Exact Inference Algorithms
Tree-Decomposition Schemes

Rina Dechter

(Dechter1 chapter 5, Darwiche chapter 6-7)
**Bucket elimination**

**Algorithm BE-bel** (Dechter 1996)

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

\[
\sum \prod_{b} \quad \text{Elimination operator}
\]

- **bucket B:** \(P(b \mid a)\), \(P(d \mid b,a)\), \(P(e \mid b,c)\)
- **bucket C:** \(P(c \mid a)\)
- **bucket D:** \(\lambda^{C}(a,d,e)\)
- **bucket E:** \(e=0\), \(\lambda^{D}(a,e)\)
- **bucket A:** \(P(a)\)

\[
P(a \mid e=0) = \frac{P(a,e=0)}{P(e=0)}
\]

**W* = 4**

"induced width" (max clique size)
Irrelevant buckets for \textbf{BE-BEL}

Buckets that sum to 1 are irrelevant.

\textbf{Identification}: no evidence, no new functions.

\textbf{Recursive recognition}: \( \text{bel}(a|e) \)

- \( \text{bucket}(E) = P(e|b,c), \ e = 0 \)
- \( \text{bucket}(D) = P(d|a,b), \ldots \text{skipable bucket} \)
- \( \text{bucket}(C) = P(c|a) \)
- \( \text{bucket}(B) = P(b|a) \)
- \( \text{bucket}(A) = P(a) \)

\textbf{Complexity}: Use induced width in moral graph without irrelevant nodes, then update for evidence arcs.

\textit{Use the ancestral graph only}
Pruning Nodes: Example

Example of pruning irrelevant subnetworks

network structure

joint on $B, E$

joint on $B$
Pruning Edges: Example

Example of pruning edges due to evidence or conditioning

\[ A \quad B \quad \Theta_{B|A} \]
\[
\begin{array}{ccc}
\text{true} & \text{true} & .2 \\
\text{true} & \text{false} & .8 \\
\text{false} & \text{true} & .75 \\
\text{false} & \text{false} & .25 \\
\end{array}
\]

\[ A \quad C \quad \Theta_{C|A} \]
\[
\begin{array}{ccc}
\text{true} & \text{true} & .8 \\
\text{true} & \text{false} & .2 \\
\text{false} & \text{true} & .1 \\
\text{false} & \text{false} & .9 \\
\end{array}
\]

\[ B \quad D \quad \sum_C \Theta_{D|BC}^{C=false} \]
\[
\begin{array}{ccc}
\text{true} & \text{true} & .9 \\
\text{true} & \text{false} & .1 \\
\text{false} & \text{true} & 0 \\
\text{false} & \text{false} & 1 \\
\end{array}
\]

\[ E \quad \sum_C \Theta_{E|C}^{C=false} \]
\[
\begin{array}{ccc}
\text{true} & 0 \\
\text{false} & 1 \\
\end{array}
\]

Evidence e : \( C = \text{false} \)
Pruning Nodes and Edges: Example

$$
\begin{array}{|c|c|}
\hline
B & \Theta'_B = \sum_A \Theta_{B|A}^A_{\text{true}} \\
\hline
\text{true} & .2 \\
\text{false} & .8 \\
\hline
\end{array}
$$

$$
\begin{array}{|c|c|}
\hline
C & \Theta'_C = \sum_A \Theta_{C|A}^A_{\text{true}} \\
\hline
\text{true} & .8 \\
\text{false} & .2 \\
\hline
\end{array}
$$

![Diagram](image)

$$
\begin{array}{|c|c|c|}
\hline
B & D & \Theta'_{D|B} = \sum_C \Theta_{D|BC}^C_{\text{false}} \\
\hline
\text{true} & \text{true} & .9 \\
\text{true} & \text{false} & .1 \\
\text{false} & \text{true} & 0 \\
\text{false} & \text{false} & 1 \\
\hline
\end{array}
$$

Query $Q = \{D\}$ and $e: A = \text{true}, C = \text{false}$
Finding

\[ \text{MPE} = \max \ P(\overline{x}) \]

Algorithm \textit{BE-mpe} (Dechter 1996)

\[ \sum \text{ is replaced by } \max : \]

\[ \text{MPE} = \max_{a,e,d,c,b} P(a)P(c \mid a)P(b \mid a)P(d \mid a,b)P(e \mid b,c) \]

Diagram:

- **Bucket B:** \( P(b \mid a) P(d \mid b,a) P(e \mid b,c) \)
  - \( \max_b \prod \)
  - **Elimination operator**
- **Bucket C:** \( P(c \mid a) \)
  - **Bucket D:** \( h^B(a, d, c, e) \)
  - **Bucket E:** \( e = 0 \)
    - **Bucket A:** \( P(a) \)
      - **MPE**

\( W^* = 4 \)

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

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

2. \( e' = 0 \)

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

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

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

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

Return \((a', b', c', d', e')\)
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: $\otimes_{i=1}^{r} f_i$. 
Algorithm General bucket elimination (GBE)

Input: $\mathcal{M} = \langle X, D, F, \otimes \rangle$. $F = \{f_1, \ldots, f_n\}$ 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 $\text{bucket}_1, \ldots, \text{bucket}_n$, where $\text{bucket}_i$ 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 $\text{bucket}_p$, do

   - If (observed variable) $X_p = x_p$ appears in $\text{bucket}_p$, 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 $d$. 
## 1998 roadmap

### Outline; Road Map

<table>
<thead>
<tr>
<th>Tasks</th>
<th>CSP</th>
<th>SAT</th>
<th>Optimization</th>
<th>Belief updating</th>
<th>MPE, MAP, MEU</th>
<th>Solving linear equalities/inequalities</th>
</tr>
</thead>
<tbody>
<tr>
<td>elimination</td>
<td>adaptive consistency join-tree</td>
<td>directional resolution</td>
<td>dynamic programming</td>
<td>join-tree, VE, SPI, elim-bel</td>
<td>join-tree, elim-mpe, elim-map</td>
<td>Gaussian/Fourier elimination</td>
</tr>
<tr>
<td>conditioning</td>
<td>backtracking search</td>
<td>backtracking (Davis-Putnam)</td>
<td>branch-and-bound, best-first search</td>
<td>branch-and-bound, best-first search</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elimination + conditioning</td>
<td>cycle-cutset forward checking</td>
<td>DCDR, BDR-DP</td>
<td>loop-cutset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>approximate elimination</td>
<td>i-consistency bounded (directional) resolution</td>
<td>mini-buckets</td>
<td>mini-buckets</td>
<td>mini-buckets</td>
<td></td>
<td></td>
</tr>
<tr>
<td>approximate conditioning</td>
<td>greedylocal search (GSAT)</td>
<td>GSAT</td>
<td>gradient descent</td>
<td>stochastic simulation</td>
<td>gradient descent</td>
<td></td>
</tr>
</tbody>
</table>
Belief Updating Example

**SUM-PROD operators**
**POLY-TREE structure**

\[ P(h,f,r,m,b) = P(h) P(f) P(m|h,f) P(r|f) P(b|m) \]

\[ P(F | B=1) = \] ?  
**Probability of evidence**

\[ P(B=1) = .19643 \]

\[ P(F=1|B=1) = .3725 \]

**Updated belief**
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
First, observe the BE operates on a tree.

Second, What if we want the marginal on D?

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

Initial buckets + messages

Output buckets

\[ P(F) = \sum_{b,c} P(F|b,c) \pi_{C \rightarrow F}(b,c) \lambda_{G \rightarrow F}(F) \]

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

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

\[
\text{elim}(i,j) = \text{scope}(B_i) - \text{scope}(B_j)
\]

---

**Algorithm bucket-tree elimination (BTE)**

Input: A problem \( \mathcal{M} = \langle X, D, F, \prod, \sum \rangle \), ordering \( d \).

\( X = \{X_1, \ldots, X_n\} \) and \( F = \{f_1, \ldots, f_r\} \)

Evidence \( E = e \).

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

1. **Pre-processing:** Partition functions to the ordered buckets as usual and generate the bucket tree.
2. **Top-down phase:** \( \lambda \) messages (BE) do
   
   for \( i = n \) to 1, in reverse order of \( d \) process bucket \( B_i \):
   
   The message \( \lambda_{i \rightarrow j} \) from \( B_i \) to its parent \( B_j \), is:
   
   \[
   \lambda_{i \rightarrow j} = \sum_{\text{elim}(i,j)} \psi_i \cdot \prod_{k \in \text{child}(i)} \lambda_{k \rightarrow i}
   \]

   endfor
3. **Bottom-up phase:** \( \pi \) messages
   
   for \( j = 1 \) to \( n \), process bucket \( B_j \) do:
   
   \( B_j \) takes \( \pi_{k \rightarrow j} \) received from its parent \( B_k \), and computes a message \( \pi_{j \rightarrow i} \) for each child bucket \( B_i \) by
   
   \[
   \pi_{j \rightarrow i} = \sum_{\text{elim}(j,i)} \pi_{k \rightarrow j} \cdot \psi_j \cdot \prod_{r \neq i} \lambda_{r \rightarrow j}
   \]

   endfor
4. **Output:** augmented buckets \( B'_1, \ldots, B'_n \), 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
**Bucket-Tree Propagation (BTP)**

**Input:** A problem $\mathcal{M} = (X, D, F, \prod, \sum)$, 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. 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 $B'_1, ..., B'_n$, 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: $B_{t1}, ..., B_{tn}$

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

$\text{bel}(B_i) \leftarrow \alpha \cdot \prod_{f \in B_i} f$

$\text{bel}(X_i) \leftarrow \alpha \cdot \sum_{B_i \setminus \{X_i\}} \prod_{f \in B_i} f$

$P(\text{evidence}) \leftarrow \sum_{B_i} \prod_{f \in B_i} f$

Figure 5.4: Query answering.
Definition 5.4  Explicit function and explicit sub-model. Given a graphical model \( \mathcal{M} = \langle X, D, F, \prod \rangle \), and reasoning tasks defined by marginalization \( \sum \) and given a subset of variables \( Y, Y \subseteq X \), we define \( \mathcal{M}_Y \), the explicit function of \( \mathcal{M} \) over \( Y \):

\[
\mathcal{M}_Y = \sum_{X-Y} \prod_{f \in F} f,
\]

(5.4)

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 \( F \). We say that \((Y, F_Y)\) is an explicit submodel of \( \mathcal{M} \) iff

\[
\prod_{f \in F_Y} f = \mathcal{M}_Y
\]

(5.5)
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 \( M = \langle X, D, F, \prod, \sum \rangle \), \( r \) be the number of functions in \( F \) and \( k \) be the maximum domain size. The time complexity of BTE is \( O(r \cdot \deg \cdot k^{w^*(d)+1}) \), 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
- 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: 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.

- **Ayclic network**: when the dual graph is a tree or has a join-tree

- Tree-clustering converts a network into an acyclic one.
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.

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.
**Definition 5.11  Connectedness, join-trees.** Given a dual graph of a graphical model \( M \), 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
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 $\chi$ and $\psi$ are labeling functions, associating with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq P$ satisfying:

1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $\text{scope}(p_i) \subseteq \chi(v)$

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

**Separator-width:** maximum intersection between adjacent nodes

**Eliminator:** $\text{elim}(u, v) = x(u) - x(v)$
A tree decomposition for a graphical model < X,D,P > is a triple < T, χ,ψ >, where T = (V,E) is a tree and χ and ψ are labeling functions, associating with each vertex v ∈ V two sets, χ(v) ⊆ X and ψ(v) ⊆ P satisfying:

1. For each function p_i ∈ P there is exactly one vertex such that p_i ∈ ψ(v) and scope(p_i) ⊆ χ(v)

2. For each variable X_i ∈ X the set {v ∈ V|X_i ∈ χ(v)} forms a connected subtree (running intersection property)

---

**Tree Decompositions**

**Connectedness, or Running intersection property**
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.

\[(A)\]

\[
\begin{align*}
G,F \\
F, B, C \\
A, B, C \\
\end{align*}
\]

\[
\begin{align*}
F \\
B, C \\
A, B \\
\end{align*}
\]

\[
\begin{align*}
A, B, C \\
B, A \\
A \\
\end{align*}
\]

\[(B)\]  

\[
\begin{align*}
G,F \\
F, B, C \\
A, B, C \\
\end{align*}
\]

\[
\begin{align*}
F \\
B, C \\
A, B \\
\end{align*}
\]

\[
\begin{align*}
A, B, C \\
B, A \\
A \\
\end{align*}
\]

\[
\begin{align*}
D, B, A \\
F, B, C \\
A, B, C \\
D, B, A \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C \\
D, B, A \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]

\[
\begin{align*}
G, F \\
F \\
A, B, C, D, F \\
\end{align*}
\]
Message Passing on a Tree Decomposition

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

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

\[ m_{u→v} = \sum_{\text{elim}(u,v)} \prod f \in \text{cluster}(u)-\{m_{v→u}\} \ f \]

\[ m_{u→v} = \sum_{\text{elim}(u,v)} \psi(u) \prod r \in \text{neighbor}(u), r \neq v \ \{m_{r→u}\} \]
Cluster-Tree Elimination (CTE)

Input: A tree decomposition \( < T, \chi, \psi > \) for a problem \( M = < X, D, F, \prod, \Sigma > \),
\( X = \{ X_1, ..., X_n \}, F = \{ f_1, ..., f_r \} \). 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 \( m_{u \rightarrow v} \) 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:
   \[
   m_{u \rightarrow v} \leftarrow \sum_{\chi(u) - \text{sep}(u,v)} \psi_u \cdot \prod_{r \in \text{neighbor}(u), r \neq v} m_{r \rightarrow u}
   \]
   endfor
   Note: functions whose scopes do not contain any separator variable do not need to be combined and can be directly passed on to the receiving vertex.
3. Return: The explicit tree \( < T, \chi, \bar{\psi} > \), where
   \( \bar{\psi}(v) \leftarrow \psi(v) \cup_{u \in \text{neighbor}(v)} \{ m_{u \rightarrow v} \} \)
   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:**
  \[ O(\text{deg} \times (n+N) \times k^{w^*+1}) \]

- **Space complexity:**
  \[ O(N \times k^{\text{sep}}) \]
  
  where:
  - \( \text{deg} \) = 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
  - \( \text{sep} \) = 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 $C_1, C_2, ..., C_t$ 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

- Trees Construction

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

- \( ABCE \)

- \( DEF \)

- \( BCDE \)

- \( BCD \)

- \( ABE \)

- \( ABC \)

- \( FD \)

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F
\end{array}
\]
Tree-Clustering and Message-Passing

Two join-trees

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

Find the errors
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
Message passing on a tree decomposition

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

\[
\text{cluster}(u) = \varphi(u) \cup \{ h(x_1, u), h(x_2, u), \ldots, h(x_n, u), h(v, u) \}
\]

Compute the message:

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

\[
\text{Elim}(u,v) = \text{cluster}(u) - \text{sep}(u,v)
\]
For max-product
Just replace $\Sigma$
With max.

Cluster($u$) = $\psi(u) \cup \{m_{x_1 \rightarrow u}, m_{x_1 \rightarrow u}, m_{x_2 \rightarrow u}, \ldots m_{x_n \rightarrow u}\}$

$$m_{u \rightarrow v} = \sigma_{elim(u,v)} \prod f \in \text{cluster}(u) - \{m_{v \rightarrow u}\} f$$

$$m_{u \rightarrow v} = \sum_{elim(u,v)} \psi(u) \prod r \in \text{neighbor}(u), r \neq v \{m_{r \rightarrow u}\}$$

Elim($u, v$) = cluster($u$)-sep($u, v$)
Cluster-Tree Elimination (CTE), or Join-Tree Message-passing

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
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 $X$. 
Pearl’s 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
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 (Dechter1, 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

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

69
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
\[ P(a, e = 0) = P(a) \sum_b P(b \mid a) \sum_c P(c \mid a) \sum_d P(d \mid a, b) \sum_{e=0} P(e \mid b, c) \]

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 exp in loop-cutset size and linear space. For each cutset we can do BP
\textbf{q-Cutset, Minimal}

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

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

\textbf{Example 7.4} Consider as another example the constraint 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.
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 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: $nk^{c+q+1}$
  - 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.)
VEC and ALT-VEC: Alternate conditioning and Elimination

- VEC (q-cutset-conditioning) 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

\[ X_1 \leftarrow a \quad \cdots \quad X_1 \leftarrow c \]

- \[ X_1 \leftarrow b \]
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

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?
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. Given graph $G$, and denoting by $c_q^*$ its minimal $q$-cutset then,

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

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 cluster 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 $c_a + a$. 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 $c_{q+1}^* - 1$. 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. \qed