Exact Inference Algorithms for Probabilistic Reasoning; BTE and CTE

COMPSCI 276, Spring 2017
Set 6: Rina Dechter

(Reading: Primary: Dechter chapter 5
Secondary: , Darwiche chapters 7,8)
Bucket elimination
Algorithm BE-bel (Dechter 1996)

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

\[ \sum \prod_b \rightarrow \text{Elimination operator} \]

bucket B: \[ P(b|a), P(d|b,a), P(e|b,c) \]

bucket C: \[ P(c|a), \lambda^B(a, d, c, e) \]

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

bucket E: \[ e=0, \lambda^D(a, e) \]

bucket A: \[ P(a), \lambda^E(a) \]

\[ P(e=0) \]

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

\[ P(a|e=0) \]

\[ W^* = 4 \]

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

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

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

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

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

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

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

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

bucket A: \[ P(a) \quad h^E_E(a) \quad \text{MPE} \quad \text{W}^* = 4 \quad \text{"induced width" (max clique size)} \]
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. $c' = \arg \max_c P(c|a') \times h^B(a', d', c, e')$
5. $b' = \arg \max_b P(b|a') \times \times P(d'|b, a') \times P(e'|b, c')$

B: $P(b|a)$ $P(d|b,a)$ $P(e|b,c)$
C: $P(c|a)$ $h^B(a,d,c,e)$
D: $h^C(a,d,e)$
E: $e=0$ $h^D(a,e)$
A: $P(a)$ $h^E(a)$

Return $(a', b', c', d', e')$
Probabilistic Inference Tasks

- **Belief updating:**
  \[ \text{BEL}(X_i) = P(X_i = x_i \mid \text{evidence}) \]

- **Finding most probable explanation (MPE)**
  \[ \bar{x}^* = \arg\max_{\bar{x}} P(\bar{x}, e) \]

- **Finding maximum a-posteriory hypothesis**
  \[ (a_1^*, ..., a_k^*) = \arg\max_{\bar{a}} \sum_{X/A} P(\bar{x}, e) \]

- **Finding maximum-expected-utility (MEU) decision**
  \[ (d_1^*, ..., d_k^*) = \arg\max_{\bar{d}} \sum_{X/D} P(\bar{x}, e)U(\bar{x}) \]
Markov Networks

Definition 2.23 **Markov networks.** A Markov network is a graphical model \( M = (X, D, H, \Pi) \) where \( H = \{\psi_1, \ldots, \psi_m\} \) is a set of potential functions where each potential \( \psi_i \) is a non-negative real-valued function defined over a scope of variables \( S = \{S_1, \ldots, S_m\} \). \( S_i \). The Markov network represents a global joint distribution over the variables \( X \) given by:

\[
P_M = \frac{1}{Z} \prod_{i=1}^{m} \psi_i \quad , \quad Z = \sum_{X} \prod_{i=1}^{m} \psi_i
\]

where the normalizing constant \( Z \) is called the partition function.

Everything is applicable to Markov networks as well
Definition 2.2  Graphical model. A graphical model $\mathcal{M}$ is a 4-tuple, $\mathcal{M} = (\mathbf{X}, \mathbf{D}, \mathbf{F}, \otimes)$, 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 $S = \{S_1, \ldots, S_r\}$, where $S_i \subseteq \mathbf{X}$. They are called local functions.
4. $\otimes$ is a combination operator (e.g., $\otimes \in \{\prod, \sum, \bowtie\}$ (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: $\otimes_{i=1}^r f_i$. 
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 marginalise)
     $\lambda_p \leftarrow \downarrow_{\mathcal{S}_p} \psi_p \otimes (\otimes_{i=1}^j \lambda_i)$ and add $\lambda_p$ to the largest-index variable in $scope(\lambda_p)$.

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</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></td>
<td>join-tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conditioning</td>
<td>backtracking</td>
<td>backtracking (Davis-Putnam)</td>
<td>branch-and-bound, best-first search</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>elimination + conditioning</td>
<td>cycle-cutset</td>
<td>DCDR, BDR-DP</td>
<td>loop-cutset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>forward checking</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>approximate elimination</td>
<td>i-consistency</td>
<td>bounded (directional)</td>
<td>mini-buckets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>resolution</td>
<td></td>
<td></td>
<td></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>
<tr>
<td></td>
<td>(GSAT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>approximate (elimination + conditioning)</td>
<td>GSAT + partial path-consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)
A Bayesian Network Processed by BE

Two orderings for BE

\[ \sum \Pi \]

Bucket G: \( P(G|F) \) \( G=1 \)

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

Bucket F: \( P(F|B,C) \) \( \lambda^D(B,A) \) \( \lambda^F(B,C) \)

Bucket B: \( P(B|A) \)

Bucket C: \( P(C|A) \) \( \lambda^B(A,C) \)

Bucket A: \( P(A) \) \( \lambda^C(A) \)

\[ \sum \Pi \]

Bucket G: \( P(G|F) \) \( G=1 \)

Bucket B: \( P(F|B,C) P(D|B,A) P(B|A) \)

Bucket C: \( P(C|A) \) \( \lambda^D(A,D,C,F) \)

Bucket D: \( \lambda^C(A,D,F) \)

Bucket F: \( \lambda^D(A,F) \) \( \lambda^G(F) \)

Bucket A: \( P(A) \) \( \lambda^F(A) \)

\[ P(G=1) \]

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

Complexity exponential in \( w^*(d) \)
From Bucket Elimination to Bucket-Tree Elimination

Observation 1: BE is a message propagation down a bucket-tree
From Bucket Elimination to Bucket-Tree Elimination

What If we want the marginal on B?

Bucket G: $P(G|F)$

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

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

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

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

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

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

$$P(B) = \sum P(B \mid A)P(A)\lambda_{C \rightarrow B}(B,C)\lambda_{D \rightarrow B}(A,B)$$
From Bucket Elimination to Bucket-Tree Elimination (BTE)

What if we want the marginal on D? Imagine combining B and A, D
\[ d = (\{A,D,B\},C,F,G) \]

Bucket G: \( P(G|F) \)

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

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

Bucket C: \( P(C|A) \)

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

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

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

This example generalize: We can compute the belief in each variable by a second message-passing along the bucket-tree.

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.
Idea of BTE

This example generalize: We can compute the belief in each variable by a second message-passing along the bucket-tree.
BTE: Allows Messages Both Ways

Initial buckets + messages
BTE: Allows Messages Both Ways

Initial buckets + messages

Output buckets
A Bucket-Tree of a Bayesian Network

- The bucket-tree:
  - Nodes are the buckets. Each has functions (assigned initially) and variables: itself+ induced-parents
    - There is an arc from $B_i$ to $B_j$ iff the function created at bucket $B_i$ is placed at bucket $B_j$
    - We have a separator and eliminator between two adjacent buckets
Bucket-Tree 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 (variables, functions) pairs.
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 \( X \).
Theorem: When BTE terminates, the product of functions in each bucket is the beliefs of the variables joint with the evidence.

Algorithm bucket-tree elimination (BTE)

Input: A problem $\mathcal{M} = \langle X, D, F, \Pi \rangle$, 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 all the $\pi$ and $\lambda$ functions received from neighbors in the bucket-tree.

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

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

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

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

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

   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

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

   endfor

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

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

Input: a bucket tree processed by BTE with augmented buckets: $Bt_1, ..., Bt_n$

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

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

Figure 5.4: Query answering.
Definition 5.4  Explicit function and explicit sub-model. Given a graphical model $\mathcal{M} = \langle X, D, F, \Pi \rangle$, 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)
Bucket-Tree Propagation (BTP)

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

Output: Explicit buckets. 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.
Properties of BTE

- **Theorem (correctness)** Theorem 6.1.4 Algorithm BTE when applied to a Bayesian or Markov network is sound. Namely, in each bucket we can exactly compute the exact joint function of every subset of variables and the evidence.

- *(follows from imapness of trees)*

- **Theorem (complexity of BTE) Theorem 5.6** Let $w^*(d)$ be the induced width of $G$ along ordering $d$, $r$ be the number of functions and $k$ the maximum domain size. BTE is $O(r \cdot \deg \cdot k^{(w^*(d)+1)})$ time, where deg is the maximum degree in the bucket-tree. BTE is $O(n \cdot k^{w^*(d)})$ space.
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 $M = \langle X, D, F, \prod, \Sigma \rangle$, $r$ 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 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.
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)
From Buckets to Clusters

(A) Allows time and space tradeoff

(B) Allows time and space tradeoff

(C) Allows time and space tradeoff
From a Bucket-Tree to a Cluster-Tree

- Merge none-maximal buckets into maximal clusters.
- Connect clusters into a tree: each cluster to one with which it shares a largest subset of variables.
- Separators are the intersection of variables on the arcs of the tree.
- The cluster-tree is an i-map.
Examples

Cluster Tree propagation (CTP) = BTP on any cluster-tree.
The General Message Passing
On a General Tree-Decomposition (CTP)

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

For max-product
Just replace \( \Sigma \)
With max.

\[ h_{u \rightarrow v} \text{ new notation for } h(u, v) \]

Compute the message :

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

\[ \text{elim}(u, v) = \text{cluster}(u) - \text{sep}(u, v) \]
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)
Acyclic Networks

- **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

- **Ayclic network (alternative characteristic):** A network is acyclic if it has a tree-decomposition where each node has a single original CPT.

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

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.

Also called the running intersection property

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 Dual Graph Having a Join-Tree is Acyclic

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
Tree-decompositions

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

1. For each function $p_i \in P$ there is exactly one vertex such that $p_i \in \psi(v)$ and $\text{scope}(p_i) \subseteq \chi(v)$
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 nodes in a node of Tree-decomposition − 1
Separator-width: maximum intersection between adjacent nodes
Eliminator: $\text{elim}(u,v) = x(u) - x(v)$
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.
Examples of Tree-Decompositions or Cluster-Trees

(A)

G,F

F, B, C

A, B, C

F, B, C

D, B, A

B, A

A

(B) Allows time and space tradeoff

G,F

F

F, B, C

D, B, A

A, B, C

A, B, C

A, B, C

A, B, C

A, B, C

A, B, C

A, B, C

A

(C)

G,F

F

G,F

F

G,F

F

G,F

F

A, B, C, D, F
Cluster-Tree Elimination

CLUSTER-TREE ELIMINATION (CTE)

Input: A tree decomposition \( < T, \chi, \psi > \) for a problem \( M = < X, D, F, \Pi, \Sigma > \), \( X = \{X_1, ..., X_n\} \), \( F = \{f_1, ..., f_r\} \). 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 \( v \), \( M_{\chi(v)} = \prod_{f \in \bar{\psi}(v)} f \)
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)
A graph is chordal if every cycle of length at least 4 has a chord.

Finding $w^*$ of chordal graph is easy using the max-cardinality ordering.

A graph along a max-cardinality order has no fill-in edges iff it is chordal.

Theorem: The maximal cliques of a chordal have a join-tree. Namely, there is a tree connecting the maximal cliques such that connectedness is obeyed.
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 $C_i$. 
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 \(C_1, C_2, \text{ and } C_3\) 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.
Join-tree Clustering (JTC)

**Input:** A graphical model $\mathcal{M} = (X, D, F, \prod)$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$. Its scopes $S = S_1, ..., S_r$ and its primal graph is $G = (X, E)$.

**Output:** A join-tree decomposition $< T, \chi, \psi >$ for $\mathcal{M}$

1. Select a variable ordering, $d = (X_1, ..., X_n)$.
2. **Triangulation** (create the induced graph along $d$ and call it $G^*$):
   
   ```
   for j = n to 1 by -1 do 
   \[ E \leftarrow E \cup \{(i, k) | i < j, \ k < j, \ (i, j) \in E, \ (k, j) \in E\} \]
   ```

3. **Create a join-tree of the induced graph** ($G^*, d$) as follows:
   a. Identify all maximal cliques in the chordal graph.
      Let $C = \{C_1, ..., C_t\}$ be all such cliques, where $C_i$ is the cluster of bucket $i$.
   b. Create a tree $T$ of cliques:
      Connect each $C_i$ to a $C_j$ ($j < i$) with whom it shares largest subset of variables.

4. Create $\psi_i$: Partition input function in cluster-node whose variables contain its scope.

5. Return a tree-decomposition $< T, \chi, \psi >$, where $T$ is generated in step 3,
   $\chi(i) = C_i$ and $\psi(i)$ is determined in step 4.

**Figure 5.12:** Join-tree clustering.
Examples of Generating Join-Trees

(a)

(b)
Examples of Generating Join-Trees

(a) [Graph with nodes A, B, C, D, E, F]

(b) [Graph with nodes A, B, C, D, E, F]

(c) [Graph with nodes A, B, C, D, E, F]

(d) [Graph with nodes A, B, C, E, D, E, B, C, D, E, D, E, B, C, E, A, B, C, B, A, C]
Tree-clustering and message-passing

Two join-trees

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

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

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

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

where

- $\text{deg}$ = the maximum degree of a node
- $n$ = number of variables (= number of CPTs)
- $N$ = number of nodes in the tree decomposition
- $d$ = the maximum domain size of a variable
- $w^*$ = the induced width, treewidth
- $\text{sep}$ = the separator size
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)
Tree Decomposition for Belief Updating
Tree Decomposition for belief updating

A B C
\( p(a), p(b|a), p(c|a,b) \)

BC

B C D F
\( p(d|b), p(f|c,d) \)

BF

B E F
\( p(e|b,f) \)

EF

E F G
\( p(g|e,f) \)
CTE: Cluster Tree Elimination

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

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

For each cluster $P(X|e)$ is computed, also $P(e)$
Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition
Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition
Figure 6.12: [Execution of CTE-BU]: a) A belief network; b) A join-tree decomposition; c) Execution of CTE-BU; no individual functions appear in this case (d) the explicit tree-decomposition
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)
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.

![Images of polytree and acyclic network diagrams](Image)

Figure 4.18. (a) A fragment of a polytree and (b) the parents and children of a typical node \( X \).
PolyTrees and Acyclic-Networks:

Two acyclic Bayesian Networks

\[
\begin{align*}
P(E) & \quad P(B) \\
P(R|E) & \quad P(A|B,E) \\
P(E) & \quad P(B|E) \\
P(R|E,A) & \quad P(A|B,E) \\
P(R|E) & \quad P(B) \\
P(A|B,E) & \quad P(C|A) \\
P(C|A) & \quad P(C|A,R) \\
P(R|E,A) & \quad P(A|B,E) \\
P(E) & \quad P(B) \\
P(E) & \quad P(B|E) \\
P(R|E,A) & \quad P(A|B,E) \\
P(R|E) & \quad P(B) \\
P(A|B,E) & \quad P(C|A) \\
P(C|A) & \quad P(C|A,R) \\
\end{align*}
\]
BTE = Belief Propagation is Easy on Polytrees: Pearl’s Belief Propagation

A polytree: a tree with larger families

A polytree decomposition

\[ \lambda_{Z_1}(u_1) = P(z_1 | u_1) \]
\[ \lambda_{Z_2}(u_2) \]
\[ \lambda_{Z_3}(u_3) \]

Running BTE = running Pearl’s BP over the dual graph

BP is Time and space linear
Pearl’s Belief Propagation
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
Iterative Belief Propagation

Algorithm IBP
Input: An arc-labeled dual join-graph \( DJ = (V,E,L) \) for a graphical model \( \mathcal{M} = \langle X,D,F,\Pi \rangle \).
Output: An augmented graph whose nodes include the original functions and the messages received from neighbors. Denote by: \( h_u^v \) the message from \( u \) to \( v \); \( ne(u) \) the neighbors of \( u \) in \( V \); \( ne_v(u) = ne(u) - \{v\} \); \( l_{uv} \) the label of \( (u,v) \in E \); \( \text{elim}(u,v) = \text{scope}(u) - \text{scope}(v) \).

- One iteration of IBP
  For every node \( u \) in \( DJ \) in a topological order and back, do:
    1. Process observed variables
       Assign evidence variables to each \( p_i \) and remove them from the labeled arcs.
    2. Compute and send to \( v \) the function:
       \[
       h_u^v = \sum_{\text{elim}(u,v)} (p_u \cdot \prod_{\{h_i^u, i \in ne_v(u)\}} h_i^u)
       \]
  Endfor

- Compute approximations of \( P(F_i|e) \), \( P(X_i|e) \):
  For every \( X_i \in X \) let \( u \) be the vertex of family \( F_i \) in \( DJ \),
  \[
  P(F_i|e) = \alpha (\prod_{h_i^u, u \in ne(i)} h_i^u) \cdot p_u;
  \]
  \[
  P(X_i|e) = \alpha \sum_{\text{scope}(u)-\{X_i\}} P(F_i|e).
  \]
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

\textbf{Figure 7.1:} An instantiated variable cuts its own cycles.
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 BE.
  - 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 exp in loop-cutset size and linear space. For each cutset we can do BP
Cycle-Cutset Conditioning

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

1-cutset = \{A,B,C\}, size 3
Definition 7.3 \textit{q-cutset, 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 et al., 1998; Becker et al., 2000; Bidyuk and Dechter, 2004]. However, like in the special case of a cycle-cutset we can settle for a non-minimal \textit{q-cutset} relative to a given variable ordering. Namely,

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.
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 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 (Variable-elimination with conditioning)

**Algorithm VEC-evidence**

**Input:** A belief network $\mathcal{B} = \langle \mathcal{X}, \mathcal{D}, \mathcal{G}, \mathcal{P} \rangle$, an ordering $d = (x_1, \ldots, x_n)$; 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

\[ X_1, X_2, X_3, X_4, X_5 \]
Search Basic Step: Conditioning
Search Basic Step: Variable Branching by Conditioning

General principle: Condition until tractable
Then solve sub-problems efficiently
Search Basic Step:
Variable Branching by Conditioning

Example: solve subproblem by inference, BE(i=2)
The Cycle-Cutset Scheme: Condition Until Treeness

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

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

Solve the rest of the problem by any means
Hybrids Variants

- Condition, condition, condition ... and then only eliminate (w-cutset, cycle-cutset)

- Eliminate, eliminate, eliminate ... and then only search

- Interleave conditioning and elimination (elim-cond(i), VE+C)
Interleaving Conditioning and Elimination
(Larrosa & Dechter, CP’02)
Interleaving Conditioning and Elimination
Interleaving Conditioning and Elimination
Interleaving Conditioning and Elimination
Interleaving Conditioning and Elimination
Interleaving Conditioning and Elimination
Interleaving Conditioning and Elimination
What hybrid should we use?

- q=1? (loop-cutset?)
- q=0? (Full search?)
- q=w* (Full inference)?
- q in between?
- depends... on the graph
- What is relation between cycle-cutset and the induced-width?
Properties of Conditioning + Elimination

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

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

Definition 5.6.2 (finding a minimal w-cutset) Given a graph $G = (V, E)$ and a constant $w$, find a smallest subset of nodes $U$, such that when removed, the resulting graph has induced-width less than or equal $w$. 
Theorem 7.7. Given graph $G$, and denoting by $c_q^*$ its minimal $q$-cutset then,

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

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 $c_q + q$. 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.