Chapter 6

Inference with Tree-Clustering

As noted in Chapters 3, 4, topological characterization of tractability in graphical models is centered on the graph parameter called *induced-width* or *tree-width*. In this chapter, we take variable elimination algorithms such as bucket-elimination one step further, showing that they are a restricted version of schemes that are based on the notion of tree-decompositions which is applicable across the whole spectrum of graphical models. These methods have received different names in different research areas, such as join-tree clustering, clique-tree clustering and hyper-tree decompositions. We will refer to these schemes by the umbrella name *tree-clustering* or *tree-decomposition*. The complexity of these methods is governed by the induced-width, the same parameter that controls the performance of bucket elimination.

We will start by extending the bucket-elimination algorithm into an algorithms that process a special class of tree-decompositions, called *bucket-trees*, and will then move to the general notion of tree-decomposition.

6.1 Bucket-Tree Elimination

The bucket-elimination algorithm, BE-bel, for belief updating computes the belief of the first node in the ordering, given all the evidence or just the probability of evidence. However, it is often desirable to answer the belief query for every variable in the network. A brute-force approach will require running BE-bel n times, each time with a different variable order. We will show next that this is unnecessary. By viewing bucket-elimination

as message passing algorithm along a rooted *bucket-tree*, we can augment it with a second message passing from root to leaves which is equivalent to running the algorithm for each variable separately. This yields a two-phase variable elimination algorithm up and down the bucket-tree.

In the following we will describe the idea of message passing along the bucket tree using the general operators of *combine* and *marginalize*. However, to make it more readable we will denote "combine" by product and "marginalize" by summation. In other words instead of $\otimes$ we will write $\prod$ and instead of $\Downarrow$, $\sum$. Some exception for this will occur when we would want the reader to recognize the generality of framework. So, these symbols will stand both for their specific meaning of product and sum (e.g., in the context of probabilistic networks) as well as for these more general meanings of combine and marginalize.

Let $\mathcal{M}$ be a graphical model $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$ and d an ordering of its variables $X_1, ..., X_n$. Let $B_{X_1}, ..., B_{X_n}$ denote a set of bucket, one for each variable. Each bucket B_i contains those functions in V whose latest variable in d is X_i (i.e., according to the bucket-partitioning rule). A bucket-tree of $\mathcal{M}$ has buckets as its nodes. Bucket B_X is connected to bucket B_Y if the function generated in bucket B_X by BE is placed in B_Y. The variables of B_X, are those appearing in the scopes of any of its new and old functions. Therefore, in a bucket tree, every vertex B_X other than the root, has one parent vertex B_Y and possibly several child vertices $B_{Z_1}, ..., B_{Z_t}$. The structure of the bucket-tree can also be extracted from the induced-ordered graph of $\mathcal{M}$ along d using the following definition.

Definition 6.1.1 (bucket-tree, graph-based) *Let $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$ be a graphical model and d an ordering of its variables $d = (X_1, ..., X_n)$. Let ${G^*}_d$ be the induced graph along d of the graphical model whose primal graph is G. The bucket tree has the buckets $\{B_{X_i}\}_{i=1^n}$ as its nodes, each associated with a variable. The bucket* contains *a set of functions and a set of variables. The functions are those placed in the bucket according to the bucket partitioning rule. The set of variables in B_{X_i} is X_i and all its induced-parents in ${G^*}_d$. Abusing notation, we will denote by B_i also its set of variables. Each vertex B_X points to B_Y (or, B_Y is the parent of B_X) if Y is the latest neighbor of X that appear before X in ${G^*}_d$. Each variable X and its earlier neighbors in the induced-graph are the variables of bucket B_X. If B_Y is the parent of B_X in the bucket-tree, then the separator of X and Y is the set of variables appearing in $B_X \cap B_Y$, denoted $sep(X, Y)$.*

Example 6.1.2 Consider the Bayesian network defined over the DAG in Figure 2.5(a). Figure 6.1a shows the initial buckets along ordering $d = A, B, C, D, F, G$, and the messages, labeled by λ, that will be passed by bucket-elimination from top to bottom. Figure 6.1b depicts the same computation as a message-passing along its bucket-tree. Notice that the ordering is displayed bottom-up and messages are passed top-down in the figure. For example, we have the following set of variables in buckets: $B_G = \{G, F\}$, $B_F = \{F, B, C\}$, $B_D = \{D, A, B\}$ and so on. We will often abbreviate B_{X_i} by B_i. □

Definition 6.1.3 (elim(i,j)) *Given a bucket tree having buckets* $\{B_1, ...B_n\}$ *and given a directed edge* (B_i, B_j), $elim(i, j)$ *is the set of variables in* B_i *and not in* B_j, *namely* $elim(i, j) = B_i - sep(i, j)$.

Assume now that we have a Bayesian network and we computed $bel(A)$ using B_A as the first bucket, which is therefore processed last as shown above. Assume that we now want to compute $bel(D)$. Instead of doing all the computation from scratch using a different variable ordering whose first variable is D, we can take the bucket tree and virtually re-orient the edges making D the root of the tree. If we shift messages appropriately as dictated by the partition-rule along the new ordering, we can pass messages from the leaves to this new root. In this new bucket-tree along the ordering D, B, A, C, F, G, the bucket of A B_A includes 3 functions $\{P(A), P(B|A), P(D|B, A)\}$ with variables $\{A, B, D\}$, while the buckets B_B and B_D will have no functions. Subsequently, when BE-BEL process bucket A along this new order, will eliminate variables A and B (in this order) by summation over that product.

The only messages that need to be changed are along the path from A to B to D. It turns out that all these changes can be captured by a second message passing from root to leaves along the original bucket-tree.

Algorithm *bucket-tree-elimination(BTE)* in Figure 6.2 includes the two phases of message passing computations along the bucket-tree. The top-down phase is identical to general bucket-elimination. The bottom-up messages are defined as follows. The messages sent from the root up to the leaves will be denoted by π. The message from B_j to a child B_i combines (e.g., multiply) all the functions currently in B_j including the π messages from its parent bucket and all the λ messages from its *other* child buckets and marginalize (e.g., sum) over the eliminator from B_j to B_i. We see that upwards messages

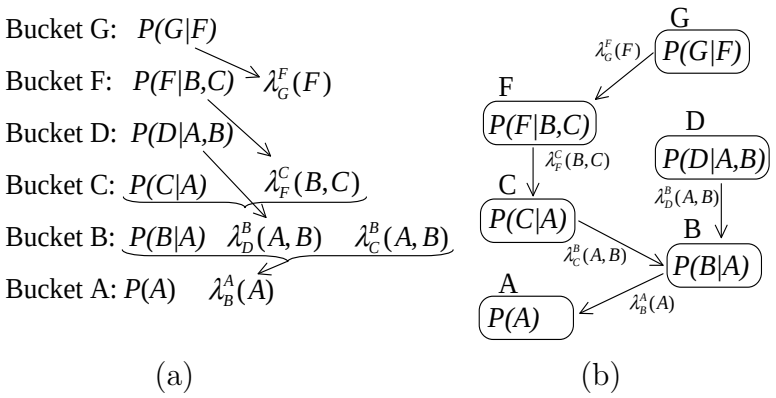

Figure 6.1: Execution of BE along the bucket-tree

may be generated by eliminating zero, one or more variables while going down each bucket eliminate a single variable.

Example 6.1.4 Figure 6.3a shows the complete execution of BTE along the linear order of buckets and along the bucket-tree, for the belief-updating task. The π and λ messages are placed on the outgoing upward directed arcs. The π functions in the up phase are depicted in Figure 6.3b and are computed as follows:
$\pi_A^B(a) = P(a)$
$\pi_B^C(c,a) = P(b|a)\lambda_D^B(a,b)\pi_A^B(a)$
$\pi_B^D(a,b) = P(b|a)\lambda_C^B(a,b)\pi_A^B(a,b)$
$\pi_C^F(c,b) = \sum_a P(c|a)\pi_B^C(a,b)$
$\pi_F^G(f) = \sum_{b,c} P(f|b,c)\pi_C^F(c,b)$
□

The formal correctness of BTE will follow as a special case of the correctness of a larger class of tree propagation algorithms, as we show later.

Theorem 6.1.5 *When Algorithm BTE terminates, the combination (e.g., product) of functions in each bucket is the marginal over the bucket's variables joint with the evidence. Respectively, then* $\Pi_{f \in B_i} f = bel(B_i, e)$.

When BTE terminates, each bucket B_i has π_j^i received from its parent B_j in the tree, its own original f functions and the λ_k^i sent from each child B_k. Then, the belief queries can be computed by combining all the functions in a bucket as specified by BTE.

Algorithm bucket-tree elimination (BTE)
Input: A problem $\mathcal{M} = \langle \mathbf{X}, \mathbf{D}, \mathbf{F}, \prod \rangle$, ordering d.
Output: Augmented buckets containing the original functions and all the π and λ functions received from neighbors in the bucket-tree.
0. Pre-processing:
Place each function in the latest bucket, along d, that mentions a variable in its scope. Connect two buckets B_i and B_j if variable X_j is the latest earlier neighbor of X_i in the induced graph G_d.

1. Top-down phase: λ messages (BE)
For $i = n$ to 1, process bucket B_i:
Let $\lambda_{i_1}, ...\lambda_{i_r}$ be all the functions in B_i at the time B_i is processed, including the original functions of F. The message λ_i^j sent from B_i to its parent B_j, is computed by

$$\lambda_i^j = \sum_{elim(j,i)} \prod_{k,k \neq j} \lambda_{i_k}$$

2. bottom-up phase: π messages
For $j = 1$ to n, process bucket B_j:
Let $\lambda_{j_1}, ..., \lambda_{j_r}$ be all the functions in B_j at the time B_j is processed, including the original functions of F. B_j takes the π message received from its child B_k, π_k^j, and computes a message π_j^i for each child bucket B_j by

$$\pi_j^i = \sum_{elim(j,i)} \pi_k^j \cdot (\prod_{r \neq i} \lambda_r^j)$$

3. Answering singleton queries (e.g., deriving beliefs)
the functions $f_1, ..., f_t$ in the augmented bucket B_X at termination, compute

$$bel(B_X) = \prod_{f \in B_i} f$$

and the belief of X is computed by

$$Bel(X) = \sum_{B_X - \{X\}} \prod_{f \in B_j} f$$

Figure 6.2: Algorithm Bucket-Tree Elimination

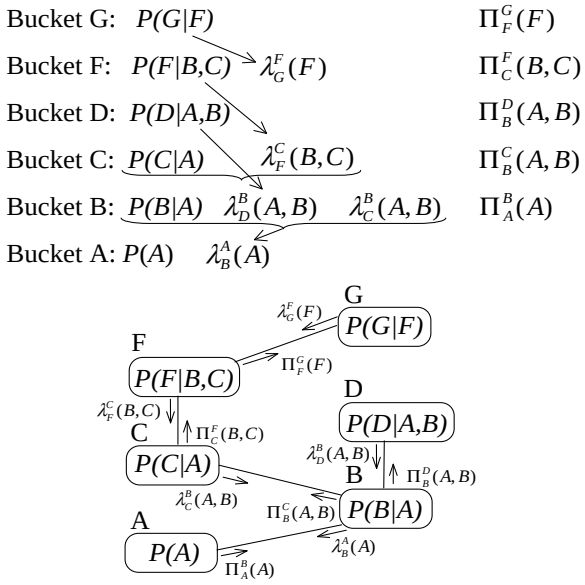

Figure 6.3: Propagation of π's and λ's along the bucket-tree

We next address the complexity of BTE, and in particular compare it with n executions of BE.

Theorem 6.1.6 (Complexity of BTE) *Let w^* be the induced width of G along ordering d, r be the number of functions in the given Bayesian network and k be the maximum domain size. The time complexity of BTE is $O(r \cdot deg \cdot k^{w^*+1})$, where deg is the maximum degree in the bucket-tree. The space complexity of BTE is $O(n \cdot k^{w^*})$.*

Proof: Since the number of buckets is n, and the induced width w^*, the downward λ messages take $O(r \cdot k^{w^*+1})$ as we already shown. The upward messages per bucket are computed for each of its child nodes and each such message takes $O(r_i \cdot k^{w*+1})$ steps, yielding a total of $O(r_i \cdot deg \cdot k^{w*+1})$. Overall, we get complexity of $O(r \cdot deg \cdot k^{w^*+1})$. Since the size of each message is k^{sep}, and since here $sep = w^*$, we get space complexity of $O(n \cdot k^{w^*})$. $\square$.

The complexity of BTE can be improved to $O(rk^{w^*+1})$ time and $O(nk^{w^*+1})$ space [**?**], but this distinction is subsumed by a more general tree-propagation algorithm that we will describe shortly.

In theory the speedup expected from running BTE vs running n-BE (BE n times) is at most n. This may seem insignificant compared with the exponential complexity in

Bucket-Tree Propagation (BTP)
Input: For each node X_i, its bucket B_i and its neighboring buckets. Let λ_j^i be the message sent to B_i from its neighbor B_j and $f_{i_1}, ..., f_{i_k}$ the original functions in bucket B_i.
The message B_i sends to a neighbor B_j, once it received all the messages from its neighbors except from B_j is:

$$\lambda_i^j = \sum_{B_i - sep(i,j)} (\prod_i f_i) \cdot (\prod_{k \neq j} \lambda_k^i)$$

Figure 6.4: The Bucket-tree propagation (BTP) for X

w^*, however in practice it can be very significant. The actual speedup of BTE relative to n-BE may be smaller than n, however. We know that the complexity of n-BE is $O(n \cdot r \cdot k^{w^*+1})$, whereas the complexity of BTE is $O(deg \cdot r \cdot k^{w^*+1})$.

6.1.1 Bucket-tree propagation, an asynchronous version

The BTE algorithm can be described in an asynchronous manner when viewing the bucket-tree as an undirected tree and passing only one type of messages which we will denote by λ. In this view, each bucket receives λ messages from each of its neighbors and each sends a λ message to every neighbors. We distinguish between the original f functions placed in bucket B_i and the messages that it received from its neighbors. The algorithm is described in Figure 6.4. We call the resulting algorithm Bucket Tree Propagation or BTP.

Proposition 6.1.7 *Let $\{f_i\}, i = 1, ...j$ be the original functions in B_X, let $Y_1, ...Y_k$. Algorithm BTP is guaranteed to converge to the same bucket content as BTE .*

Proof: see exercises ∎

6.2 From Buckets to Super-Buckets to cluster-Tree elimination

The BTE and BTP algorithms are special cases of a wider class of algorithms all based on an underlying tree decomposition. A tree-decomposition takes a graphical model and embeds it in a tree of clusters, where each cluster is defined by a set of variables and a set of the input functions. The decomposition allows message-passing between the clusters in a manner similar to BTE. The correctness of this message passing algorithms is tied to the fact that the clusters are connected in a tree. Graphical models having such graphical structures are called *acyclic Graphical models* [28], and also called *decomposable graphical models* in [33].

6.2.1 Acyclic graphical models

To review, a hypergraph is a structure $\mathcal{H} = (V, S)$ that consists of a set of vertices $V = \{v_1, .., v_n\}$ and a set of subsets of these vertices $S = \{S_1, ..., S_l\}$, $S_i \subseteq V$, called hyperedges. The hyperedges differ from regular edges in that they each "connect" more than two variables.

As noted earlier, a hypergraph $\mathcal{H} = (V, S)$ can be mapped to a regular graph called a *dual graph* $\mathcal{H}^{dual}$. The nodes of the dual graph are the hyperedges from the hypergraph, and a pair of such nodes is connected if they share vertices in V. The arc that connects two such nodes is labeled by the shared vertices. Formally, given $\mathcal{H} = (V, S)$, $\mathcal{H}^{dual} = (S, E)$ where

$S = \{S_1, ..., S_l\}$ are edges in $\mathcal{H}$, and $(S_i, S_j) \in E$ iff $S_i \cap S_j \neq \emptyset$. A primal graph of a hypergraph $\mathcal{H} = (V, S)$ has V as its set of nodes, and any two nodes are connected by an arc if they appear in the same hyperedge. Note that if all the functions in the graphical model have scopes of 2, then its hypergraph is identical to its primal graph.

Any graphical model $\mathcal{R} =< X, D, G, \mathcal{F} >$, $F = \{f_{S_1}, ..., f_{S_t}\}$ can be associated with a hypergraph $\mathcal{H}_{\mathcal{R}} = (X, H)$, where X is the set of nodes (variables), and H is the set of scopes of the functions in $\mathcal{F}$, namely $H = \{S_1, ..., S_l\}$. Therefore, the corresponding dual graph of the graphical model associates a node with each function's scope and an arc for each two nodes sharing variables.

If a problem's dual graph happens to be a tree, it can be shown that it can be solved in linear time using a BTE-like message-passing algorithm. It turns out, however, that sometimes some arcs can be removed from the dual graph while maintaining the same independency relationships that hold in the hypergraph and in the primal graph. These arcs can be viewed as redundant. An arc can be deleted if the variables labeling the arc are shared by every arc along an *alternate* path between the two end points. This is because the alternate path already enforces the necessary dependencies.

We call the property that ensures such legitimate arc removal the *running intersection property* or *connectedness* property. The running intersection property can be defined over hypergraphs or over their dual graphs, and is used to characterize equivalent concepts such as *join-trees* (defined over dual graphs) or *hypertrees* (defined over hypergraphs). An *arc subgraph* of a graph contains the same set of nodes as the graph, and a subset of its arcs.

Definition 6.2.1 (connectedness, join-trees, hypertrees and acyclic networks) *Given a dual graph of a hypergraph, 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, each containing the shared variables. An arc subgraph of the dual graph that satisfies the connectedness property is called a* join-graph. *A join-graph that is a tree is called a* join-tree. *A hypergraph whose dual-graph has a join-tree is called a* hypertree. *A graphical model whose hypergrpah is a hyper-tree is called an* acyclic graphical model.

Example 6.2.2 Considering again the graphs in Figure 5.1, we can see that the join-tree in Figure 5.1(d) satisfies the connectedness property. The hypergraph in Figure 5.1(a) has a join-tree and is therefore a hypertree. □

Theorem 6.2.3 *Given an acyclic graphical model, algorithm BTE can compute the marginal problem on each scope of an input function in linear time and space.*

Proof: left as an exercise ∎

So now that we have established that acyclic decomposable models can be solved efficiently, all that remains is to transform a general graphical model into an acyclic one. This indeed is what algorithm tree-decomposition does.

6.2.2 Tree-decomposition and Cluster-tree elimination

We now define those *cluster-tree decompositions* which facilitate message propagation along a tree of clusters. We will show that a bucket-tree is a special case.

Definition 6.2.4 (tree-decomposition, cluster tree) *Let* $\mathcal{M} =< X, D, F, \prod >$ *be a graphical model. A* tree-decomposition *for* $\mathcal{M}$ *is a triple* $< T, \chi, \psi >$*, where* $T = (V, E)$ *is a tree, and* χ *and* ψ *are labeling functions which associate with each vertex* $v \in V$ *two sets,* $\chi(v) \subseteq X$ *and* $\psi(v) \subseteq F$ *satisfying:*

1. *For each function* $f_i \in F$*, there is* exactly *one vertex* $v \in V$ *such that* $f_i \in \psi(v)$*, and* $scope(f_i) \subseteq \chi(v)$*.*
2. *For each variable* $X_i \in X$*, the set* $\{v \in V | X_i \in \chi(v)\}$ *induces a connected subtree of* T*. This is also called the running intersection property.*

We will often refer to a node and its functions as a cluster *and use the term* tree-decomposition *and* cluster tree *interchangeably.*

We will redefine now earlier graph-parameters using the notion of tree-decomposition.

Definition 6.2.5 (treewidth, separator-width, eliminator) *The* treewidth *[3] of a tree-decomposition* $< T, \chi, \psi >$ *is* $max_{v \in V} |\chi(v)|$*. Given two adjacent vertices* u *and* v *of a tree-decomposition, the* separator *of* u *and* v *is defined as* $sep(u, v) = \chi(u) \cap \chi(v)$*, and the* eliminator *of* u *with respect to* v *is* $elim(u, v) = \chi(u) - \chi(v)$*. The separator-width is the maximum over all separators.*

Example 6.2.6 Consider the belief network in Figure 2.5a. Any of the trees in Figure 6.6 are tree-decompositions for this problem where the functions can be partitioned into clusters that contain their scopes. The labeling χ are the sets of variables in each node. For example, Figure 6.6C shows a cluster-tree decomposition with two vertices, and labelling $\chi(1) = \{G, F\}$ and $\chi(2) = \{A, B, C, D, F\}$. Any function with scope $\{G\}$ must be placed in vertex 1 because vertex 1 is the only vertex that contains variable G (placing a function having G in its scope in another vertex will force us to add variable G to that vertex as well). Any function with scope $\{A, B, C, D\}$ or its subset must be placed in vertex 2, and any function with scope $\{F\}$ can be placed either in vertex 1 or 2. Note that the trees in Figure 6.6 are drawn upside-down, namely, the leaves are at the top and the root is at the bottom. □

Algorithm cluster-tree elimination (CTE)
Input: A tree decomposition $< T, \chi, \psi >$ for a problem $M =< X, D, F, \prod\} >$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$.
Output: An augmented tree whose vertices are clusters containing the original functions as well as messages received from neighbors. A solution computed from the augmented clusters.
Compute messages:
For every edge (u, v) in the tree, do

- Let $m_{(u,v)}$ denote the message sent by vertex u to vertex v.
- Let $cluster(u) = \psi(u) \cup \{m_{(i,u)} | (i, u) \in T\}$.
- If vertex u has received messages from all adjacent vertices other than v, then compute and send to v,

$$m_{(u,v)} = \sum_{sep(u,v)} (\prod_{f \in cluster(u), f \neq m_{(v,u)}} f)$$

Endfor
Note: functions whose scope does not contain elimination variables do not need to be processed, and can instead be directly passed on to the receiving vertex.
Return: A tree-decomposition augmented with messages, and for every $v \in T$

Figure 6.5: Algorithm Cluster-Tree Elimination (CTE)

Notice that it may be that $sep(u, v) = \chi(u)$ (that is, all variables in vertex u belong to an adjacent vertex v). In this case the size of the tree-decomposition can be reduced by merging vertex u into v without increasing the tree-width of the tree-decomposition.

Definition 6.2.7 (minimal tree-decomposition) *A tree-decomposition is minimal if* $sep(u, v) \subset \chi(u)$ *and* $sep(u, v) \subset \chi(v)$.

We immediately observe that the bucket-tree is often not minimal. We can make it minimal however, by having each subsumed bucket be absorbed into its containing bucket, yielding *super-bucket* tree.

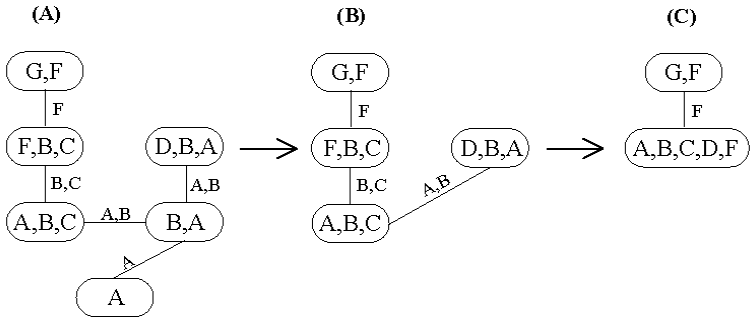

Figure 6.6: From a bucket-tree to join-tree to a super-bucket-tree

A tree-decomposition allows answers to queries using message-passing as we saw in BTE. The algorithm, called *Cluster-tree elimination algorithm CTE*, is presented in Figure 6.5. Each vertex of the tree sends a function to each of its neighbors. All the functions in vertex u and all messages received by u from all its neighbors other than v are combined using the combination operator (e.g., product). The combined function is projected onto the separator of u and v using the marginalization operator and the projected function is then sent from u to v. Functions that do not share variables with the eliminated variables are passed along separately in the message.

Vertex activation can be asynchronous and convergence is guaranteed. If processing is performed from leaves to root and back, convergence is guaranteed after two passes, where only one message is sent on each edge in each direction. If the tree contains m edges, then a total of $2m$ messages will be sent.

Example 6.2.8 Consider again the graphical model whose primal graph appears in Figure 2.5(a). Assume all functions are on pairs of variables (you can think of this as a Markov network). Two tree-decompositions are described in Figure 6.8. Figure 6.9 shows the messages propagated for the tree-decomposition in Figure 6.8b. Since cluster 1 contains only one function, the message from cluster 1 to 2 is the f_{FD} over the separator between cluster 1 and 2, which is variable D. The message $m_{(2,3)}$ from cluster 2 to cluster 3 combines the functions in cluster 2 with the message $m_{(1,2)}$, and projects over the separator between cluster 2 and 3, which is $\{B, C\}$, and so on. □

Once all vertices have received messages from all their neighbors, a solution to the problem can be generated using the output augmented tree (as described in the algorithm)

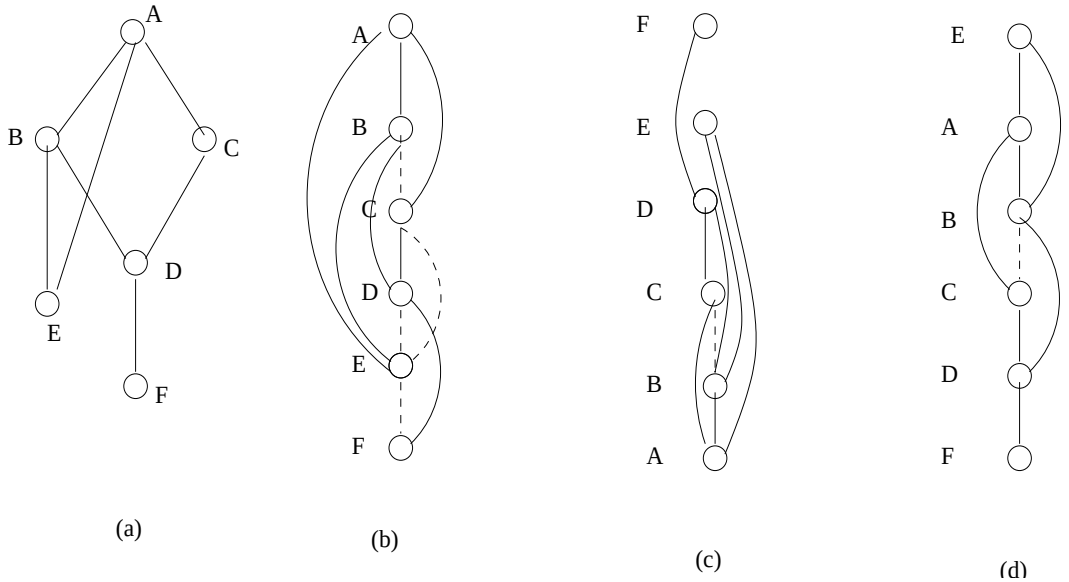

Figure 6.7: A graph (a) and two of its induced graphs (b) and (c).

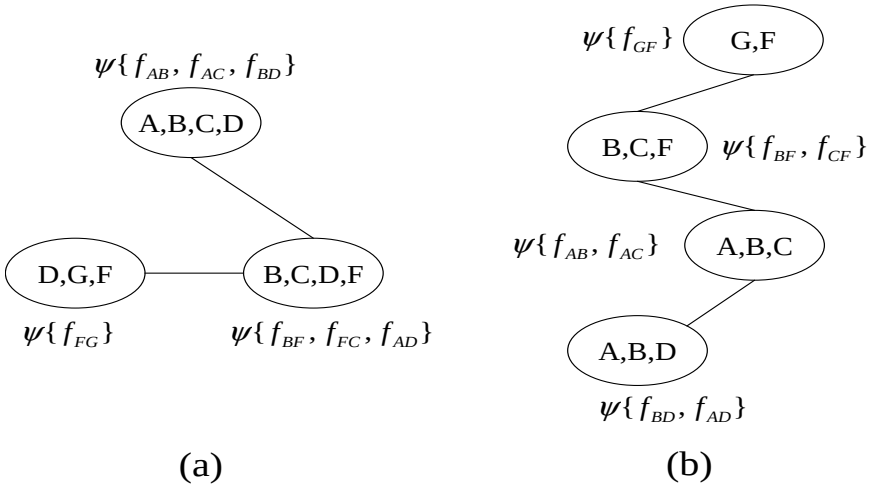

Figure 6.8: Two tree-decompositions of a graphical model

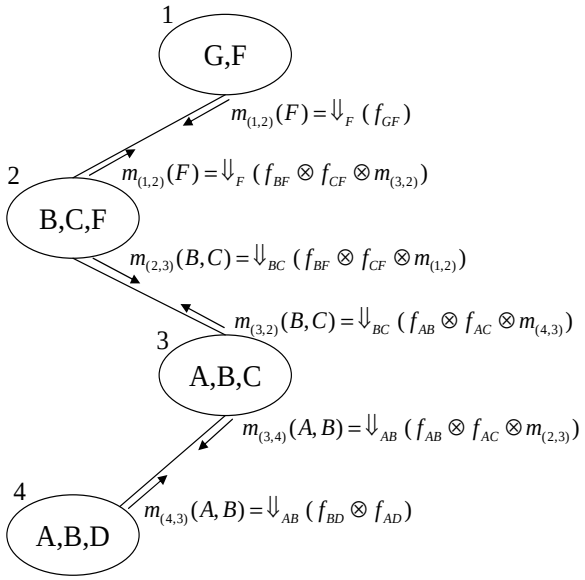

Figure 6.9: Example of messages sent by CTE

.

in output linear time. For some tasks the whole output tree is used to compute the solution (e.g., computing an optimal tuple).

6.2.3 The special case of Belief Updating

As noted earlier, the most used tree decomposition method is called join-tree decomposition [27, 17] (also called junction-trees). Such decompositions can be generated by embedding the network's moral graph, G, in a chordal graph, using a triangulation algorithm. The maximal cliques of the generated chordal graph can serve as nodes in the join-tree. Subsequently, every CPT p_i is placed in one clique containing its scope. A join-tree decomposition of a belief network (G, P) is a tree $T = (V, E)$, where V is the set of maximal cliques of a chordal graph G' that contains G, and E is a set of edges that form a tree between cliques, satisfying the running intersection property [28].

Algorithm CTE for belief updating denoted CTE-BU is described again in Figure 6.11. The algorithm pays a special attention to the processing of observed variables since the presence of evidence is a central component in belief updating. When a cluster sends a

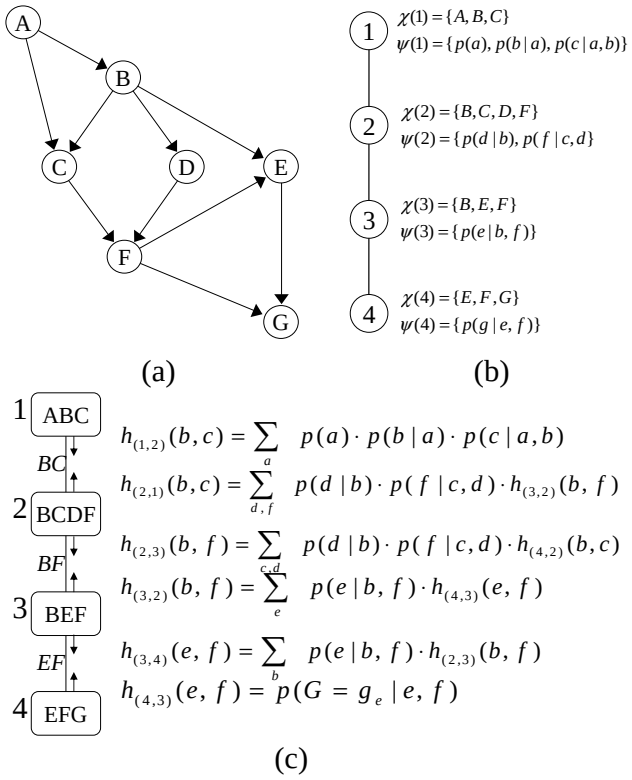

Figure 6.10: *a*) A belief network; *b*) A join-tree decomposition; *c*)Execution of CTE-BU; no individual functions appear in this case

message to a neighbor, the algorithm operates on all the functions in the cluster except the message from that particular neighbor. The message contains a single *combined* function and *individual* functions that do not share variables with the relevant eliminator. All the non-individual functions are *combined* in a product and summed over the eliminator.

Example 6.2.9 Figure 6.10 describes a belief network (a) and a join-tree decomposition for it (b). Figure 6.10c shows the trace of running CTE-BU. In this case no individual functions appear between any of the clusters. □

Algorithm **CTE for Belief-Updating (CTE-BU)**
Input: A tree decomposition $< T, \chi, \psi >$, $T = (V, E)$ for $BN =< X, D, G, P >$. Evidence variables $var(e)$.
Output: An augmented tree whose nodes are clusters containing the original CPTs and the messages received from neighbors. $P(X_i, e),\ \forall X_i \in X$.

Denote by $H_{(u,v)}$ the message from vertex u to v, $ne_v(u)$ the neighbors of u in T excluding v.
$cluster(u) = \psi(u) \cup \{H_{(v,u)} | (v,u) \in E\}$.
$cluster_v(u) = cluster(u)$ excluding message from v to u.

• **Compute messages:**
For every node u in T, once u has received messages from all $ne_v(u)$, compute message to node v:

1. **Process observed variables:**
   Assign relevant evidence to all $p_i \in \psi(u)$
2. **Compute the combined function:**

$$h_{(u,v)} = \sum_{elim(u,v)} \prod_{f \in A} f.$$

   where A is the set of functions in $cluster_v(u)$ whose scope intersects $elim(u, v)$.
   Add $h_{(u,v)}$ to $H_{(u,v)}$ and add all the individual functions in $cluster_v(u) - A$
   Send $H_{(u,v)}$ to node v.

• **Compute $P(X_i, e)$:**
For every $X_i \in X$ let u be a vertex in T such that $X_i \in \chi(u)$. Compute $P(X_i, e) = \sum_{\chi(u)-\{X_i\}} (\prod_{f \in cluster(u)} f)$. $bel(X_i) = \alpha P(X_i, e)$

Figure 6.11: Algorithm Cluster-Tree-Elimination for Belief Updating (CTE-BU)

6.3 Properties of CTE

6.3.1 Correctness of CTE

We can prove correctness by relying on the correctness of CTE when applied to an acyclic network and by realizing the a tree-decomposition can be viewed as transforming a general graphical model into an acyclic one.

We will next give a general direct proof. We will prove the general case using basic properties of the *combine* and *marginalize* operators in order to emphasize the broad applicability of this algorithm. For emphasis we will now use general operator notations. The theorem articulates the properties (which are all obeyed by graphical models) required for correctness.

Theorem 6.3.1 (soundness and Completeness) *Assuming that the combination operator $\bigotimes_i$ and the marginalization operator $\Downarrow_Y$ satisfy the following properties:*

1. *Order of marginalization does not matter:*
   $\Downarrow_{X-\{X_i\}} (\Downarrow_{X-\{X_j\}} f(X)) = \Downarrow_{X-\{X_j\}} (\Downarrow_{X-\{X_i\}} f(X))$
2. *Commutativity:* $f \bigotimes g = g \bigotimes f$
3. *Associativity:* $f \bigotimes (g \bigotimes h) = (f \bigotimes g) \bigotimes h$
4. *Restricted distributivity:*
   $\Downarrow_{X-\{X_k\}} [f(X - \{X_k\}) \bigotimes g(X)] = f(X - \{X_k\}) \bigotimes \Downarrow_{X-\{X_k\}} g(X)$

Algorithm CTE is sound and complete.

Proof. By definition, solving an automated reasoning problem P requires computing a function $F(Z_i) = \Downarrow_{Z_i} \bigotimes_{i=1}^{r} f_i$ for each Z_i. Using the four properties of combination and marginalization operators, the claim can be proved by induction on the depth of the tree as follows.

Let $< T, \chi, \psi >$ be a cluster-tree decomposition for P. By definition, there must be a vertex $v \in T$, such that $Z_i \subseteq \chi(v)$. We create a partial order of the vertices of T by making v the root of T. Let $T_u = (N_u, E_u)$ be a subtree of T rooted at vertex u. We define $\chi(T_u) = \bigcup_{w \in N_u} \chi(w)$ and $\chi(T - T_u) = \bigcup_{w \in \{N - N_u\}} \chi(w)$.

We rearrange the order in which functions are combined when $F(Z_i)$ is computed. Let $d(j) \in N, j = 1, ..., |N|$ be a partial order of vertices of the rooted tree T, such that a vertex must be in the ordering before any of its children. The first vertex in the ordering is the root of the tree. Let $F_u = \bigotimes_{f \in \psi(u)} f$. We define

$$F^{'}(Z_i) = \Downarrow_{Z_i} \bigotimes_{j=1}^{|N|} F_{d(j)}$$

Because of associativity and commutativity, we have $F^{'}(Z_i) = F(Z_i)$.

We define $e(u) = \chi(u) - sep(u, w)$, where w is the parent of u in the rooted tree T. For the root vertex v, $e(v) = X - Z_i$. In other words, $e(u)$ is the set of variables that are eliminated when we go from u to w. We define $e(T_u) = \bigcup_{w \in N_u} e(w)$, that is, $e(T_u)$ is the set of variables that are eliminated in the subtree rooted at u. Because of the connectedness property, it must be that $e(T_u) \bigcap \{X_i | X_i \in \chi(T - T_u)\} = \emptyset$. Therefore, variables in $e(T_u)$ appear only in the subtree rooted at u.

Next, we rearrange the order in $F^{'}(Z_i)$ in which the marginalization is applied. If $X_i \notin Z_i$ and $X_i \in e(d(k))$ for some k, then the marginalization eliminating X_i can be applied to $\bigotimes_{j=k}^{|N|} F_{d(j)}$ instead of $\bigotimes_{j=1}^{|N|} F_{d(j)}$. This is safe to do, because as shown above, if a variable X_i belongs to $e(d(k))$, then it cannot be part of any $F_{d(j)}$, $j < k$. Let $ch(u)$ be the set of children of u in the rooted tree T. If $ch(u) = \emptyset$ (vertex u is a leaf vertex), then we define $F^u = \Downarrow_{X - e(u)} F_u$. Otherwise we define $F^u = \Downarrow_{X - e(u)} (F_u \bigotimes_{w \in ch(u)} F^w)$. If v is the root of T, we define

$$F^{''}(Z_i) = F^v$$

Because of properties 1 and 4, we have $F^{''}(Z_i) = F(Z_i)$. However, $F^{''}(Z_i)$ is exactly what the cluster-tree algorithm computes. The message that each vertex u sends to its parent is F^u. This concludes the proof. □

6.3.2 Complexity of CTE

Algorithm CTE can be subtly varied to influence its time and space complexities. The description in Figure 6.5 may imply an implementation whose time and space complexity are the same. At first glance, it seems that the space complexity is also exponential in w^*. Indeed, if we first record the combined function in Equation 6.2.2 and subsequently marginalized on the separator, we will have space complexity exponential in w^*. However, we can interleave the combination and marginalization operations, and thereby make the space complexity identical to the size of the sent message as follows. In Equation 6.2.2, we compute the message m, which is a function defined over the separator, *sep*, because all the variables in the *eliminator*, $elim(u) = \chi(u) - sep$, are eliminated by marginalization (e.g., summation). This can be implemented by enumeration (or search) as follows: For each assignment a to $\chi(u)$, we compute the combined functional value, and accumulate

the marginalization value on the separator, *sep*, updating a_{sep}, of the message function $m(sep)$.

Theorem 6.3.2 (Complexity of CTE) *Let N be the number of vertices in the tree decomposition, w its tree-width, sep its maximum separator size, r the number of input functions in F, deg the maximum degree in T, and k the maximum domain size of a variable. The time complexity of CTE is $O((r+N)\cdot deg\cdot k^{w+1})$ and its space complexity is $O(N\cdot k^{sep})$.*

Proof. The time complexity of processing a vertex u is $deg_u\cdot(|\psi(u)|+deg_u-1)\cdot k^{|\chi(u)|}$, where deg_u is the degree of u, because vertex u has to send out deg_u messages, each being a combination of $(|\psi(u)|+deg_u-1)$ functions, and requiring the enumeration of $k^{|\chi(u)|}$ combinations of values. The time complexity of CTE is

$$Time(CTE)=\sum_u deg_u\cdot(|\psi(u)|+deg_u-1)\cdot k^{|\chi(u)|}$$

By bounding the first occurrence of deg_u by deg and $|\chi(u)|$ by the tree-width $w+1$, we get

$$Time(CTE)\leq deg\cdot k^{w+1}\cdot\sum_u(|\psi(u)|+deg_u-1)$$

Since $\sum_u|\psi(u)|=r$ we can write

$$Time(CTE)\leq deg\cdot k^{w+1}\cdot(r+N)$$

$$=O((r+N)\cdot deg\cdot k^{w+1})$$

For each edge CTE will record two functions. Since the number of edges is bounded by N and the size of each function we record is bounded by k^{sep}, the space complexity is bounded by $O(N\cdot k^{sep})$.

If the cluster-tree is minimal (for any u and v, $sep(u,v)\subset\chi(u)$ and $sep(u,v)\subset\chi(v)$), then we can bound the number of vertices N by n. Assuming $r\geq n$, the time complexity of a minimal CTE is $O(deg\cdot r\cdot k^{w+1})$. □

Indeed, the above complexity is what we also observed for BTE. We will next show that bucket-trees are tree-decompositions. From this we can infer that CTE on a bucket-tree (which coincides with BTE) is sound, providing an alternative proof for its correctness.

Theorem 6.3.3 *A bucket tree of a graphical model $\mathcal{M}$ is a tree-decomposition of $\mathcal{M}$.*

Proof: We need to show how to construct a tree $T = (V, E)$ and mappings χ and ψ, as well as prove that conditions of Definition 6.2.4 are satisfied. There is a one-to-one correspondence between vertices of V and buckets B_i (in the following we will refer to buckets as vertices of the cluster-tree). If a bucket B_i has a parent (is connected to) bucket B_j, there is an edge $(B_i, B_j) \in E$. Labelling $\chi(B_i)$ is the union of the signatures of new and old functions in B_i, and labeling $\psi(B_i)$ is the set of new functions in B_i.

Condition 1 of Definition 6.2.4 is satisfied because each function is placed in exactly one bucket; condition 2 of Definition 6.2.4 is also satisfied because labeling $\chi(B_i)$ is the union of scopes of all functions in B_i. Condition 4 of Definition 6.2.4 is trivially satisfied since there is exactly one bucket for each variable.

Finally we need to prove the connectedness property (condition 3 of Definition 6.2.4). Lets assume that there is a variable X_k with respect to which the connectedness property is violated. This means that there must be (at least) two disjoint subtrees, T_1 and T_2, of T, such that each vertex in both subtrees contains X_k, and there is no edge between a vertex in T_1 and T_2. Let B_I be a vertex in T_1 such that X_i is the earliest relative to ordering d, and B_j a vertex in T_2 such that X_j is the earliest in ordering d. Since T_1 and T_2 are disjoint, it must be that $X_i \neq X_j$. However, this is impossible since this would mean that there are two buckets that eliminate variable X_k. □

One way of structuring tree-decompositions beyond bucket-trees is to generate a bucket-tree from an induced graph, and then create subsequent trees by merging adjacent clusters. Indeed,

Proposition 6.3.4 *If T is a tree-decomposition, then any tree obtained by merging adjacent clusters is also a tree-decomposition.*

Proof: see exercises.

So, to obtain a new tree-decomposition we can start from a bucket-tree and merge adjacent buckets, yielding *super-buckets*. The maximal cliques in the induced-order graph are a special kind of super-buckets which form a tree-decomposition called *join-tree*.

Example 6.3.5 [show a detailed example showing the computation by CTE that is bounded by the separator size] □

The separator sizes in bucket-trees are equal to the cluster sizes (minus 1) and therefore the time complexity and space complexity for BTE/BTP are the same. In general however, for any cluster tree and, in particular, for the join-tree, the separators sizes may be far smaller than the maximal cliques sizes.

Example 6.3.6 Consider the e tree-decompositions in Figure 6.7. The first two yield CTE having time exponential in three and space exponential in 2. The third yield time exponential in 5 and space linear. □

6.4 Message propagation schemes

It is clear that whenever we have a graphical model that is already a real tree, its treewidth or induced-width is 1. We also saw that whenever a graphical model is acyclic message-passing is efficient and can be accomplished in linear time and space. A special case of acyclic graphical models which are not strictly speaking real trees, are *polytrees*. Therefore, when a directed graphical model (e.g., a Bayesian network) is a *polytree* most queries of interest such as marginals and optimization can be accomplished in linear time and space [33]. This case deserves special attention for historical reasons; it was recognized by Pearl as a generalization of trees on which his belief propagation algorithm was defined and shown to be sound and complete, and it also gives rise to an iterative approximation algorithm over general networks, as we will discuss later.

Definition 6.4.1 (polytree) *A polytree is a directed acyclic graph whose underlying undirected graph has no cycles (see Figure 6.12(a)).*

A polytree decomposition. Given a Bayesian network which is a polytree, its dual graph can be easily seen to be a tree, and thus yield a join-tree decomposition. Namely, each variable X and its parents $pa(X)$ is a node C_X that includes its CPT, $P(X|pa(X))$ and the family variables. Note, that the separators of this cluster tree are all singleton variables. Clearly,

Proposition 6.4.2 *A polytree has a tree dual graph and it is therefore an acyclic graphical model.*

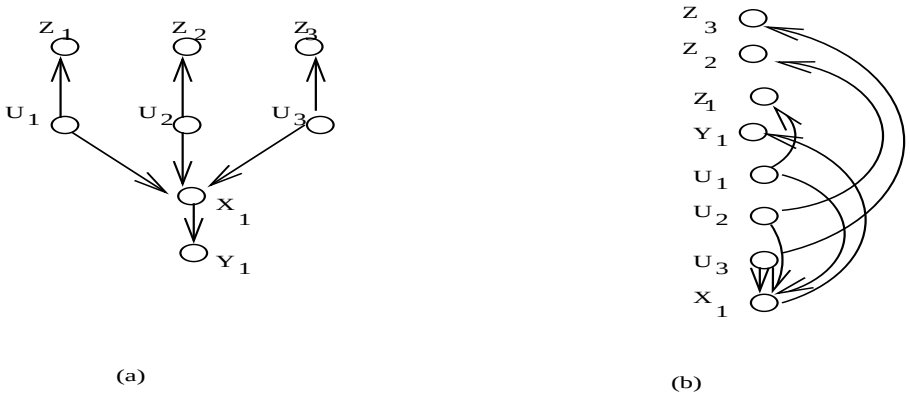

Figure 6.12: (a) A polytree and (b) a legal processing ordering

If we direct the edges of the polytree-decomposition from a cluster of a parent node X to the cluster of its child, the π and λ messages that propagate along the polytree decomposition using BTE (or CTE) can be shown to be identical to the original Pearl's belief propagation messages. We therefore define *belief propagation (BP)* as algorithm CTE algorithm that is applied to a polytree dual graph decomposition. Indeed,

Theorem 6.4.3 *Given a polytree network and its dual graph, algorithms CTE applied along its join-tree is time and space linear in the network's size.* □

Algorithm belief propagation is one iteration of the algorithm presented in Figure 6.14

Example 6.4.4 Consider the polytree given in Figure 6.12. It is easy to see that if we use a width-1 ordering of the variables (looking at the undirected tree) then each bucket contains exactly one function. Therefore, the bucket tree generated is identical to the dual join-tree in this case. Message passing from leaves to the root X_1 and back will facilitate the marginals for each variable. □

In the next section we will discussion how can belief propagation be considered as an iterative algorithm for general graphical models.

6.4.1 Iterative Belief Propagation over Dual Join-Graphs

Since the message propagation algorithm over tree networks is defined distributively as a message-passing algorithm between the original functions, it is well defined even if

executed over a network with loops. Note that the notion of bucket-tree allows the view of message passing between the original variables, while the dual graph notion emphasize the view of message-passing between functions. Both view are equally useful and correspond to the same algorithm.

Iterative belief propagation (IBP) is an iterative application of *belief propagation BP* defined above for poly-trees [33]. In this section we will present IBP as an instance of cluster-tree elimination over variants of the *dual graph* that may have loops. While we already defined the notion of a dual graph using the intermediate concept of hypergraphs, we will now redefine it directly for graphical models.

Definition 6.4.5 (dual graphs, join dual graphs, arc-minimal dual-graphs) *Given a graphical model* $\mathcal{M} =< X, D, F, \prod >$.

- *The* dual graph $\mathcal{D}_{\mathcal{F}}$ *of the graphical model* $\mathcal{M}$*, is an arc-labeled graph defined over the its functions. Namely, it has a node for each function labeled with the function's scope and a labeled arc connecting any two nodes that share a variable in the function's scope. The arcs are labeled by the shared variables.*
- *A* dual join-graph *is a labeled arc subgraph of* $\mathcal{D}_{\mathcal{F}}$ *whose arc labels are subsets of the labels of* $\mathcal{D}_{\mathcal{F}}$ *such that the* running intersection property*, is satisfied.*
- *An* arc-minimal *dual join-graph is a dual join-graph for which none of its labels can be further reduced while maintaining the connectedness property.*

Recall that the running intersection property requires that any two nodes that share a variable in the dual join-graph be connected by a path of arcs whose labels contain the shared variable. Clearly the dual graph itself is a dual join-graph because any two nodes that share a variable are directly connected. Interestingly, there are many dual join-graphs of the same dual graph and many of them are arc-minimal.

We define Iterative Belief Propagation on a dual join-graph. Each node sends a message over an arc whose scope is identical to the label on that arc. Since the polytree algorithm sends messages whose scopes are singleton variables only, we will highlight arc-minimal singleton dual join-graph which capture this polytree property. Given a Bayesian network, one such dual graph can be constructed directly from its directed graph by labeling each arc connecting a child family with its parent family by its parent name. It can be shown that:

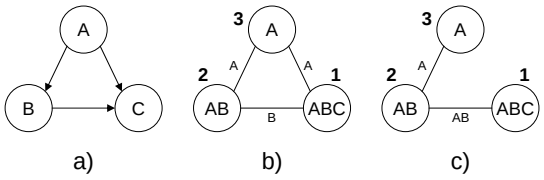

Figure 6.13: a) A belief network; b) A dual join-graph with singleton labels; c) A dual join-graph which is a join-tree

Proposition 6.4.6 *The dual graph of any Bayesian network has an arc-minimal dual join-graph where each arc is labeled by a single variable.*

Proof: Consider a topological ordering of the nodes in the directed acyclic graph associated with the Bayesian network $d = X_1, ..., X_n$. We define the following dual join-graph. Every node in the dual graph $\mathcal{D}$, associated with p_i is connected to node p_j, $j < i$ if $X_j \in pa_i$. We label the arc between p_j and p_i by variable X_j, namely $l_{ij} = \{X_j\}$. It is easy to see that the resulting arc-labeled subgraph of the dual graph satisfies connectedness. (Take the original acyclic graph G and add to each node its CPT family, namely all the other parents that precede it in the ordering. Since G already satisfies connectedness so is the arc-minimal graph generated.) The resulting labeled graph is a dual graph with singleton labels. ∎

Example 6.4.7 Consider the acyclic directed graph on 3 variables A, B, C that corresponds to a Bayesian networks having the CPTs 1) $P(C|A, B)$, 2) $P(B|A)$ and 3) $P(A)$, given in Figure 6.13a. Figure 6.13b shows a dual graph with singleton labels on the arcs. Figure 6.13c shows a dual graph which is a join tree. In one dual graph we have the arcs between CPTs $(1, 2)$ labeled B, between CPTs $(2, 3)$ labeled A and between CPTs $(1, 3)$ labeled A. This is the dual graph that has singleton labels. Another dual graph is a tree with only 2 arcs: between CPTs $(1, 2)$ labeled AB and between CPTs $(2, 3)$ labeled by A. This later one is a join-tree on which belief propagation can solve the problem exactly in 2 iterations. □

Algorithm Iterative belief propagation (IBP) is given in Figure 6.14. It is easy to see that one iteration of IBP is time and space linear in the size of the belief network, and when IBP is applied to the singleton labeled dual graph it coincides with Pearl's belief

Algorithm **IBP**
Input: An arc-labeled dual join-graph $DJ = (V, E, L)$ for a graphical model $\mathcal{M} =< X, D, F, \prod >$.
Output: An augmented graph whose nodes include the original functions and the messages received from neighbors. Denote by: h_u^v the message from u to v; $ne(u)$ the neighbors of u in V; $ne_v(u) = ne(u) - \{v\}$; l_{uv} the label of $(u, v) \in E$; $elim(u, v) = scope(u) - sep(u, v)$.
• **One iteration of IBP**
For every node u in DJ in a topological order and back, do:
1. **Process observed variables**
Assign evidence variables to the each p_i and remove them from the labeled arcs.
2. **Compute and send to v the function:**

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

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

Figure 6.14: Algorithm Iterative Belief Propagation

propagation which was presented as a message-passing applied directly to the acyclic graph representation with one change. The message were normalized. Clearly when the dual join-graph is a tree IBP converges after one iteration (two passes, up and down the tree) to the exact representation, from which beliefs can be computed.

6.4.2 the semantic of the polytree messages

(to be completed)

6.5 Combining Elimination and Conditioning

A serious drawback of elimination and clustering algorithms is that they require considerable memory for recording the intermediate functions. We already observed that when variables are assigned values this help reduce the computation by reducing the induced-width. Cutset conditioning is a scheme that exploit this property in a systematic way.

We can select a subset of variables, assign them values (i.e., condition on them) and solve the remaining problem by inference. This yield a conditioning-based decomposition of the problem into a collection of easier problems which all need to be solved. This set of conditioned problems can be traversed systematically be a search algorithm, yielding a scheme called *conditioning search* or *cutset conditioning* scheme. The nice thing about conditioning search is that it requires only linear space. By combining conditioning and elimination, we may be able to reduce the amount of memory needed while still having performance guarantee.

Full conditioning for probabilistic networks is brute-force search, namely, traversing the tree of partial value assignments and accumulating the appropriate sums of probabilities. (It can be viewed as an algorithm for processing the algebraic expressions from left to right, rather than from right to left as was demonstrated for elimination). For example, we can compute the expression for belief updating or the probability of evidence in the network of Figure 2.5:

$$Bel(A=a) = \sum_{c,b,f,d,g} P(g|f)P(f|b,c)P(d|a,b)P(c|a)P(b|a)P(a)$$

$$= P(a)\sum_{c} P(c|a)\sum_{b} P(b|a)\sum_{f} P(f|b,c)\sum_{d} P(d|b,a)\sum_{g} P(g|f), \qquad (6.1)$$

by traversing the tree in Figure 6.15, going along the ordering from first variable to last variable.

The tree can be traversed either breadth-first or depth-first resulting in algorithms such as best-first search and branch and bound, respectively. The sum can be accumulated for each value of variable A.

Notation: Let X be a subset of variables and $V = v$ be a value assignment to V. $f(X)|_v$ denotes the function f where the arguments in $X \cap V$ are assigned the corresponding values in v.

Let C be a subset of conditioned variables, $C \subseteq X$, and $V = X - C$. We denote by v an assignment to V and by c an assignment to C. Obviously,

$$\sum_{x} P(x,e) = \sum_{c}\sum_{v} P(c,v,e) = \sum_{c,v} \Pi_i P(x_i|x_{pa_i})|_{(c,v,e)}$$

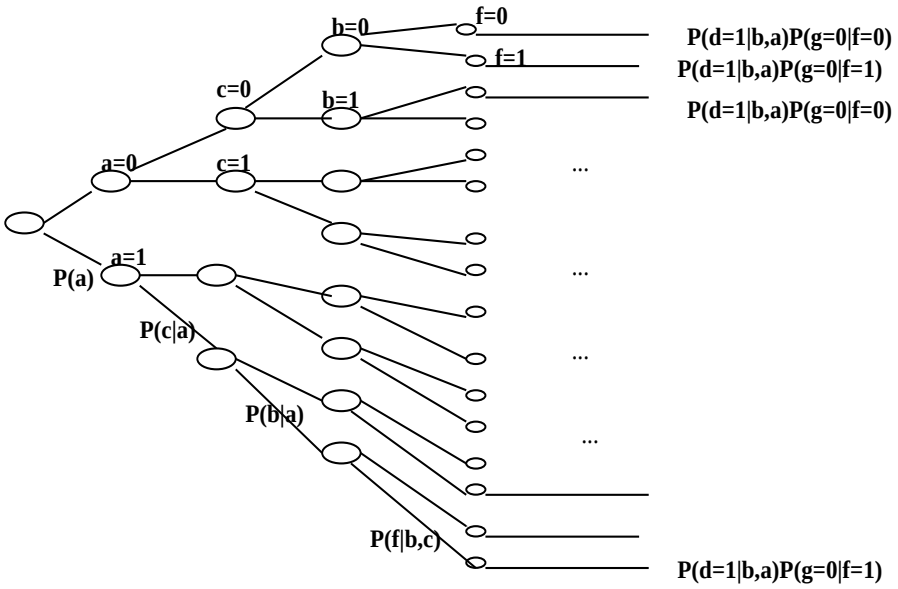

Figure 6.15: probability tree

Therefore, for every partial tuple c, we can compute $\sum_v P(v, c, e)$ using bucket elimination, while treating the conditioned variables as observed variables. This basic computation will be enumerated for all value combinations of the conditioned variables, and the sum will be accumulated. This straightforward algorithm is presented in Figure 6.16.

Given a particular value assignment c, the time and space complexity of computing the probability over the rest of the variables is bounded exponentially by the induced width of the ordered moral graph along d adjusted for both observed and conditioned nodes, denoted $w^*(d, e \cup c)$. Therefore, the induced graph is generated without connecting earlier neighbors of both evidence and conditioned variables.

Theorem 6.5.1 *Given a set of conditioning variables, C, the space complexity of algorithm vec-bel is $O(n \cdot exp(w^*(d, c \cup e))$, while its time complexity is $O(n \cdot exp(w^*(d, e \cup c) + |C|))$, where the induced width $w^*(d, c \cup e)$, is computed on the ordered moral graph that was adjusted relative to e and c.* □

When the variables in $e \cup c$ constitute a cycle-cutset of the primal, or moral graph, the graph can be ordered so that its adjusted induced width equals 1 and vec-bel reduces to the well known cycle-cutset algorithm [12], or if the cutset yield a polytree we get the well know loop-cutset algorithm [33].

Algorithm VEC-bel
Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, d; a subset C of conditioned variables; observations e.
Output: $Bel(A) = P(A|e)$.
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** λ.

Figure 6.16: Algorithm *vec-bel*

Definition 6.5.2 *Given an undirected graph, G a cycle-cutset is a subset of the nodes that breaks all its cycles. Namely, when removed, the graph has no cycles. A cutset is called* loop-cutset *if it removal from a directed graph generates a polytree*

In general Theorem 6.5.1 calls for a secondary optimization task on graphs:

Definition 6.5.3 (secondary-optimization task) *Given a graph $G = (V, E)$ and a constant r, find a smallest subset of nodes C_r, such that $G\prime = (V - C_r, E\prime)$, where $E\prime$ includes all the edgs in E that are not incident to nodes in C_r, has induced-width less or equal r.*

Clearly, the minimal cycle-cutset corresponds to the case where the induced-width is $r = 1$. The loop-cutset corresponds to the case when conditioning creates a poly-tree. The general task is clearly NP-complete.

Clearly, algorithm vec-bel can be implemented more effectively if we take advantage of shared partial assignments to the conditioned variables. There is a variety of possible hybrids between conditioning and elimination that can refine this basic procedure. One method imposes an upper bound on the scope of functions recorded and decides dynamically, during processing, whether to process a bucket by elimination or by conditioning Another method which uses the super-bucket approach collects a set of consecutive buckets into one super-bucket that it processes by conditioning, thus avoiding recording some intermediate results

Bibliography

[1] Darwiche A. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[2] S. Arnborg and A. Proskourowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

[3] S. A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

[4] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Uncertainty in AI (UAI'96)*, pages 81–89, 1996.

[5] E. Bensana, M. Lemaitre, and G. Verfaillie. Earth observation satellite management. *Constraints*, 4(3):293–299, 1999.

[6] U. Bertele and F. Brioschi. *Nonserial Dynamic Programming*. Academic Press, 1972.

[7] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the Association of Computing Machinery*, 44, No. 2:165–201, 1997.

[8] C. Cannings, E.A. Thompson, and H.H. Skolnick. Probability functions on complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978.

[9] R. McEliece D. C. MacKay and J. Cheng. Turbo decoding as an instance of pearl's "belief propagation" algorithm. 1996.

[10] S. de Givry, J. Larrosa, and T. Schiex. Solving max-sat as weighted csp. *In Principles and Practice of Constraint Programming (CP-2003)*, 2003.

[11] S. de Givry, I. Palhiere, Z. Vitezica, and T. Schiex. Mendelian error detection in complex pedigree using weighted constraint satisfaction techniques. In *ICLP Workshop on Constraint Based Methods for Bioinformatics*, 2005.

[12] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[13] R. Dechter. Mini-buckets: A general scheme of generating approximations in automated reasoning. In *IJCAI-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1297–1302, 1997.

[14] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[15] R. Dechter and D. Larkin. Hybrid processing of belief and constraints. *Proceeding of Uncertainty in Artificial Intelligence (UAI01)*, pages 112–119, 2001.

[16] R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[17] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

[18] R. Dechter and P. van Beek. Local and global relational consistency. In *Principles and Practice of Constraint programming (CP-95)*, pages 240–257, 1995.

[19] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, pages 283–308, 1997.

[20] E. C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

[21] M. R Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. In *W. H. Freeman and Company, San Francisco*, 1979.

[22] F.V. Jensen. *Bayesian networks and decision graphs*. Springer-Verlag, New-York, 2001.

[23] U. Kjæaerulff. Triangulation of graph-based algorithms giving small total state space. In *Technical Report 90-09, Department of Mathematics and computer Science, University of Aalborg, Denmark*, 1990.

[24] U. Kjæaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence (UAI'93)*, pages 121–149, 1993.

[25] D. Koller and N. Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

[26] F. R. Kschischang and B.H. Frey. Iterative decoding of compound codes by probability propagation in graphical models. *submitted*, 1996.

[27] S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50(2):157–224, 1988.

[28] D. Maier. The theory of relational databases. In *Computer Science Press, Rockville, MD*, 1983.

[29] R.J. McEliece, D.J.C. MacKay, and J.-F.Cheng. Turbo decoding as an instance of Pearl's belief propagation algorithm. *To appear in IEEE J. Selected Areas in Communication*, 1997.

[30] L. G. Mitten. Composition principles for the synthesis of optimal multistage processes. *Operations Research*, 12:610–619, 1964.

[31] R.E. Neapolitan. *Learning Bayesian Networks*. Prentice hall series in Artificial Intelligence, 2000.

[32] Jurg Ott. *Analysis of Human Genetics*. Cambridge University Press, 1999.

[33] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[34] A. Dechter R. Dechter and J. Pearl. Optimization in constraint networks. In *Influence Diagrams, Belief Nets and Decision Analysis*, pages 411–425. John Wiley & Sons, 1990.

[35] B. D'Ambrosio R.D. Shachter and B.A. Del Favero. Symbolic probabilistic inference in belief networks. In *National Conference on Artificial Intelligence (AAAI'90)*, pages 126–131, 1990.

[36] R.G.Gallager. A simple derivation of the coding theorem and some applications. *IEEE Trans. Information Theory*, IT-11:3–18, 1965.

[37] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *Proc. IJCAI-99*, pages 542–547, 1999.

[38] L. K. Saul and M. I. Jordan. Learning in boltzmann trees. *Neural Computation*, 6:1173–1183, 1994.

[39] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423,623–656, 1948.

[40] P.P. Shenoy. Valuation-based systems for bayesian decision analysis. *Operations Research*, 40:463–484, 1992.

[41] K. Shoiket and D. Geiger. A proctical algorithm for finding optimal triangulations. In *Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 185–190, 1997.

[42] C.E. Leiserson T. H. Cormen and R.L. Rivest. In *Introduction to algorithms*. The MIT Press, 1990.

[43] R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs and selectively reduce acyclic hypergraphs. *SIAM Journal of Computation.*, 13(3):566–579, 1984.

[44] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, pages 365–379, 1990.

[45] P. Thbault, S. de Givry, T. Schiex, and C. Gaspin. Combining constraint processing and pattern matching to describe and locate structured motifs in genomic sequences. In *Fifth IJCAI-05 Workshop on Modelling and Solving Problems with Constraints*, 2005.

[46] N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research (JAIR)*, 1996.