Inference for probabilistic networks (continued)

- Bucket elimination
  - Belief-updating, P(e), partition function
  - Marginals, probability of evidence
  - The impact of evidence
  - for MPE (→ MAP)
  - for MAP (→ Marginal Map)

- Induced-Width (Dechter 3.4,3.5)
The Impact of Evidence?

Algorithm \textit{BE-bel}

\[ P(A \mid E = 0) = \alpha \sum_{E=0,D,C,B} P(A) \cdot P(B \mid A) \cdot P(C \mid A) \cdot P(D \mid A,B) \cdot P(E \mid B,C) \]

\[ \sum \prod_b \]

Elimination operator

\begin{align*}
\text{bucket } B: & \quad P(b \mid a) \quad P(d \mid b,a) \quad P(e \mid b,c) \\
\text{bucket } C: & \quad P(c \mid a) \quad \lambda^B(a, d, c, e) \\
\text{bucket } D: & \quad \lambda^C(a, d, e) \\
\text{bucket } E: & \quad e=0 \quad \lambda^D(a, e) \\
\text{bucket } A: & \quad P(a) \quad \lambda^E(a) \quad P(e=0) \\
\end{align*}

\(B=1\)

\(W^*=4\)

"induced width" (max clique size)
\[ \text{MPE} = \max_{\bar{x}} P(\bar{x}) \]

\[ \sum \text{ is replaced by } \max : \]
\[ \text{MPE} = \max_{a,e,d,c,b} P(a)P(c \mid a)P(b \mid a)P(d \mid a,b)P(e \mid b,c) \]

\[ \max_b \prod \]

**Elimination operator**

**bucket B:** \[ P(b \mid a) \quad P(d \mid b,a) \quad P(e \mid b,c) \]

**bucket C:** \[ P(c \mid a) \quad h^B(a, d, c, e) \]

**bucket D:** \[ h^C(a, d, e) \]

**bucket E:** \[ e=0 \quad h^D(a, e) \]

**bucket A:** \[ P(a) \quad h^E(a) \]

\[ \\text{MPE} \]

\[ W^*=4 \]

"induced width" (max clique size)
Generating the MPE-tuple

1. \( a' = \text{arg max}_a P(a) \cdot h^E(a) \)

2. \( e' = 0 \)

3. \( d' = \text{arg max}_d h^C(a', d, e') \)

4. \( c' = \text{arg max}_c P(c | a') \times h^B(a', d', c, e') \)

5. \( b' = \text{arg max}_b P(b | a') \times P(d' | b, a') \times P(e' | b, c') \)

\[ \begin{align*}
B: & \quad P(b|a) \quad P(d|b,a) \quad P(e|b,c) \\
C: & \quad P(c|a) \quad h^B(a,d,c,e) \\
D: & \quad h^C(a,d,e) \\
E: & \quad e=0 \quad h^D(a,e) \\
A: & \quad P(a) \quad h^E(a) \\
\end{align*} \]

Return \((a', b', c', d', e')\)
Inference for probabilistic networks

- **Bucket elimination**
  - Belief-updating, \( P(e) \), partition function
  - Marginals, probability of evidence
  - The impact of evidence
  - for MPE (\( \rightarrow \) MAP)
  - for MAP (\( \rightarrow \) Marginal Map)

- **Induced-Width** (Dechter 3.4, 3.5)
Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)

- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]
Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)
- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]
Finding a Small Induced-Width

- NP-complete
- A tree has induced-width of ?
- Greedy algorithms:
  - Min width
  - Min induced-width
  - Max-cardinality and chordal graphs
  - Fill-in (thought as the best)

- Anytime algorithms
  - Search-based [Gogate & Dechter 2003]
  - Stochastic (CVO) [Kask, Gelfand & Dechter 2010]
Min-width Ordering

MIN-WIDTH (MW)

input: a graph $G = (V, E)$, $V = \{v_1, ..., v_n\}$
output: A min-width ordering of the nodes $d = (v_1, ..., v_n)$.

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

**Proposition:** algorithm min-width finds a min-width ordering of a graph

**What is the Complexity of MW?**

$O(e)$
Greedy Orderings Heuristics

- **Min-induced-width**
  - From last to first, pick a node with smallest width, then connect parent and remove

- **Min-Fill**
  - From last to first, pick a node with smallest fill-edges

*Complexity? \( O(n^3) \)
Min-Fill Heuristic

Select the variable that creates the fewest “fill-in” edges

Eliminate B next?
Connect neighbors
“Fill-in” = 3:
(A,D), (C,E), (D,E)

Eliminate E next?
Neighbors already connected
“Fill-in” = 0
Example

(a)
Different Induced-Graphs

(a) A
(b) B
(c) C
(d) D
Which Greedy Algorithm is Best?

- Min-Fill, prefers a node who adds the least number of fill-in arcs.

- Empirically, fill-in is the best among the greedy algorithms (MW, MIW, MF, MC)

- Complexity of greedy orderings?
  - MW is $O(e)$, MIW: $O(n^3)$, MF $O(n^3)$, MC is $O(e+n)$ (MC: read on your own)
Inference for probabilistic networks

- **Bucket elimination (Dechter chapter 4)**
  - Belief-updating, \( P(e) \), partition function
  - Marginals, probability of evidence
  - The impact of evidence
  - for MPE (\( \rightarrow \) MAP)
  - for MAP (\( \rightarrow \) Marginal Map)
  - Influence diagrams ?

- **Induced-Width (Dechter, Chapter 3.4)**
### Marginal Map

<table>
<thead>
<tr>
<th>Inference Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max-Inference</td>
<td>$f(x^*) = \max_x \prod_{\alpha} f_\alpha(x_\alpha)$</td>
</tr>
<tr>
<td>Sum-Inference</td>
<td>$Z = \sum_x \prod_{\alpha} f_\alpha(x_\alpha)$</td>
</tr>
<tr>
<td>Mixed-Inference</td>
<td>$f(x^*<em>M) = \max</em>{x_M} \sum_{x_S} \prod_{\alpha} f_\alpha(x_\alpha)$</td>
</tr>
</tbody>
</table>

- **NP-hard**: exponentially many terms
Example for MMAP Applications

- Haplotype in Family pedigrees
- Coding networks
- Probabilistic planning
- Diagnosis
Bucket Elimination for MMAP

\[ \mathbf{x}_M = \{A, D, E\} \]
\[ \mathbf{x}_S = \{B, C\} \]
\[ \max_{\mathbf{x}_M} \sum_{\mathbf{x}_S} P(\mathbf{X}) \]

MAP* is the marginal MAP value

\[ \lambda(A, C, D, E) \]

\[ \lambda^E(A) \]

\[ \max_E \]

\[ \lambda^D(A, E) \]

\[ \max_D \]

\[ \lambda^C(A, D, E) f(A, C) f(C, E) \]

\[ \sum_C \]

\[ \lambda^B(A, C, D, E) f(A, C) f(B, D) f(B, E) \]

\[ \sum_B \]
Why is MMAP harder?

In practice, constrained induced is much larger!

\[ X_M = \{A, D, E\} \]
\[ X_S = \{B, C\} \]

(Park & Darwiche, 2003)
(Yuan & Hansen, 2009)
Inference for probabilistic networks

- Bucket elimination (Dechter chapter 4)
  - Belief-updating, $P(e)$, partition function
  - Marginals, probability of evidence
  - The impact of evidence
    - for MPE ($\rightarrow$ MAP)
    - for MAP ($\rightarrow$ Marginal Map)
- Induced-Width (Dechter, Chapter 3.4)
- Mixed networks
- Influence diagrams?
Ex: “oil wildcatter”

- Influence diagram:

- Three actions: test, drill, sales policy
- Chance variables:
  \[
  P(\text{oil}) \quad P(\text{seismic}|\text{oil}) \quad P(\text{result} | \text{seismic}, \text{test}) \quad P(\text{produced} | \text{oil}, \text{drill}) \quad P(\text{market})
  \]
- Utilities capture costs of actions, rewards of sale
  Oil sales - Test cost - Drill cost - Sales cost
Influence diagram $ID = (X,D,P,R)$.

**Chance variables** $X = X_1, \ldots, X_n$ over domains.

**Decision variables** $D = D_1, \ldots, D_m$

**CPT’s for chance variables** $P_i = P(X_i \mid pa_i), i = 1 \ldots n$

**Reward components** $R = \{r_1, \ldots, r_j\}$

**Utility function** $u = \sum_i r_i$
Common examples

- Markov decision process
  - Markov chain state sequence
  - Actions "di" influence state transition
  - Rewards based on action, new state
  - Temporally homogeneous

- Partially observable MDP
  - Hidden Markov chain state sequence
  - Generate observations
  - Actions based on observations
**Influence Diagrams (continue)**

A decision rule for $D_i$ is a mapping: $\delta_i : \Omega p a_{D_i} \rightarrow \Omega_{D_i}$

where $\Omega_S$ is the cross product of domains in $S$.

A policy is a list of decision rules $\Delta = (\delta_1, \ldots, \delta_m)$

**Task:** Find an optimal policy that maximizes the expected utility.

$$E = \max_{\Delta=(\delta_1,\ldots,\delta_m)} \sum_{x=(x_1,\ldots,x_n)} \prod_i P_i(x)u(x)$$
Definition 2.2 Graphical model. A graphical model $\mathcal{M}$ is a 4-tuple, $\mathcal{M} = (X, D, F, \otimes)$, where:

1. $X = \{X_1, \ldots, X_n\}$ is a finite set of variables;
2. $D = \{D_1, \ldots, D_n\}$ is the set of their respective finite domains of values;
3. $F = \{f_1, \ldots, f_r\}$ is a set of positive real-valued discrete functions, defined over scopes of variables $S = \{S_1, \ldots, S_r\}$, where $S_i \subseteq X$. They are called local functions.
4. $\otimes$ is a combination operator (e.g., $\otimes \in \{\prod, \sum, \Join\}$ (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 $X$ which is the combination of all its functions: $\otimes_{i=1}^r f_i$. 
General Bucket Elimination

Algorithm General bucket elimination (GBE)

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

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 $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all the functions whose highest variable in their scope is $X_i$. An input function 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 $\psi_p, \lambda_1, \lambda_2, \ldots, \lambda_j$ 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)
     $\lambda_p \leftarrow \downarrow_{\mathcal{S}_p} \psi_p \otimes (\otimes_{i=1}^j \lambda_i)$ and add $\lambda_p$ to the largest-index variable in $\text{scope}(\lambda_p)$.

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 $d$. 
Causal and Probabilistic Reasoning

Slides Set 6:
Exact Inference Algorithms
Tree-Decomposition Schemes

Rina Dechter

(Dechter chapter 5, Darwiche chapter 6-7)
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?

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

\[ \text{bel}(d) = \alpha \sum_{a,b} P(d|a,b) \cdot \pi_{B \rightarrow D}(a,b). \]
BTE: Allows Messages Both Ways

Initial buckets + messages

$\begin{align*}
P(F) &= \sum_{b,c} P(F|b,c)\pi_{C\rightarrow F}(b,c)\lambda_{G\rightarrow F}(F) \\
P(D) &= \sum_{a,b} P(D|a,b)\pi_{B\rightarrow D}(a,b)
\end{align*}$
**Theorem:** When BTE terminates the product of functions in each bucket is the beliefs of the variables joint with the evidence.

\[
\text{elim}(i,j) = \text{scope}(B_i) - \text{scope}(B_j)
\]
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 $\mathcal{M} = \langle X, D, F, \Pi, \Sigma \rangle$, ordering $d$. $X = \{X_1, \ldots, X_n\}$ and $F = \{f_1, \ldots, f_r\}$, $E = e$. 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. 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
     $$\lambda_{i \rightarrow j} \leftarrow \sum_{\text{elim}(i,j)} \psi_i \cdot (\prod_{k \neq j} \lambda_{k \rightarrow i})$$
3. **Output:** augmented buckets $B'_1, \ldots, B'_n$, where each $B'_i$ contains the original bucket functions and the $\lambda$ messages it received.
Computing marginal beliefs

Input: a bucket tree processed by BTE with augmented buckets: $B_{t_1}, \ldots, B_{t_n}$

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

\[
\begin{align*}
\text{bel}(B_i) & \leftarrow \alpha \cdot \prod_{f \in B_i} f \\
\text{bel}(X_i) & \leftarrow \alpha \cdot \sum_{B_i \setminus \{X_i\}} \prod_{f \in B_i} f \\
P(\text{evidence}) & \leftarrow \sum_{B_i} \prod_{f \in B_i} f
\end{align*}
\]

Figure 5.4: Query answering.
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 $M = \langle X, D, F, \Pi, \Sigma \rangle$, $r$ be the number of functions in $F$ and $k$ be the maximum domain size. The time complexity of BTE is $O(r \cdot \text{deg} \cdot k^{w^*(d)+1})$, where $\text{deg}$ 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*
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.
Message Passing on a Tree Decomposition

Cluster(u) = ψ(u) ∪ \{m_{x_1 \to u}, m_{x_1 \to u}, m_{x_2 \to u}, \ldots m_{x_n \to u}\}

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

\[ m_{u \to v} = \sum_{elim(u,v)} \psi(u) \prod_{r \in neighbor(u), r \neq v} \{m_{r \to u}\} \]

For max-product
Just replace \( \sum \) With max.
Messages can propagate both ways and we get beliefs for each variable

- Propagation in Both Directions

\[
\begin{align*}
\mathbb{P}(X) & \quad \mathbb{P}(Y|X) \\
\mathbb{P}(Z|X) & \quad P(T|Y) \\
& \quad P(R|Y) \\
& \quad P(L|Z) \\
& \quad P(M|Z)
\end{align*}
\]
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)
The Idea of Cutset-Conditioning

Figure 7.1: An instantiated variable cuts its own cycles.
Conditioning - the Probability Tree

\[
P(D = 1, G = 0) = \sum_a P(a) \sum_c P(c|a) \sum_b P(b|a) \sum_f P(f|b,c) P(d = 1|b,a) P(g = 0|f)
\]

**Complexity of conditioning:** exponential time, linear space
Cycle-Cutset Conditioning

Cycle cutset = \{A, B, C\}

1-cutset = \{A, B, C\}, size 3
Search Over the Cutset (cont)

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

Graph Coloring problem

2-cutset = \{A,B\}, size = 2
The Impact of Observations

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
The Cycle-Cutset Scheme: Condition Until Treeness

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

(a) [Graph showing cycle-cutset]
(b) [Graph showing i-cutset]
(c) [Graph showing tree and cutset parts]
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 BE (belief propagation.)
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 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 Algorithms

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

Input: A belief network $\mathcal{B} = \langle \mathcal{X}, \mathcal{D}, \mathcal{G}, \mathcal{P} \rangle$, 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.)
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?
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 $w$. 
Theorem 7.7  \textit{Given graph }G, \textit{and denoting by }c_q^* \textit{ its minimal }q\textit{-cutset then,}

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

\textit{Proof.} Let's assume that we have a }q\textit{-cutset of size }c_q. \textit{Then if we remove it from the graph the result is a graph having a tree decomposition whose treewidth is bounded by }q. \textit{Let’s }T \textit{ be this decomposition where each cluster has size }q + 1 \textit{ or less. If we now take the }q\textit{-cutset variables and add them back to every cluster of }T, \textit{we will get a tree decomposition of the whole graph (exercise: show that) whose treewidth is }c_q + q. \textit{Therefore, we showed that for every }c_q\textit{-size }q\textit{-cutset, there is a tree decomposition whose treewidth is }c_a + q. \textit{In particular, for an optimal }q\textit{-cutset of size }c_q^*, \textit{we have that }w^*, \textit{the treewidth obeys, }w^* \leq c_q^* + q. \textit{This does not complete the proof because we only showed that for every }q, \textit{ }w^* \leq c_q^* + q. \textit{But, if we remove even a single node from a minimal }q\textit{-cutset whose size is }c_q^*, \textit{we get a }q + 1 \textit{ cutset by definition, whose size is }c_q^* - 1. \textit{Therefore, }c_{q+1}^* \leq c_q^* - 1. \textit{Adding }q \textit{ to both sides of the last inequality we get that for every }1 \leq q \leq w^*, \textit{ }q + c_q^* \geq q + 1 + c_{q+1}^*, \textit{which completes the proof.} \hfill \square
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 $C_1, C_2, ..., C_t$ by rank of the highest vertex in each clique.

4. Form the join tree by connecting each $C_i$ to a predecessor $C_j$ ($j < i$) sharing the highest number of vertices with $C_i$. 
EXAMPLE: Consider the graph in Figure 3.9a. One maximum cardinality ordering is \((A, B, C, D, E)\).

- Every vertex in this ordering has its preceding neighbors already connected, hence the graph is chordal and no edges need be added.
- The cliques are ranked \(C_1, C_2,\) and \(C_3\) as shown in Figure 3.9b.
- \(C_3 = \{C, E\}\) shares only vertex C with its predecessors \(C_2\) and \(C_1\), so either one can be chosen as the parent of \(C_3\).
- These two choices yield the join trees of Figures 3.9b and 3.9c.
- Now suppose we wish to assemble a join tree for the same graph with the edge \((B, C)\) missing.
- The ordering \((A, B, C, D, E)\) is still a maximum cardinality ordering, but now when we discover that the preceding neighbors of node \(D\) (i.e., \(B\) and \(C\)) are nonadjacent, we should fill in edge \((B, C)\).
- This renders the graph chordal, and the rest of the procedure yields the same join trees as in Figures 3.9b and 3.9c.
Examples of (Join)-Trees Construction