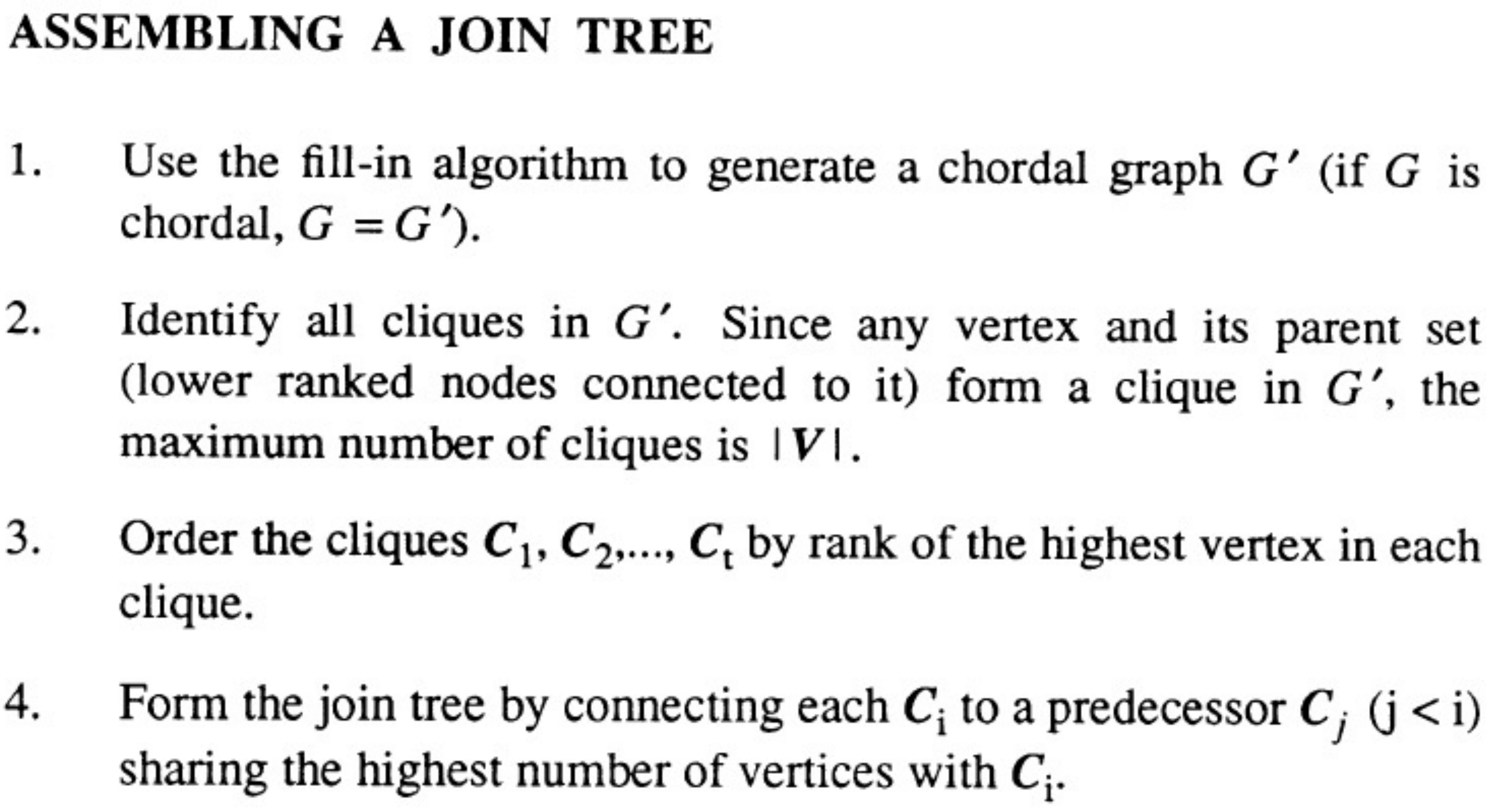
- Theorem: Correctness and completeness: Algorithm CTE is correct, i.e. it computes the exact joint probability of a single variable and the evidence. Moreover, it generates explicit clusters.

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

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

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

Generating Join-trees (Junction-trees); a special type of tree-decompositions



ASSEMBLING A JOIN TREE

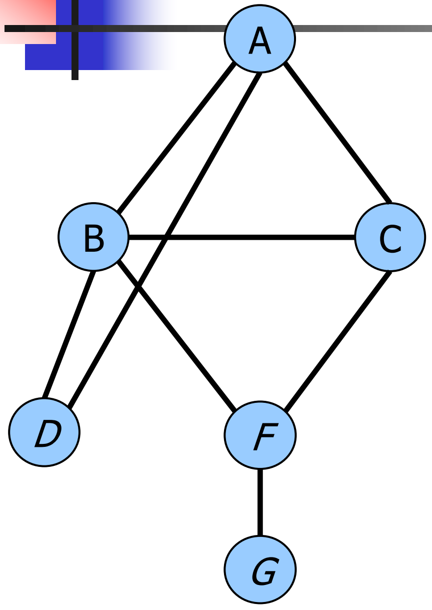
1. Use the fill-in algorithm to generate a chordal graph G' (if G is chordal, $G = G'$).

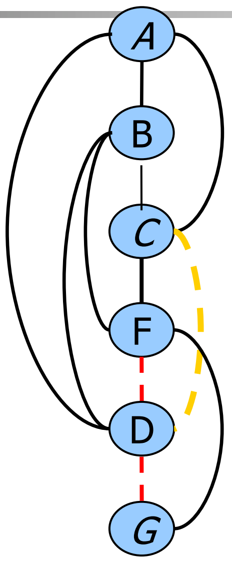
2. Identify all cliques in G'. Since any vertex and its parent set (lower ranked nodes connected to it) form a clique in G', the maximum number of cliques is $|V|$.

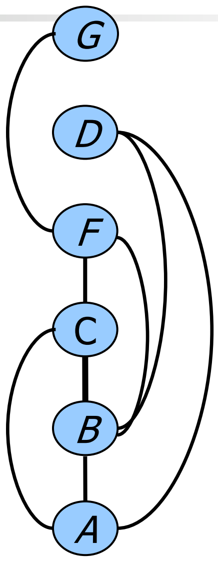
3. Order the cliques $C_1, C_2, \ldots, C_t$ by rank of the highest vertex in each clique.

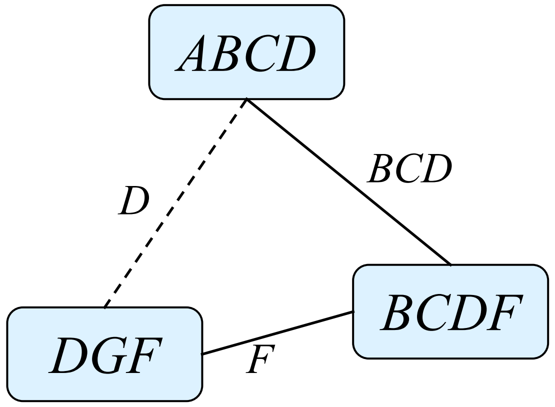
4. Form the join tree by connecting each C_i to a predecessor C_j $(j < i)$ sharing the highest number of vertices with C_i.

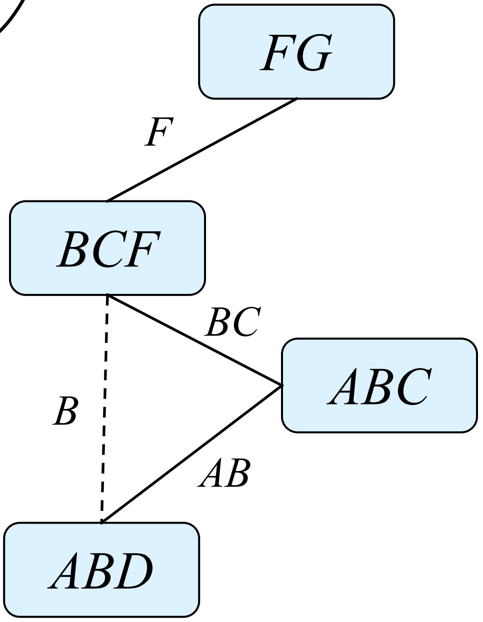
Examples of (Join)-Trees Construction



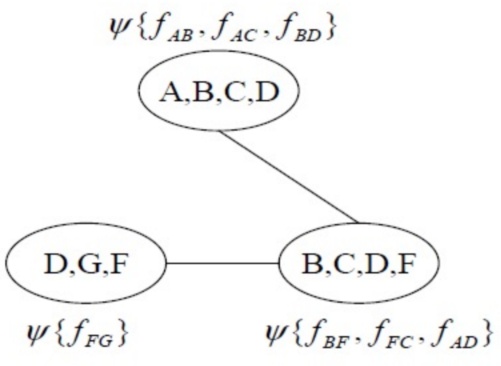




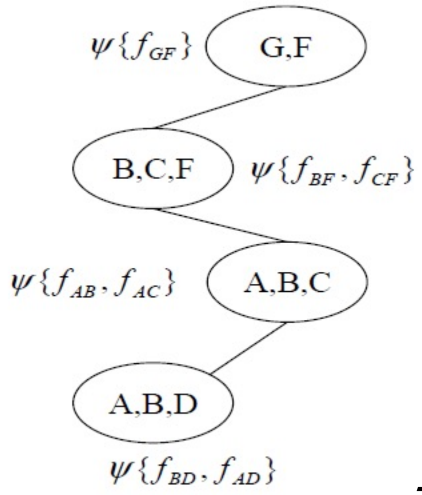




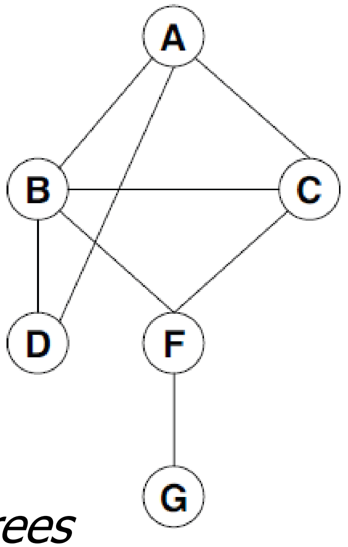
Tree-Clustering and Message-Passing


$\psi\{f_{AB}, f_{AC}, f_{BD}\}$
A,B,C,D
D,G,F
B,C,D,F
$\psi\{f_{FG}\}$
$\psi\{f_{BF}, f_{FC}, f_{AD}\}$

(a)



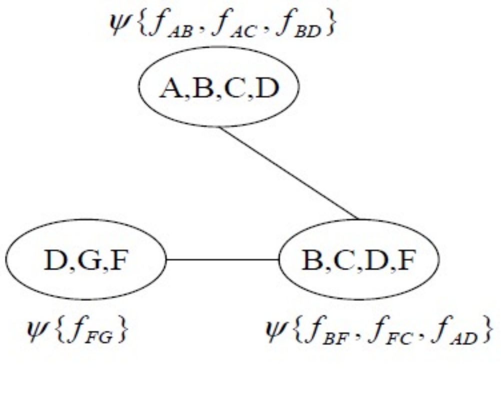
(b)



Two join-trees

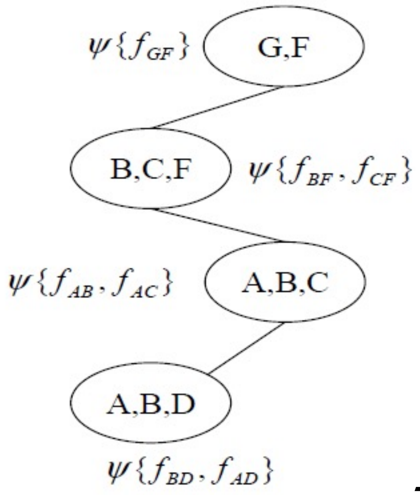
Message-passing by CTE on The tree in (b)

Tree-Clustering and Message-Passing

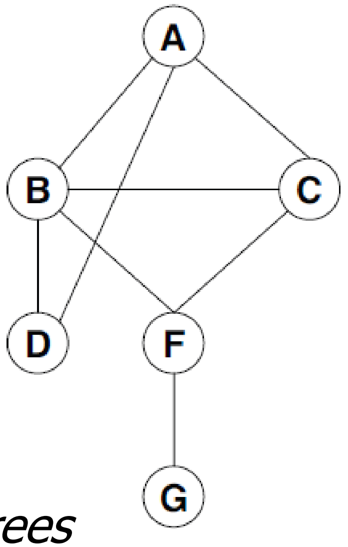


$\psi\{f_{AB}, f_{AC}, f_{BD}\}$
A,B,C,D
D,G,F
B,C,D,F
$\psi\{f_{FG}\}$
$\psi\{f_{BF}, f_{FC}, f_{AD}\}$

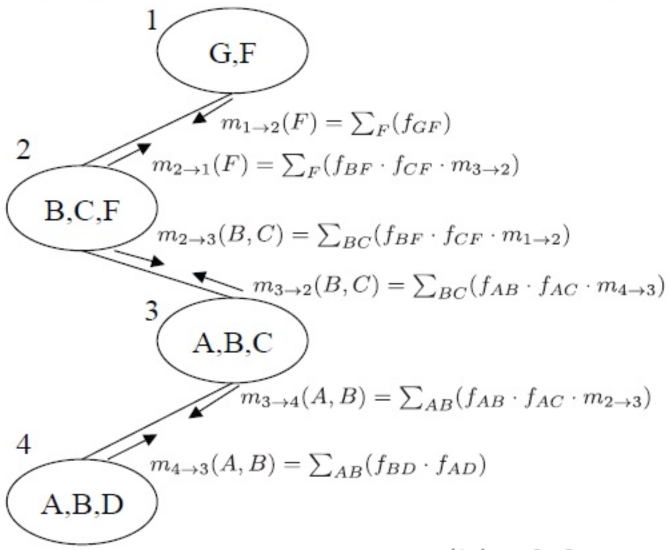
(a)



(b)



Two join-trees

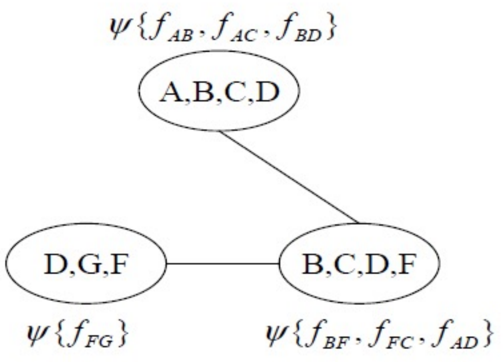


1
G,F
$m_{1\to 2}(F) = \sum_F (f_{GF})$
2
$m_{2\to 1}(F) = \sum_F (f_{BF} \cdot f_{CF} \cdot m_{3\to 2})$
B,C,F
$m_{2\to 3}(B,C) = \sum_{BC} (f_{BF} \cdot f_{CF} \cdot m_{1\to 2})$
$m_{3\to 2}(B,C) = \sum_{BC} (f_{AB} \cdot f_{AC} \cdot m_{4\to 3})$
3
A,B,C
$m_{3\to 4}(A,B) = \sum_{AB} (f_{AB} \cdot f_{AC} \cdot m_{2\to 3})$
4
$m_{4\to 3}(A,B) = \sum_{AB} (f_{BD} \cdot f_{AD})$
A,B,D

Message-passing by CTE on The tree in (b)

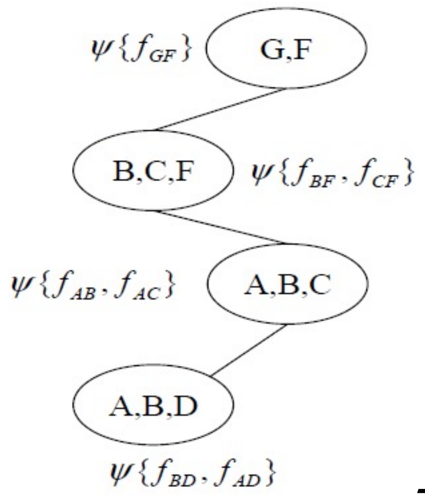
Find the errors

Tree-Clustering and Message-Passing

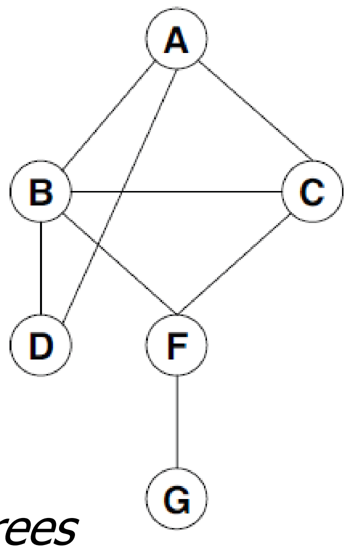


$\psi\{f_{AB}, f_{AC}, f_{BD}\}$
A,B,C,D
D,G,F
B,C,D,F
$\psi\{f_{FG}\}$
$\psi\{f_{BF}, f_{FC}, f_{AD}\}$

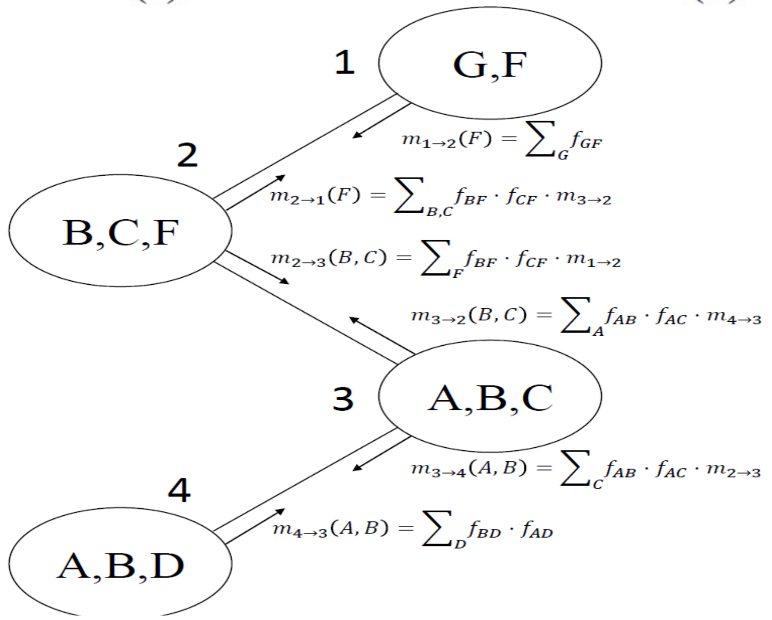
(a)



(b)



Two join-trees



1
G,F
$m_{1\to 2}(F) = \sum_G f_{GF}$
2
B,C,F
$m_{2\to 1}(F) = \sum_{B,C} f_{BF} \cdot f_{CF} \cdot m_{3\to 2}$
$m_{2\to 3}(B,C) = \sum_F f_{BF} \cdot f_{CF} \cdot m_{1\to 2}$
$m_{3\to 2}(B,C) = \sum_A f_{AB} \cdot f_{AC} \cdot m_{4\to 3}$
3
A,B,C
$m_{3\to 4}(A,B) = \sum_C f_{AB} \cdot f_{AC} \cdot m_{2\to 3}$
4
$m_{4\to 3}(A,B) = \sum_D f_{BD} \cdot f_{AD}$
A,B,D

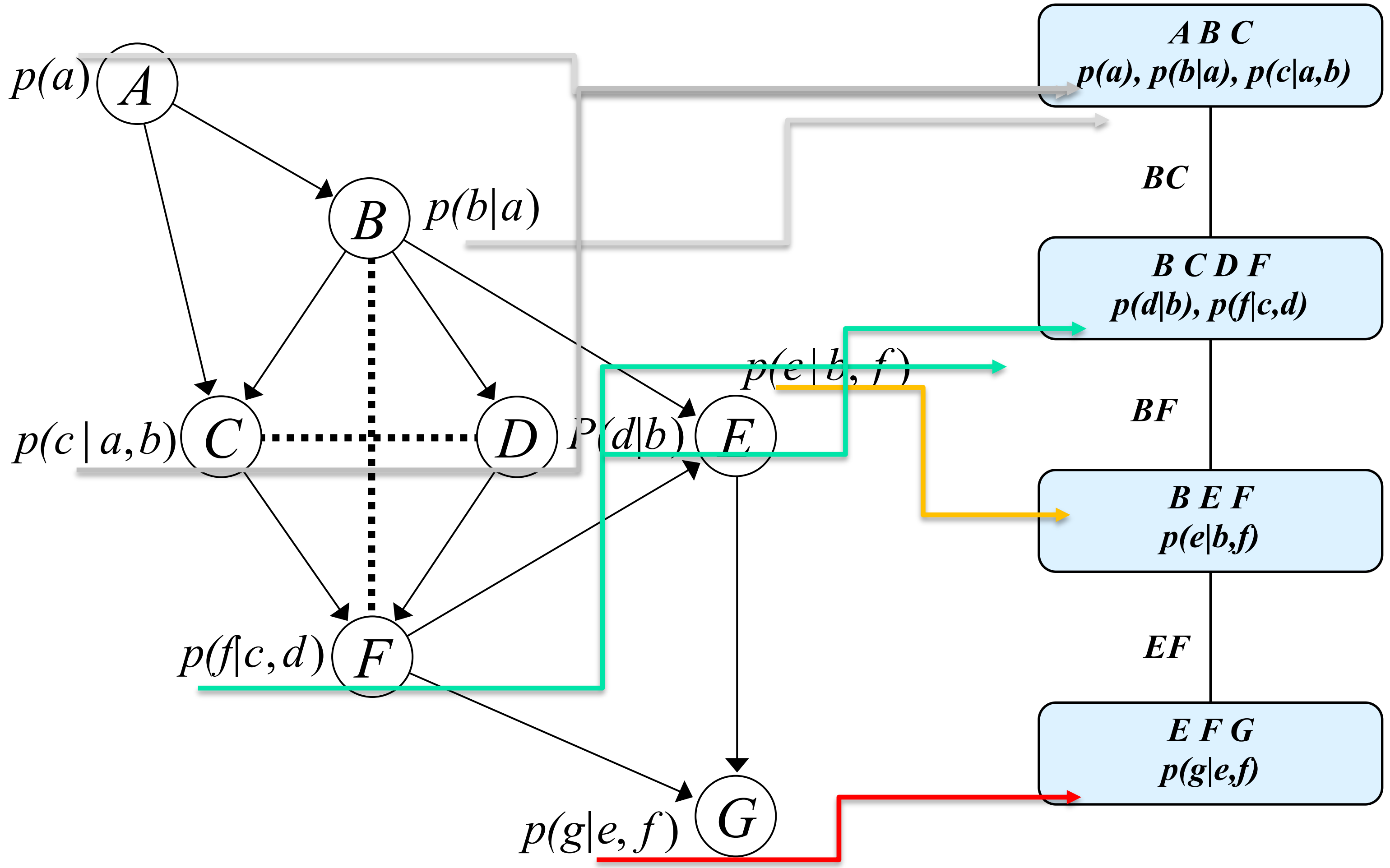
Message-passing by CTE on The tree in (b)

Correct message-passing

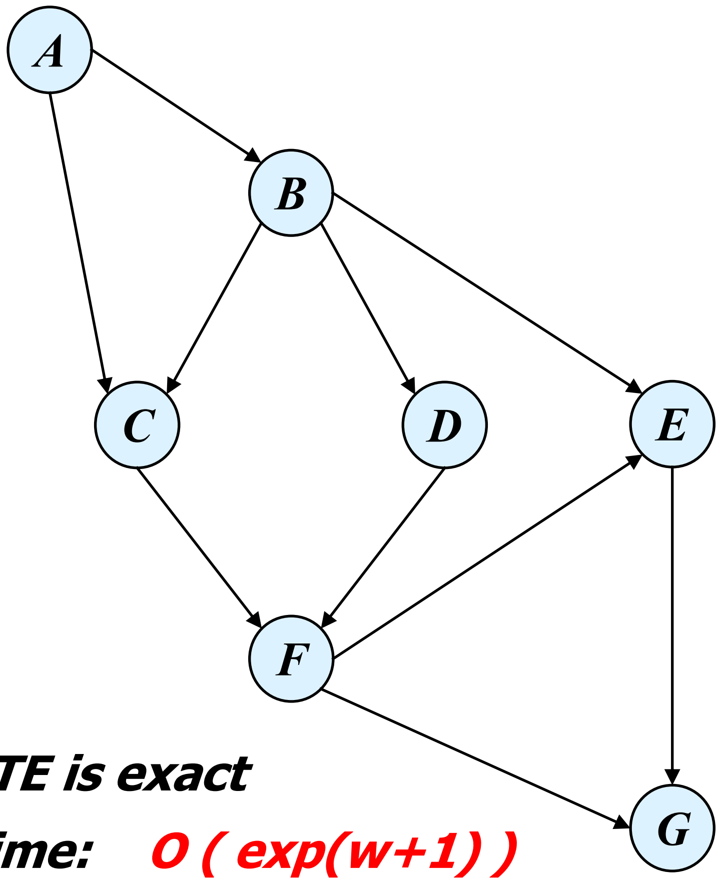
Outline

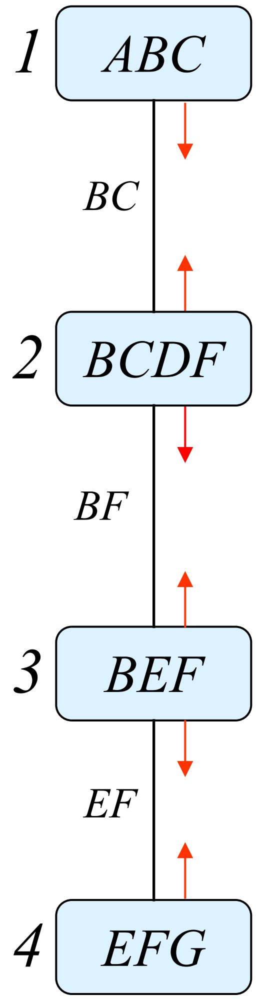
- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks

Example of a Tree Decomposition



Cluster-Tree Elimination (CTE), or Join-Tree Message-passing





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

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

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

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

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

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

CTE is exact

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

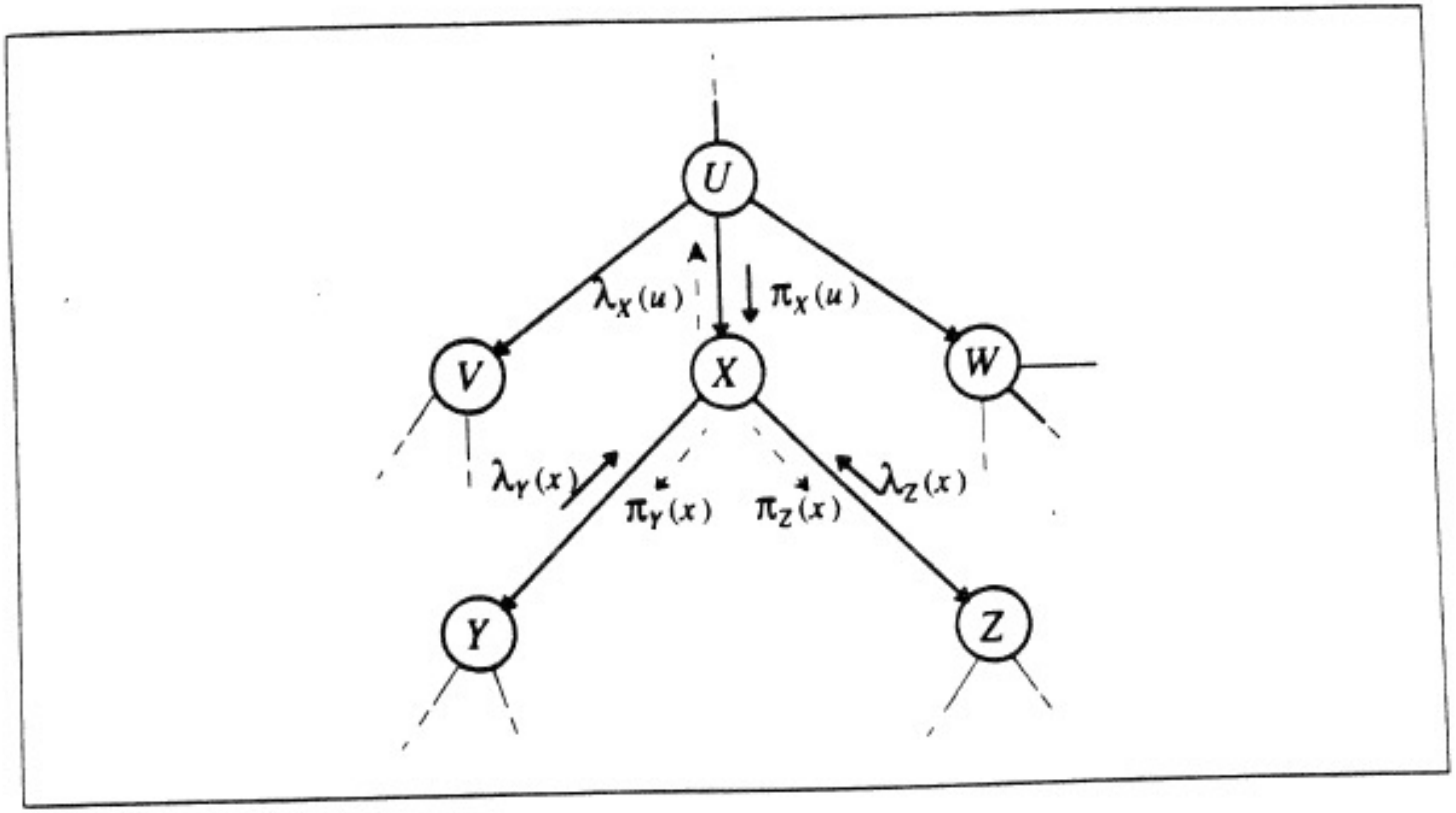
Space: ***O (exp(sep))***

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

Outline

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks

Pearl’s Belief Propagation



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.
- A polytree is an acyclic model.

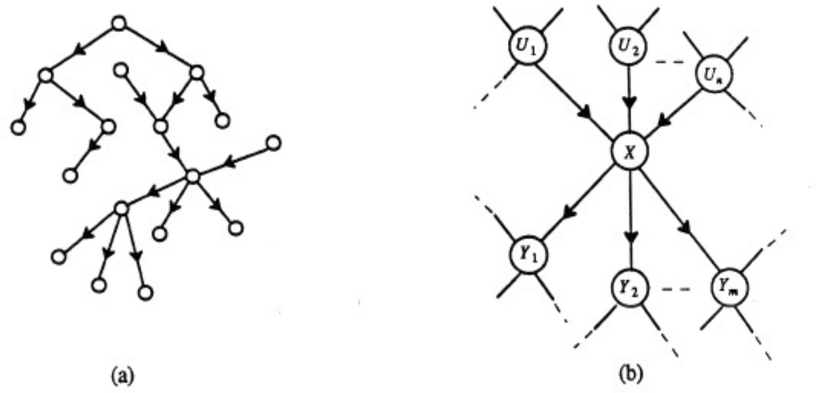
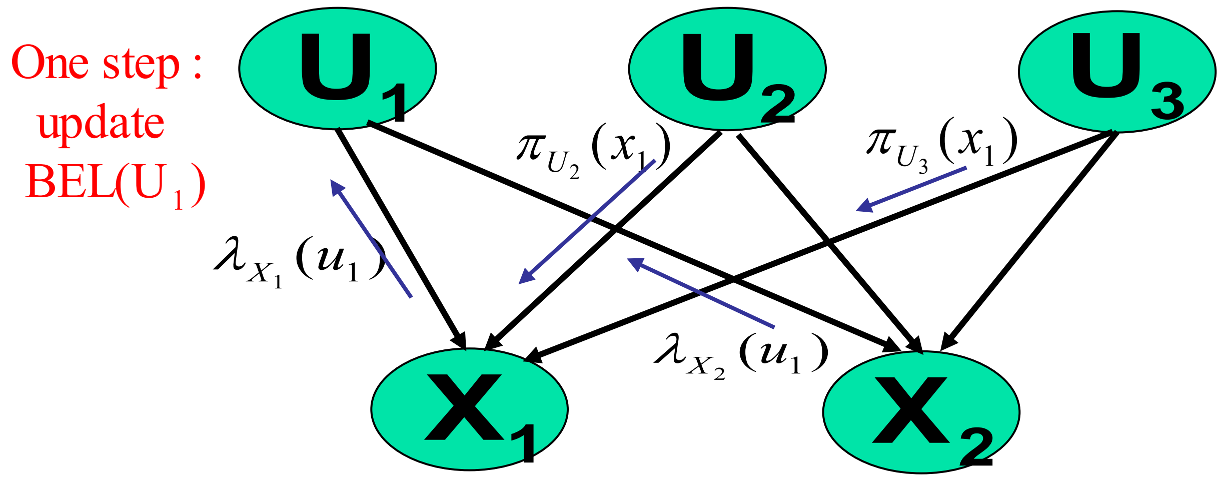


Figure 4.18. (a) A fragment of a polytree and (b) the parents and children of a typical node X.

From Exact to Approximate: Iterative Belief Propagation

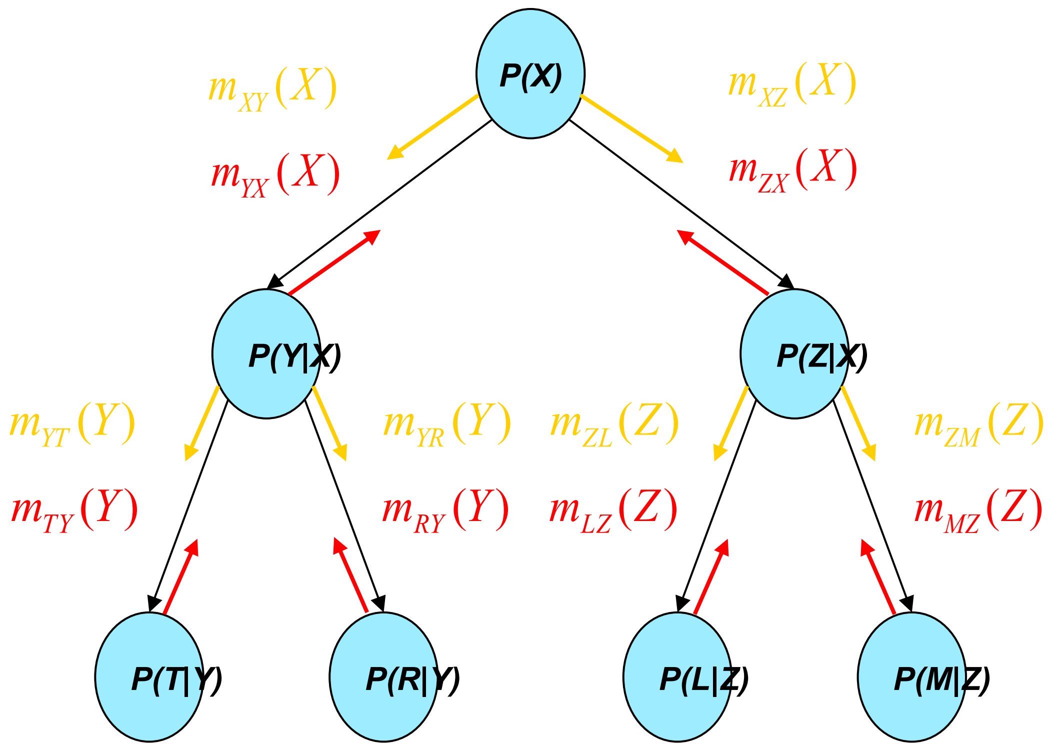
- Belief propagation is exact for poly-trees
- IBP - applying BP iteratively to cyclic networks



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

Propagation in Both Directions

- Messages can propagate both ways and we get beliefs for each variable



Agenda

- From bucket-elimination (BE) to bucket-tree elimination (BTE)
- From BTE to CTE, Acyclic networks, the join-tree algorithm
- Generating join-trees, the treewidth
- Examples of CTE for Bayesian network
- Belief-propagation on acyclic probabilistic networks (poly-trees) and Loopy networks
- Conditioning with elimination (Dechter, 7.1, 7.2)

The Idea of Cutset-Conditioning

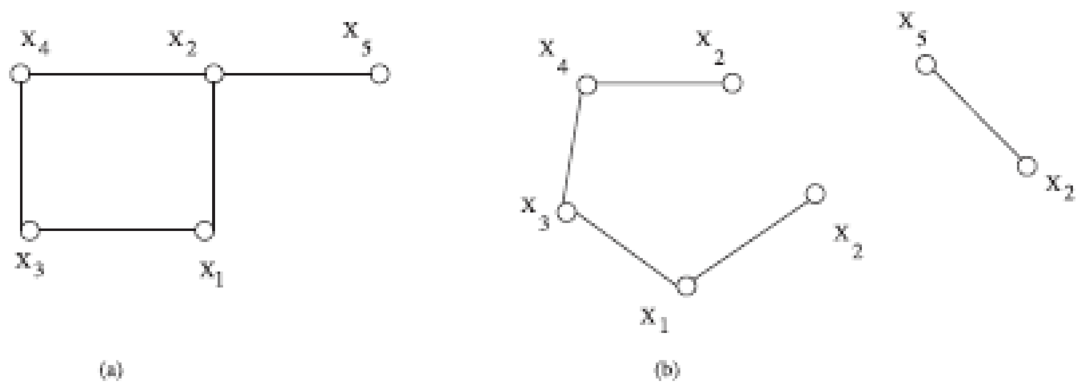
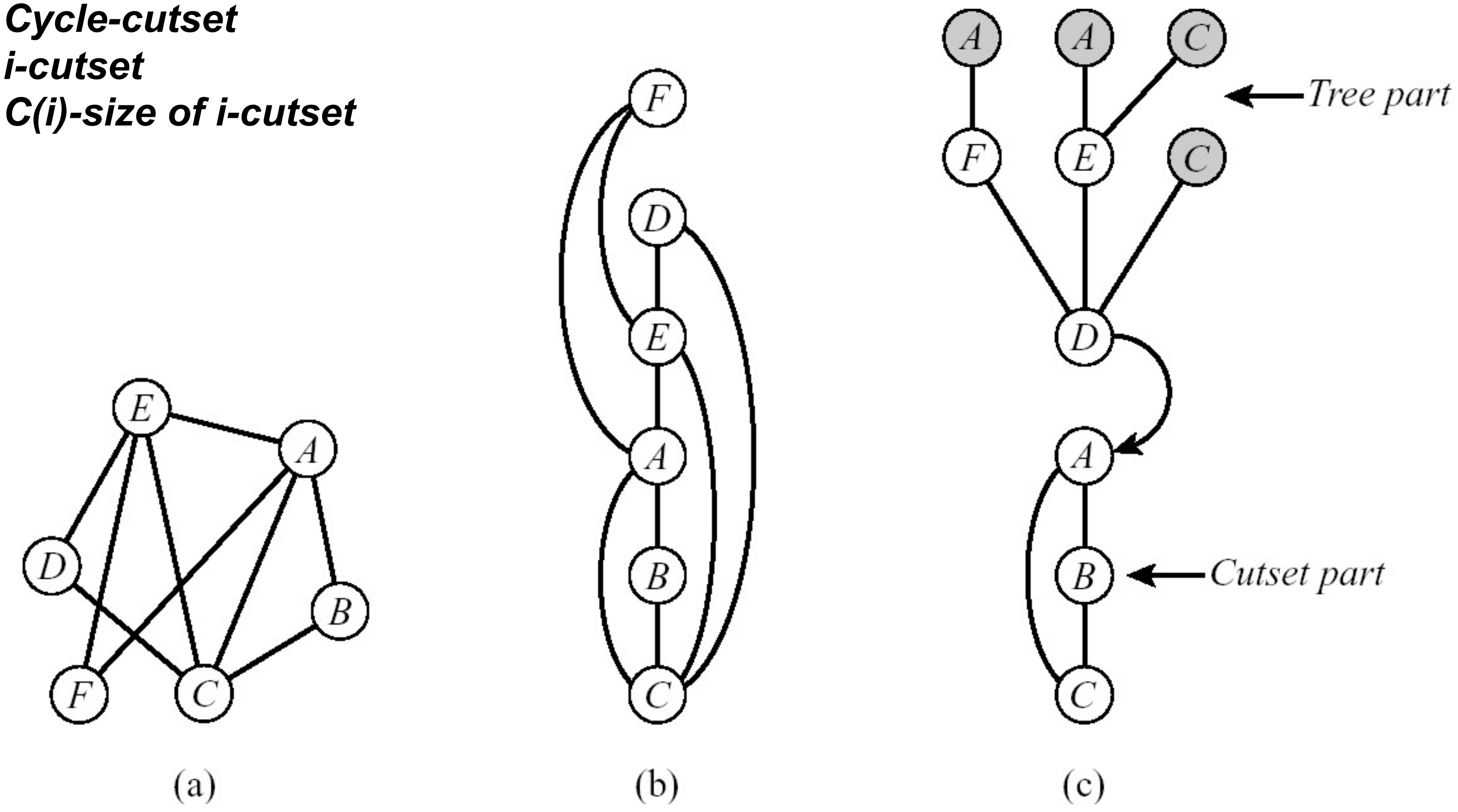


Figure 7.1: An instantiated variable cuts its own cycles.

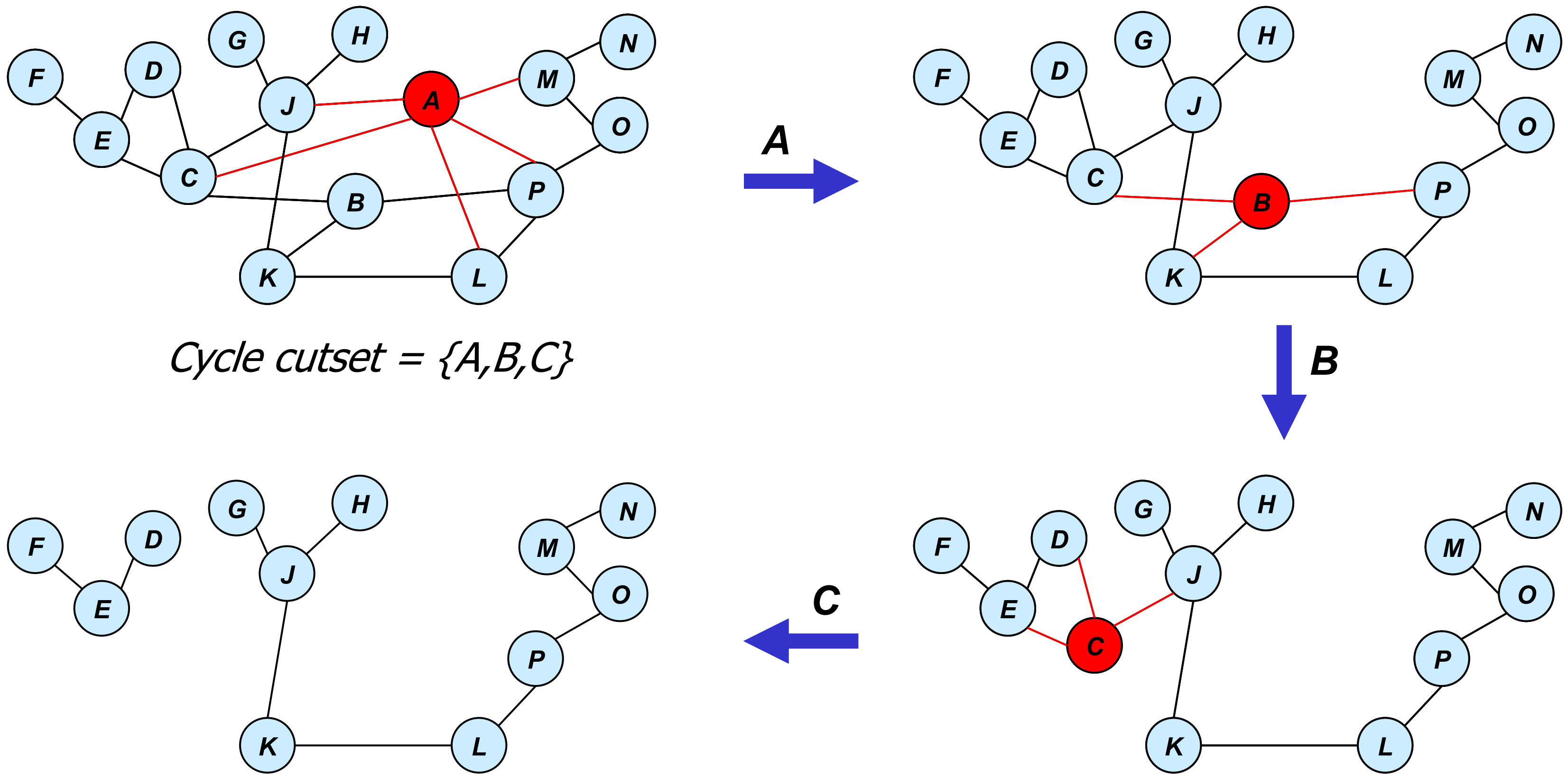
The Cycle-Cutset Scheme:
Condition Until Treeness

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



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

Cycle-Cutset Conditioning

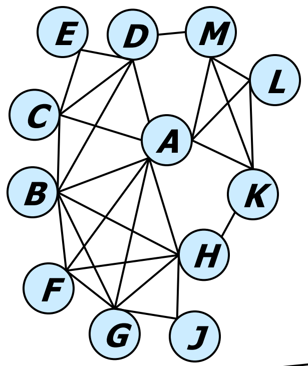


Cycle cutset = {A,B,C}

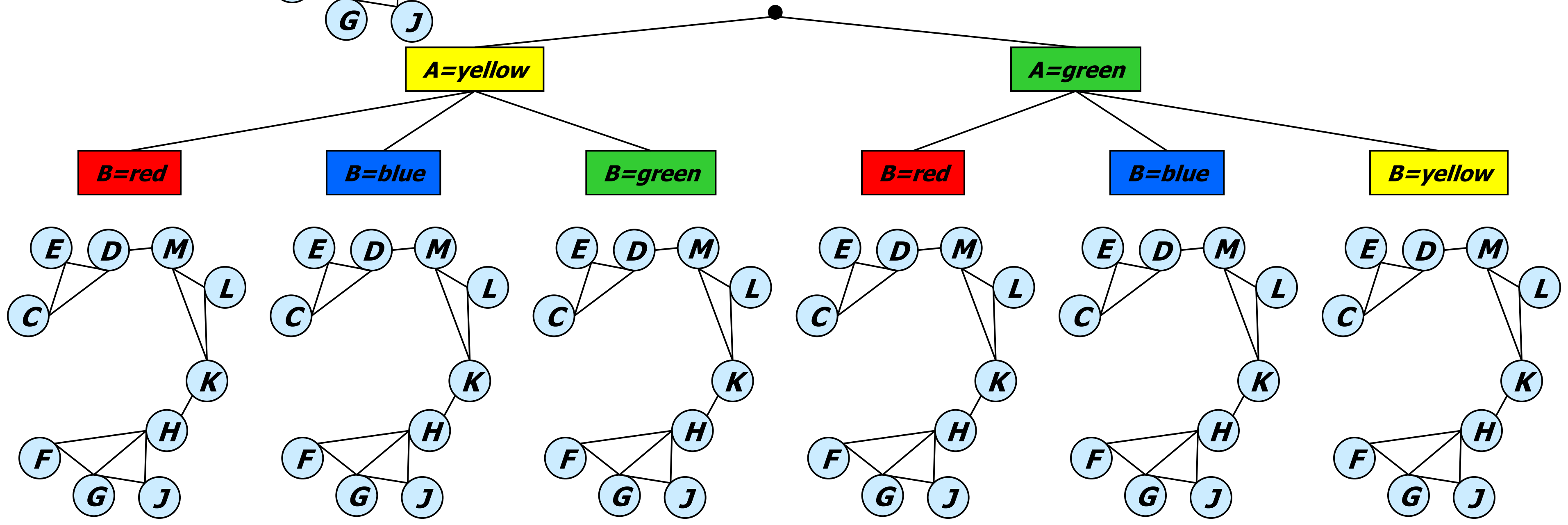
1-cutset = {A,B,C}, size 3

Search Over the Cutset (cont)

Graph Coloring problem

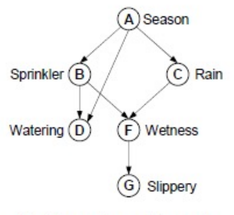


- *Inference may require too much memory*
- ***Condition*** *on some of the variables*

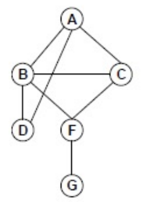


2-cutset = {A,B}, size 2

The Impact of Observations



(a) Directed acyclic graph



(b) Moral graph

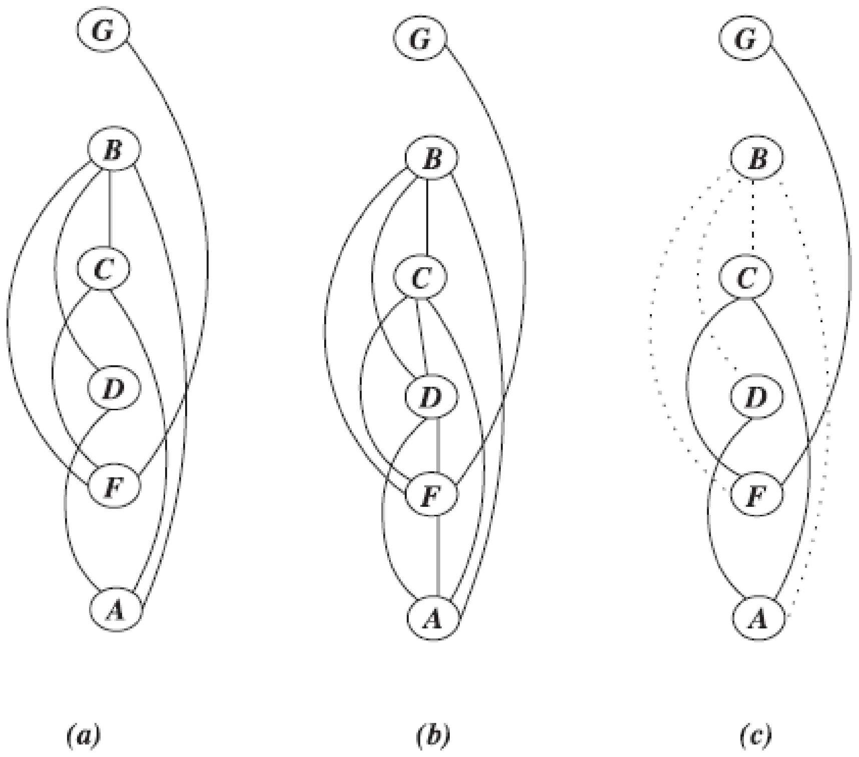


Figure 4.9: Adjusted induced graph relative to observing B.

Ordered graph *Induced graph* *Ordered conditioned graph*

The Idea of Cutset-Conditioning

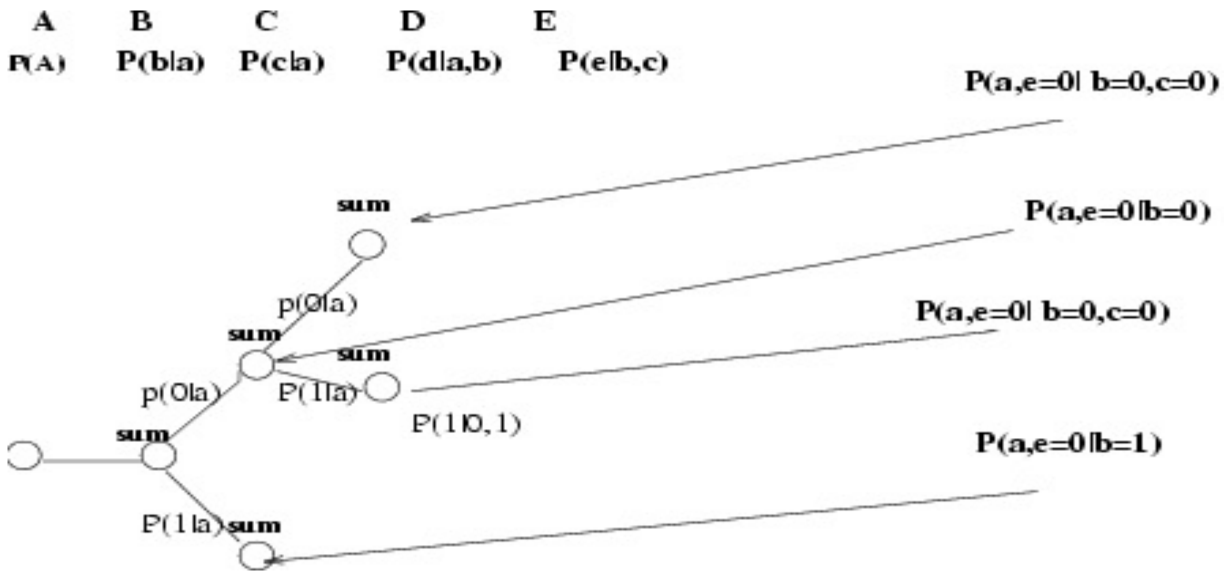
We observed that when variables are assigned, connectivity reduces.
The magnitude of saving is reflected through the "conditioned-induced graph"

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

Algorithm VEC

Conditioning+Elimination

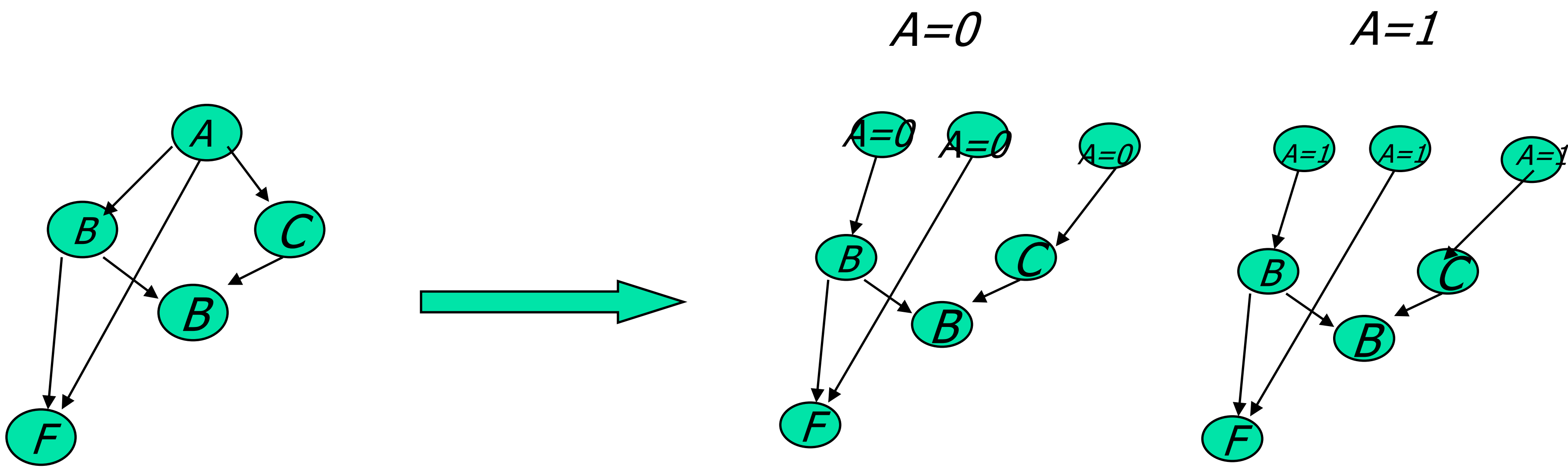
$$P(a,e=0)=P(a)\sum_b P(b\mid a)\sum_c P(c\mid a)\sum_d P(d\mid a,b)\sum_{e=0} P(e\mid b,c)$$



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

Loop-Cutset Conditioning

- You condition until you get a polytree



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

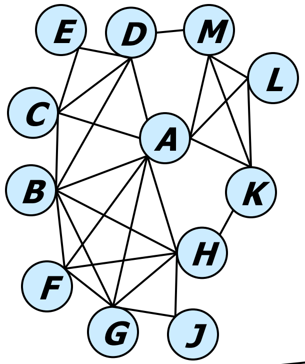
Loop-cutset method is time exponential in loop-cutset size but linear space. For each cutset we can do BP.

Loop-Cutset, q-Cutset, cycle-cutset

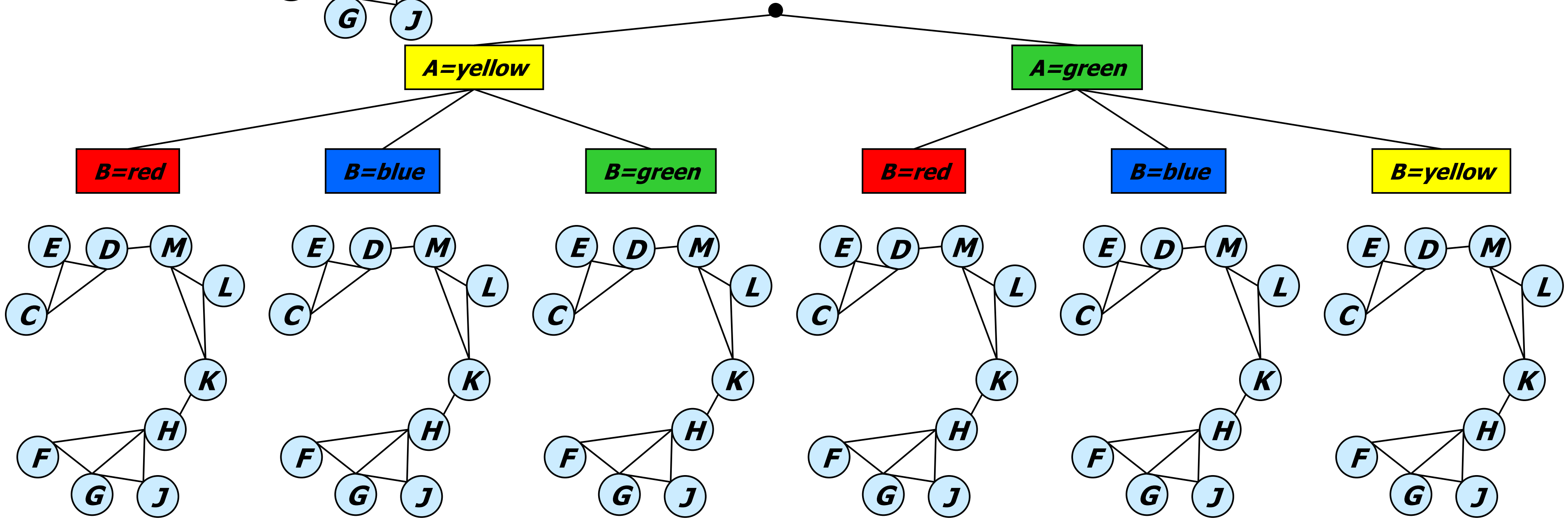
- A loop-cutset is a subset of nodes of a *directed* graph that when removed the remaining graph is a poly-tree
- A q-cutset is a subset of nodes of an *undirected* graph that when removed the remaining graph is has an induced-width of q or less.
- A cycle-cutset is a q-cutset such that q=1.

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

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

- VEC-bel:
  - Identify a q-cutset, C, of the network
  - For each assignment to C=c solve the conditioned sub-problem by CTE or BE.
  - Accumulate probabilities.
  - Time complexity: nk^{c+q+1}
  - Space complexity: nk^{q}

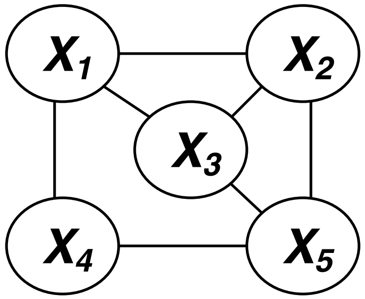
Algorithm VEC (Variable-elimination with conditioning)

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

VEC and ALT-VEC: Alternate conditioning and Elimination

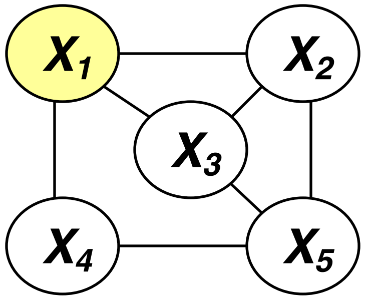
- VEC (q-cutset-conditioining) can also alternate search and elimination, yielding ALT-VEC.
- A time-space tradeoff

Search Basic Step: Conditioning

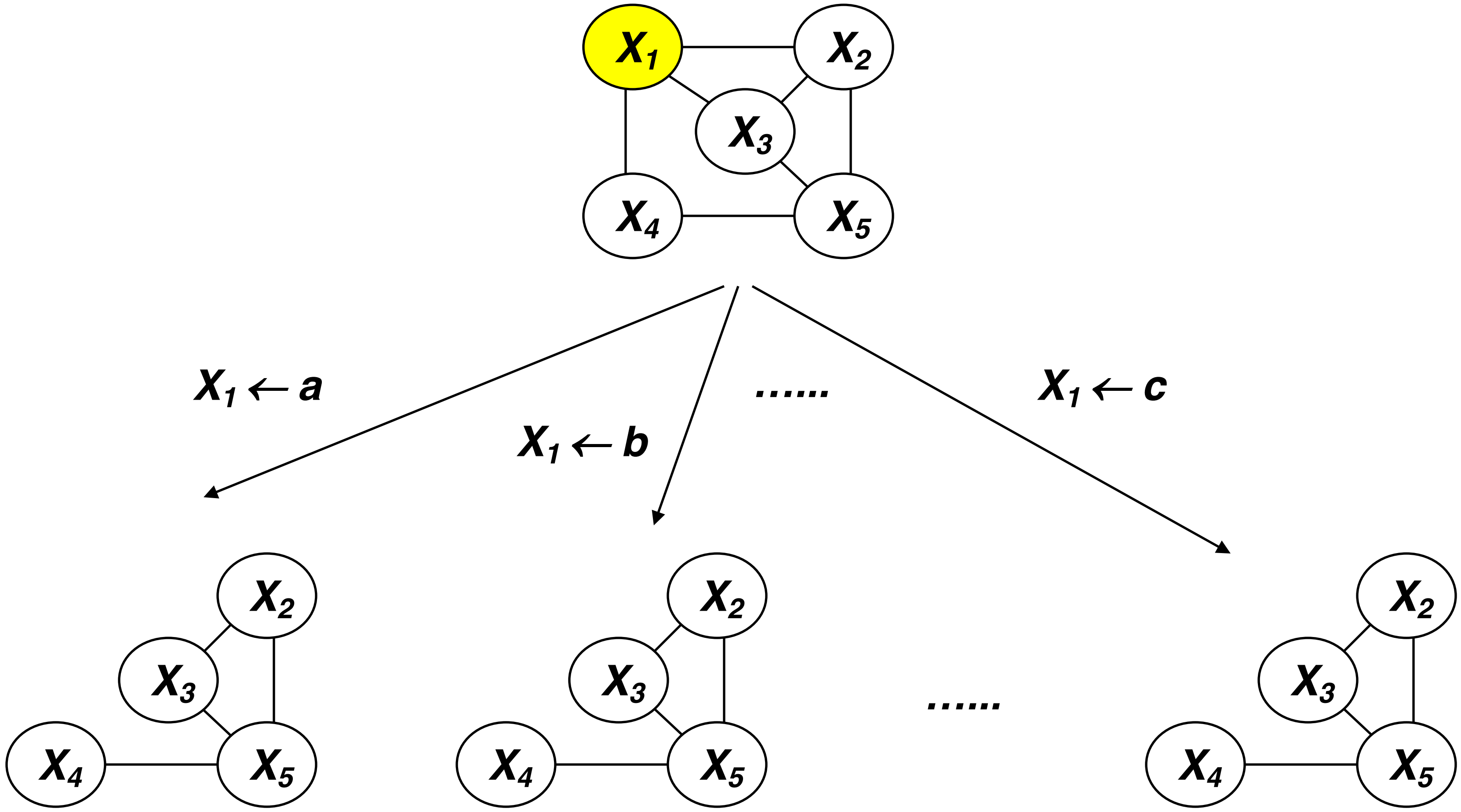


Search Basic Step: Conditioning

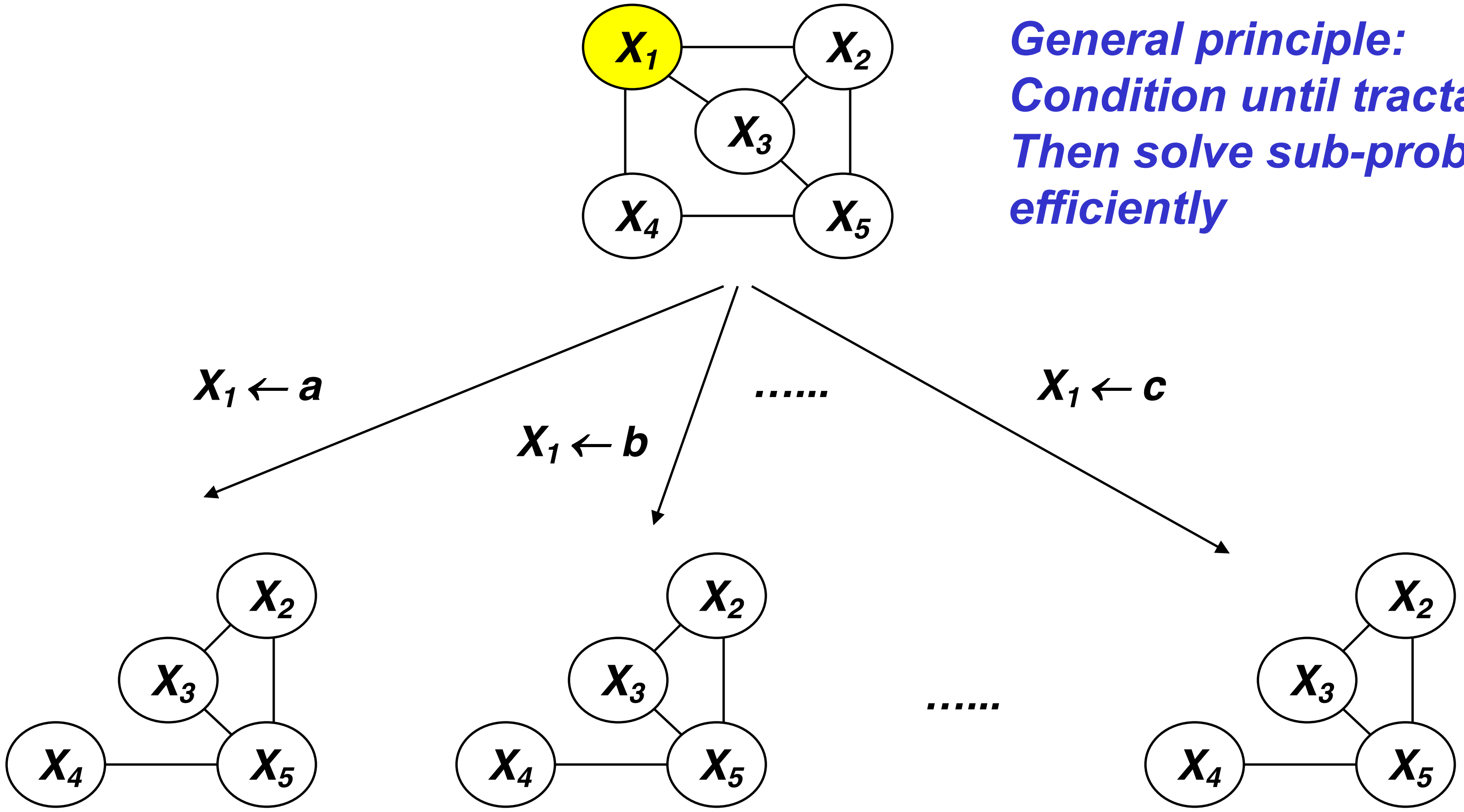
- ***Select a variable***



Search Basic Step: Conditioning

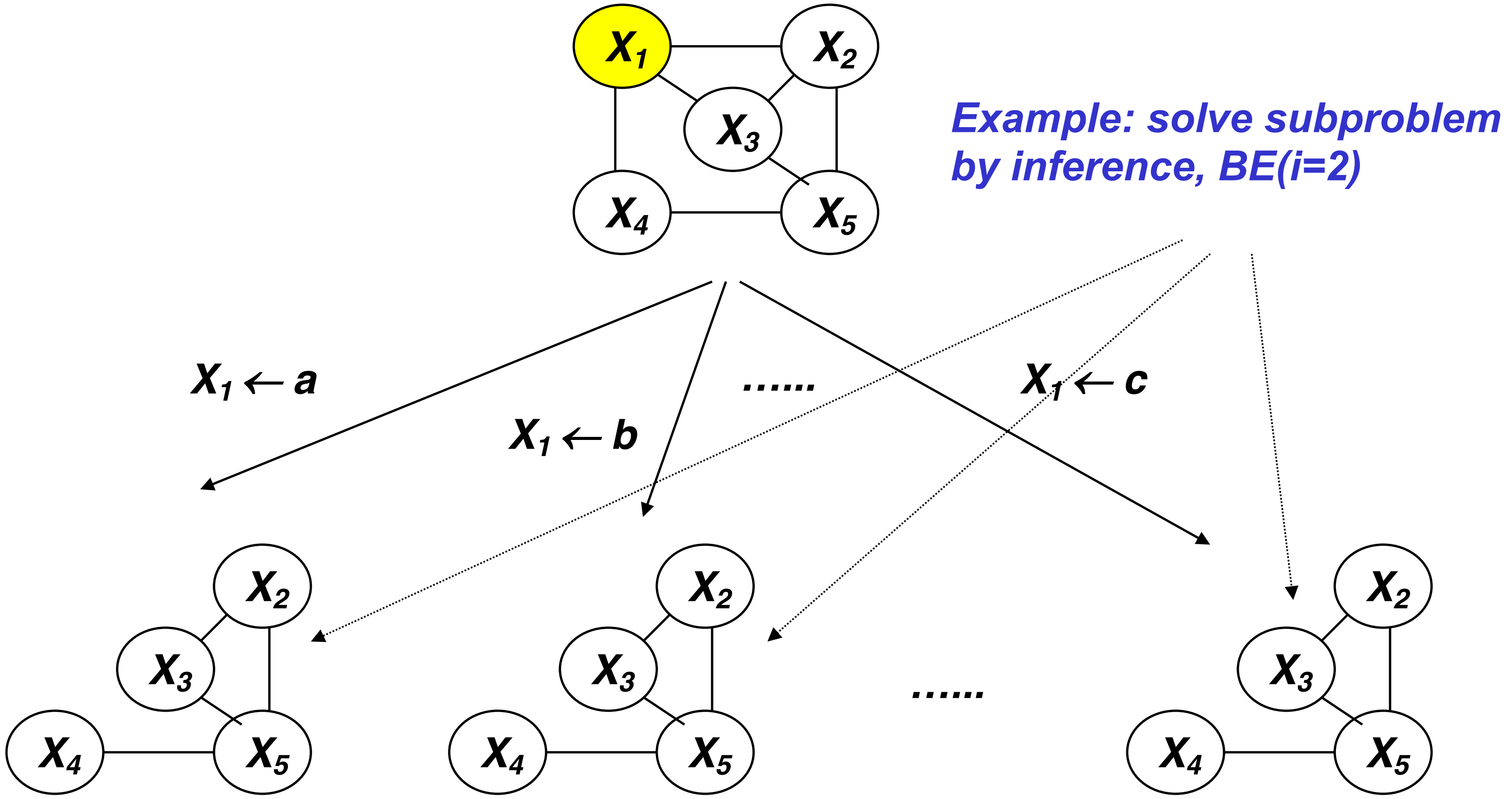


Search Basic Step:
Variable Branching by Conditioning

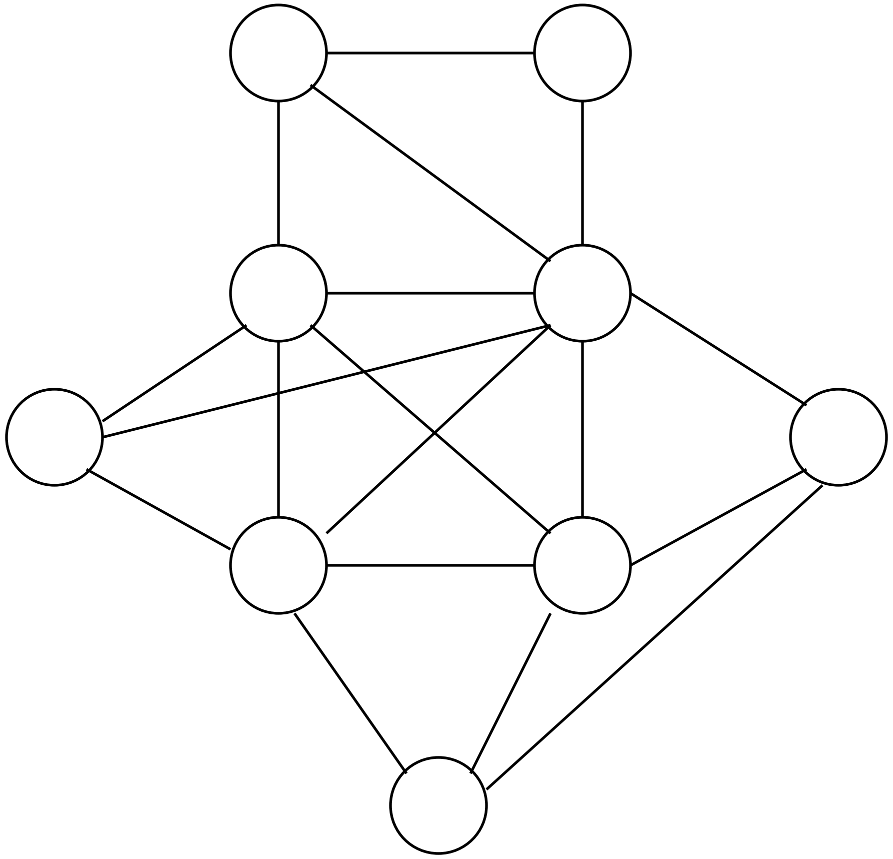


General principle:
Condition until tractable
Then solve sub-problems efficiently

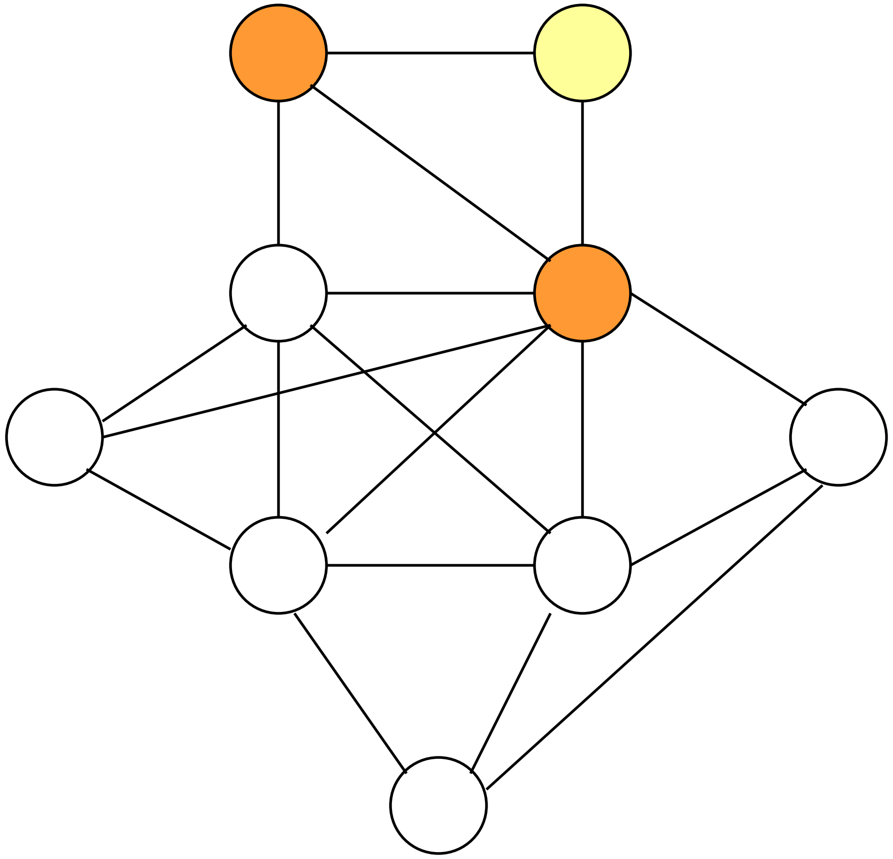
Search Basic Step:
Variable Branching by Conditioning



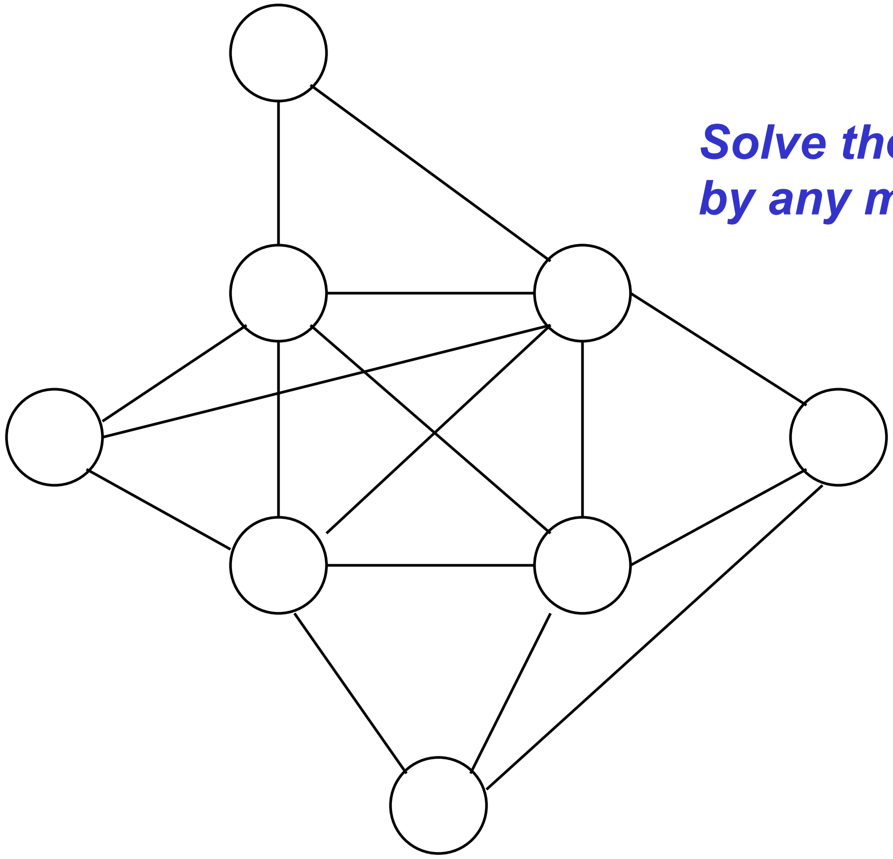
Eliminate First



Eliminate First



Eliminate First



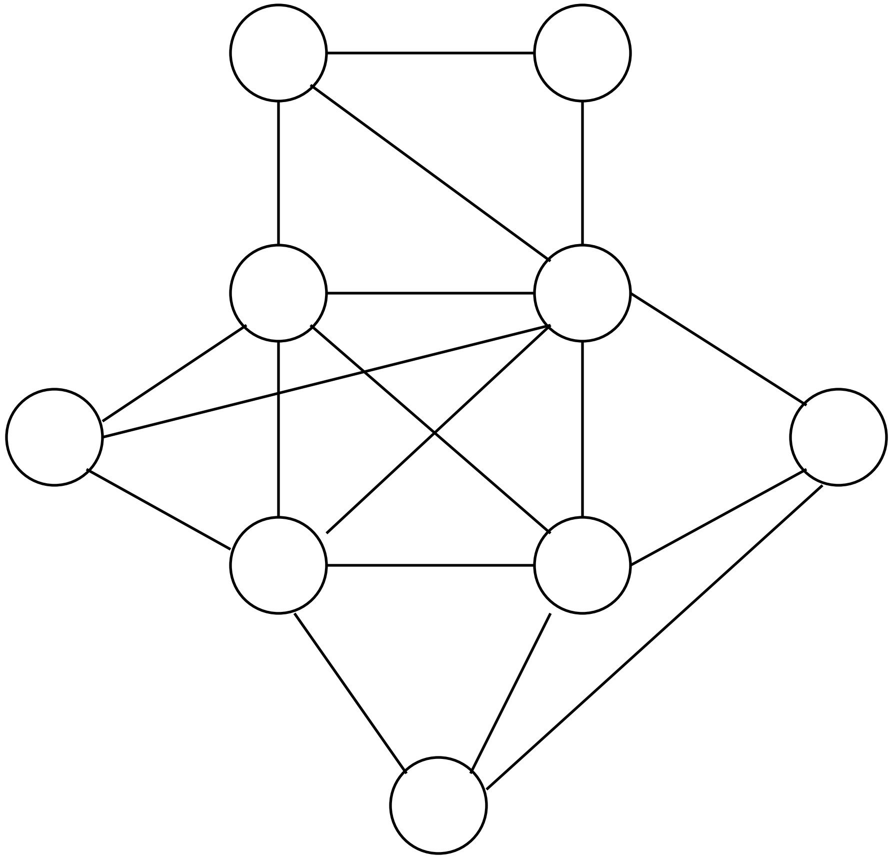
Solve the rest of the problem by any means

Hybrids Variants

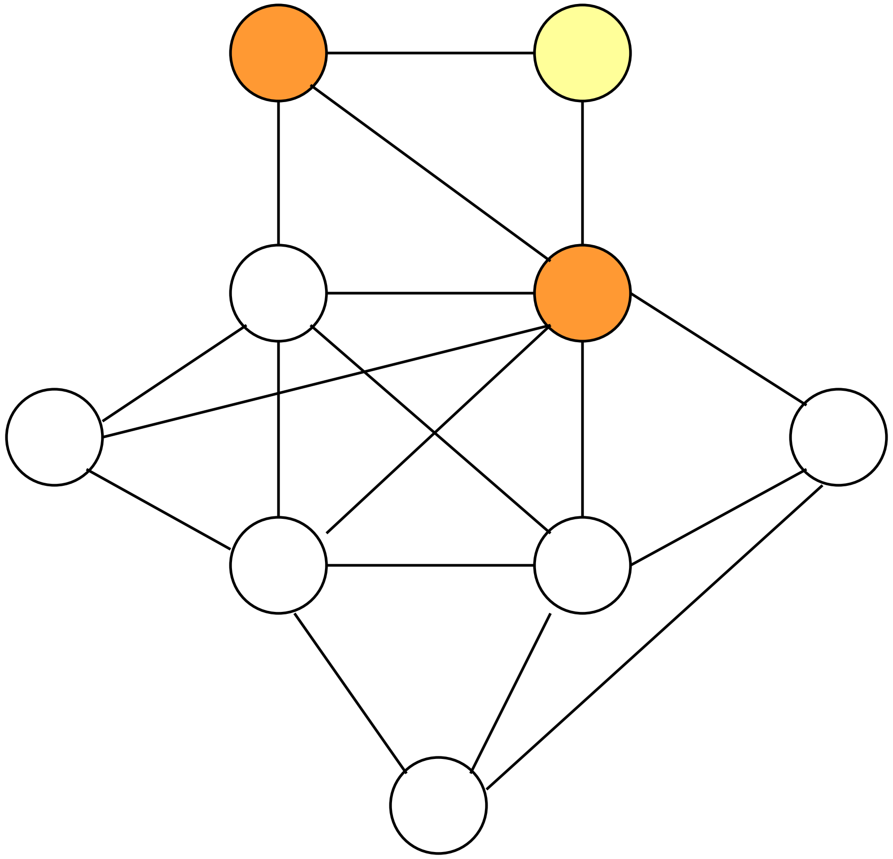
- Condition, condition, condition … and then only eliminate (w-cutset, cycle-cutset)
- Eliminate, eliminate, eliminate … and then only search
- Interleave conditioning and elimination (elim-cond(i), VE+C)

Interleaving Conditioning and Elimination

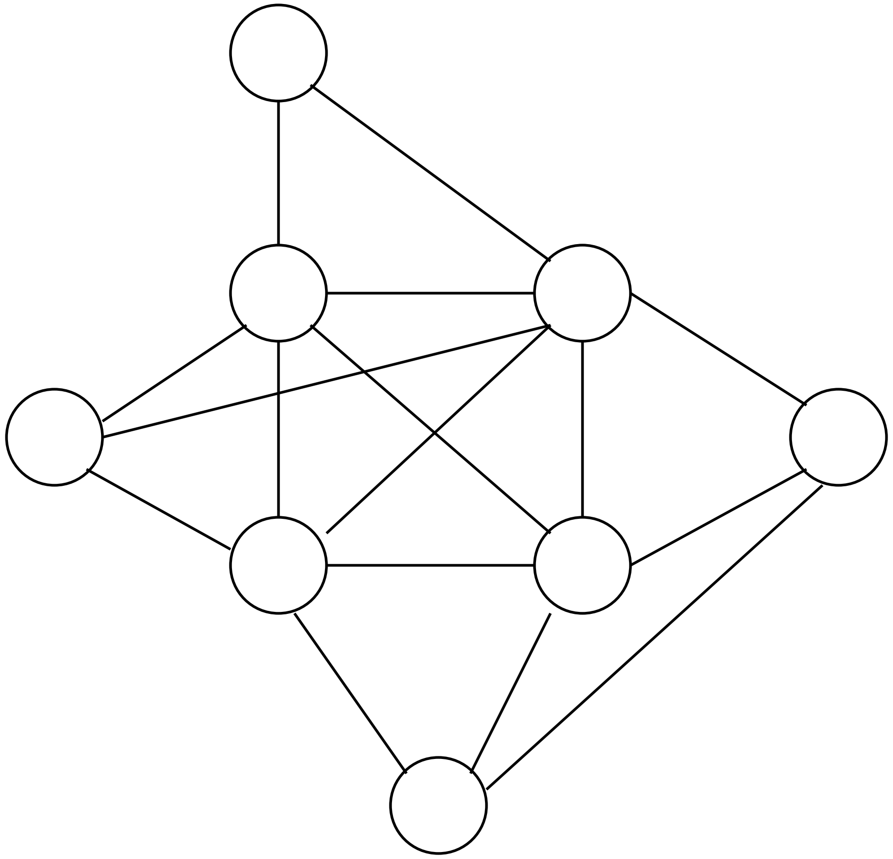
(Larrosa & Dechter, CP'02)



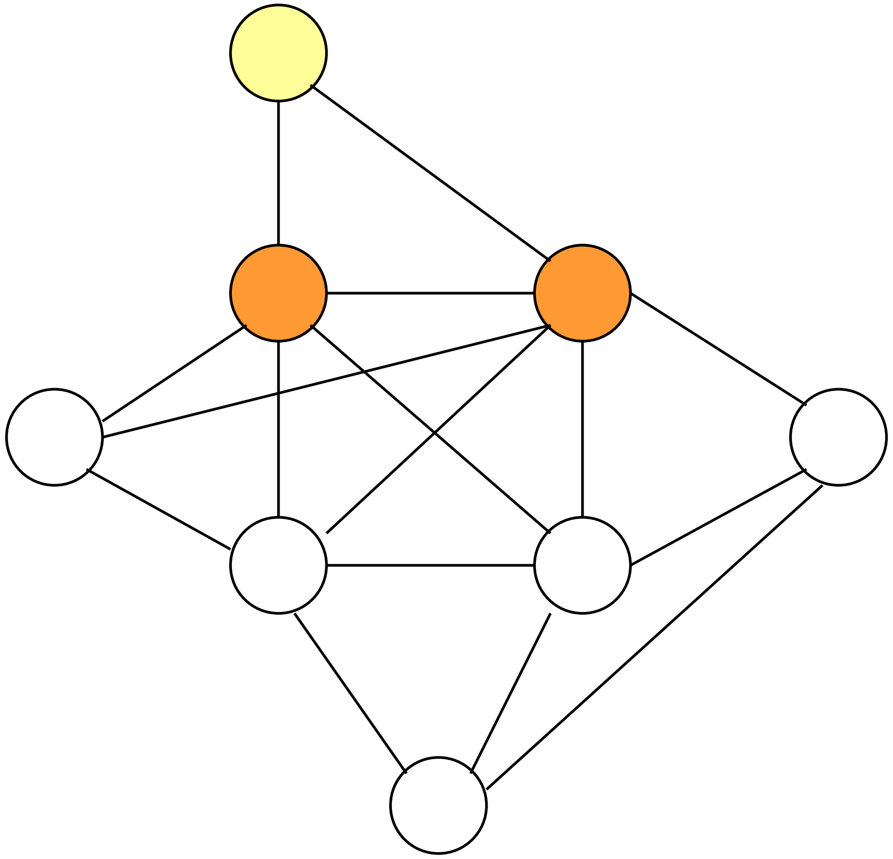
Interleaving Conditioning and Elimination



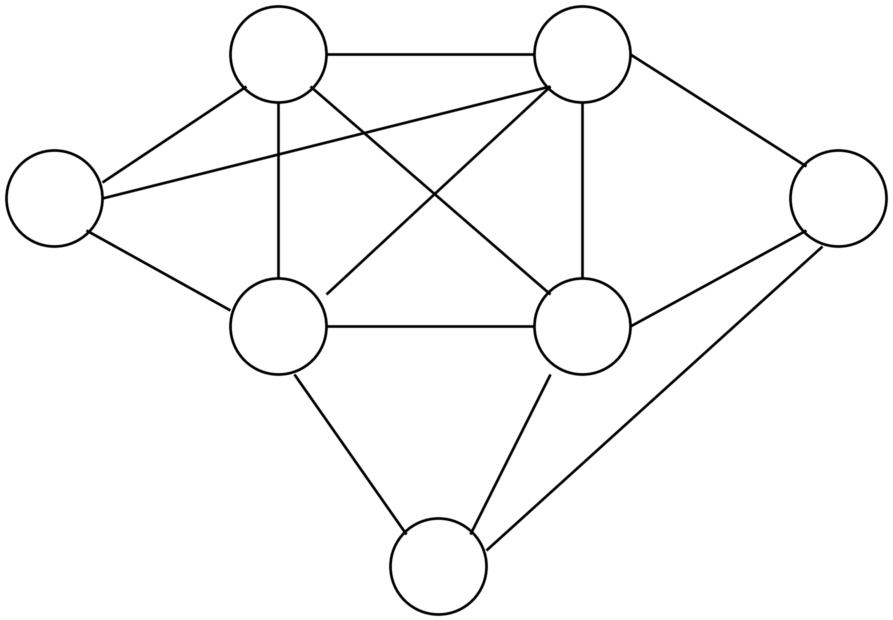
Interleaving Conditioning and Elimination



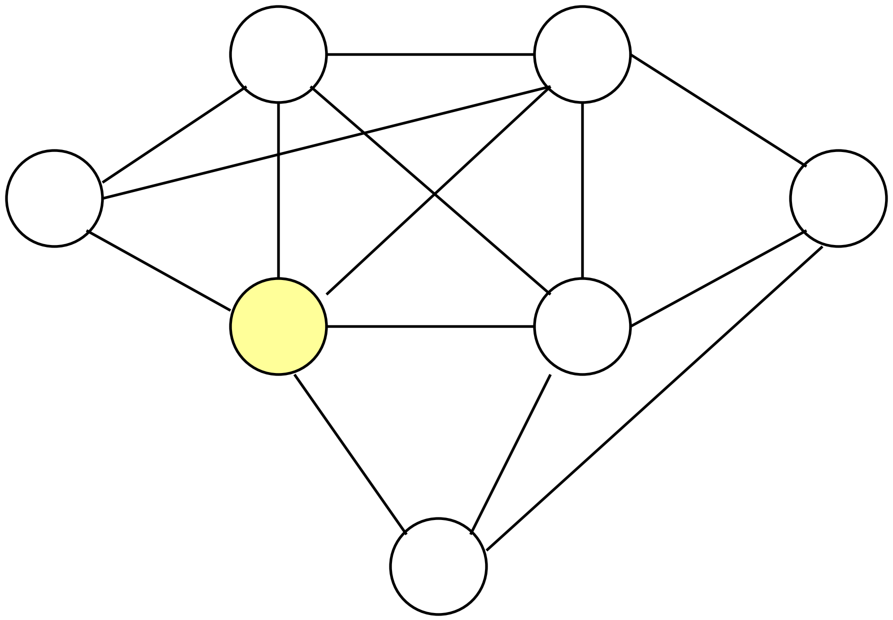
Interleaving Conditioning and Elimination



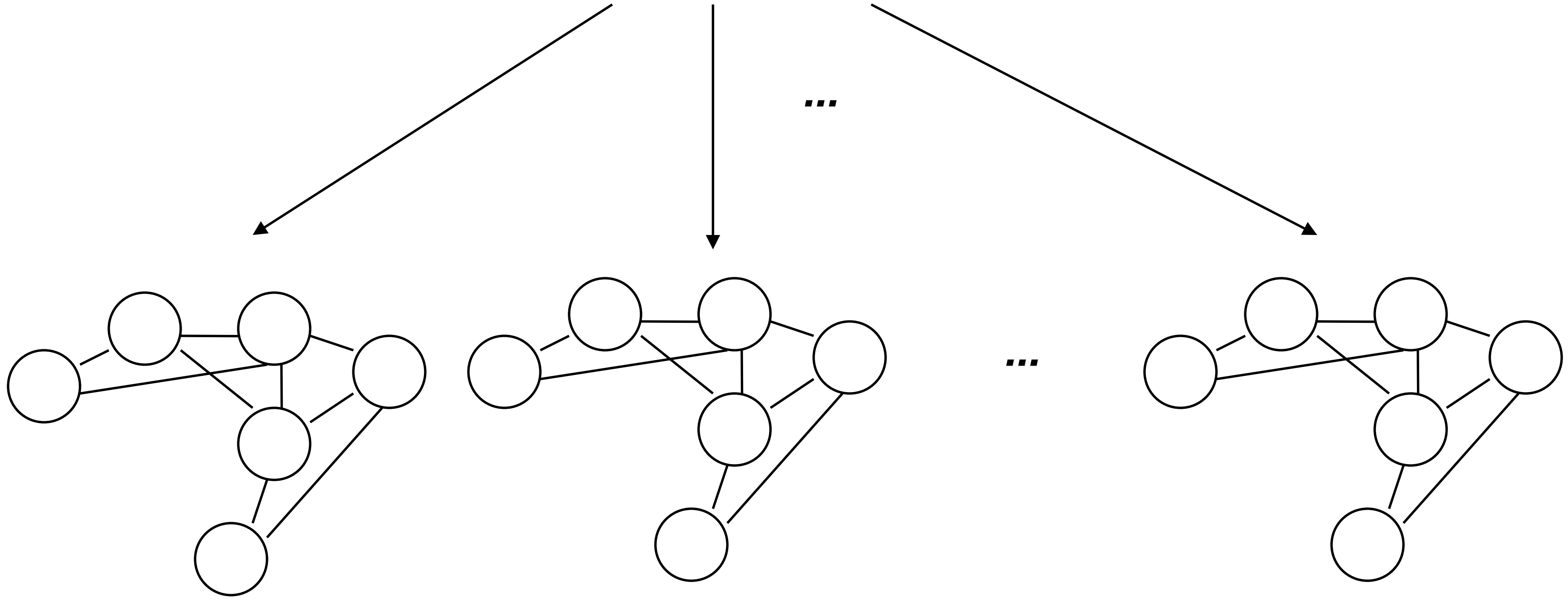
Interleaving Conditioning and Elimination



Interleaving Conditioning and Elimination



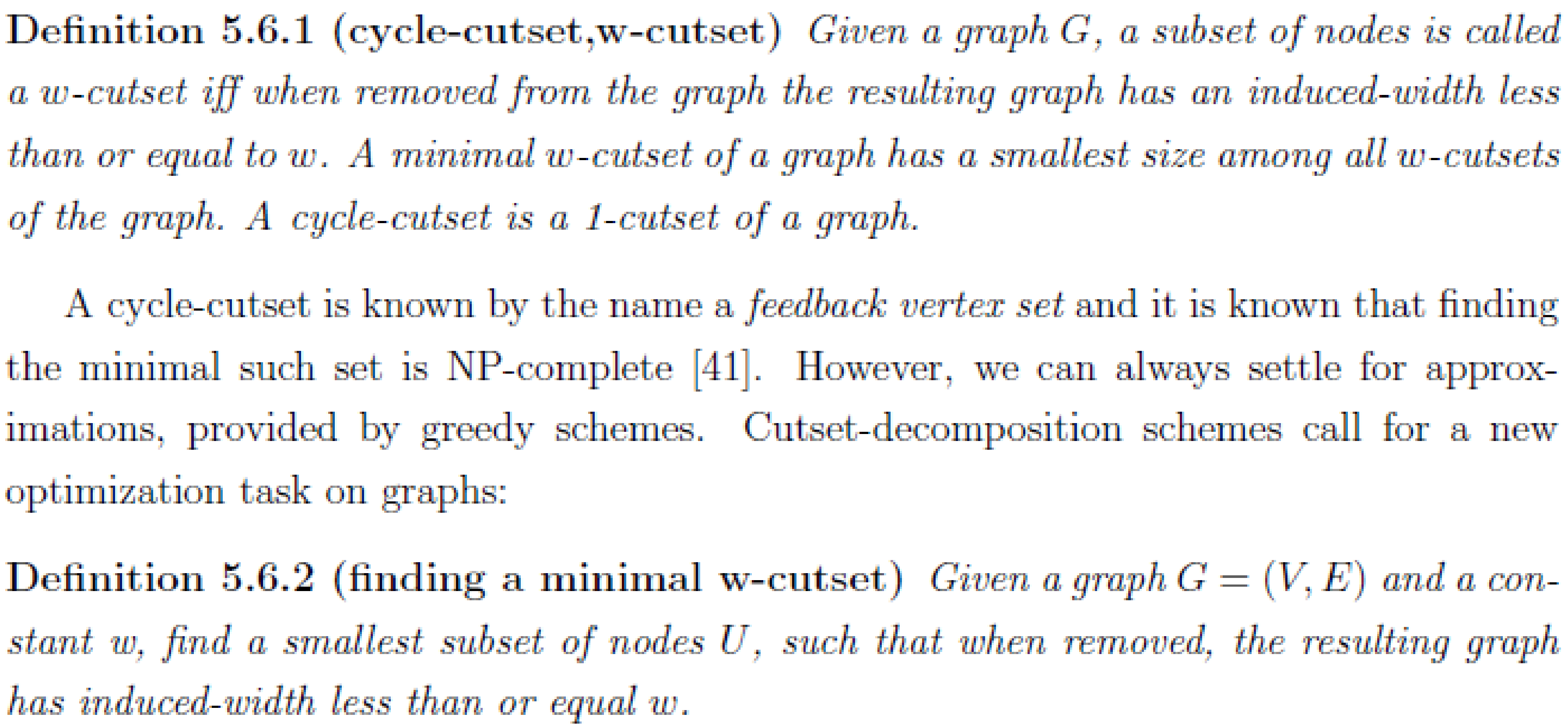
Interleaving Conditioning and Elimination



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. □