Unifying Tree-Decomposition Schemes for Automated Reasoning

Keywords : Automated Reasoning

Abstract

The paper provides a unifying perspective of tree-decomposition algorithms appearing in various automated reasoning areas, such as join-tree clustering for constraint-satisfaction and the clique-tree algorithm for probabilistic reasoning. Subsequently, the paper extends the variable-elimination scheme called bucket-elimination (BE) into a two-phase message passing along a bucket-tree (BTE), making it another instance of tree-decomposition. Our analysis shows that the new algorithm BTE may provide a substantial speed-up over BE for important reasoning tasks. Time-space tradeoffs are cast within the tree-decomposition framework.

1 Introduction

The paper introduces a general tree-decomposition framework for solving a wide range of automated reasoning tasks. We show that many existing decomposition schemes, such as join-tree clustering, junction-tree decompositions, and hyper-tree decomposition, are instances of tree-decomposition. We then introduce a new algorithm, called *bucket-tree elimination (BTE)*, based on the Bucket Elimination technique [Dechter, 1999] and show that it is an instance of tree-decomposition.

The unifying framework provides clarity which is likely to encourage technology transfer. In particular, a recent approximation scheme for Bucket-Elimination algorithms, called *mini-bucket elimination*, can now be extended to general tree-decompositions, as we show in a companion paper [anonymous, 2001].

Section 2 defines the automated reasoning task and related background concepts. Section 3 introduces the tree-decomposition framework and its associated algorithm. Section 4 introduces and analyzes the bucket-tree elimination algorithm, and Section 5 relates existing decomposition methods within the context of tree-decomposition. Section 6 discusses time-space tradeoffs, Section 8 provides related work and concludes.

2 Automated reasoning tasks

Definition **2.1** *An* automated reasoning task P *is a sixtuple* $P =< X, D, F, \bigotimes, \Downarrow, \{Z_1, ..., Z_t\} >$ *defined as follows:*

1. $X = \{1, ..., n\}$ *is a set of variables.*
2. $D = \{D_1, ..., D_n\}$ *is a set of finite domains.*
3. $F = \{f_1, ..., f_r\}$ *is a set of functions. The scope of function* f_i*, denoted* $scope(f_i) \subseteq X$*, is the set of arguments of* f_i*.*
4. $\bigotimes_i f_i \in \{\prod_i f_i, \sum_i f_i, \bowtie_i f_i\}$ *is a combination operator.*
5. $\Downarrow_Y f \in \{ \max_{S-Y} f, \min_{S-Y} f, \prod_{S-Y} f, \sum_{S-Y} f\}$*, where* S *is the scope of function* f *and* $Y \subseteq X$ *is a marginalization operator.*
6. *The problem is to compute,* $\forall \; Z_i$ $\Downarrow_{Z_1} \bigotimes_{i=1}^r f_i, ..., \Downarrow_{Z_t} \bigotimes_{i=1}^r f_i$

Definition **2.2** *The primal graph of a problem* P *has the variables as its nodes, and two nodes are connected if they appear in a scope of a function in* F*. The hypergraph of a problem* P *has the variables as its nodes and the scopes of functions as its hyperedges.*

Definition **2.3 (graph concepts)** *An* ordered graph *is a pair* (G, d) *(also denoted* G_d*), where* G *is an undirected graph and* $d = X_1, ..., X_n$ *is an ordering of the nodes. The* width of a node *in an ordered graph is the number of its earlier neighbors, while the* width of an ordering d, $w(d)$, *is the maximum width over all nodes. In an ordered graph, the* induced width, $w^*(d)$, *is the width of the induced ordered graph obtained by processing the nodes recursively, from last to first. When node* X *is processed, all its earlier neighbors are connected.*

Examples of reasoning tasks

Probabilistic Inference Queries over Bayesian networks [Pearl, 1988] can be formulated as automated reasoning tasks where the functions in F denote conditional probability table and the scopes of these functions is determined by a directed acyclic graph (DAG): Each function f_i ranges over variable i and its parents in the dag. The primal graph of a Bayesian network is its moral graph.

- **Belief-updating** is the task of computing belief in variable y in Bayesian networks. For this task, the combination operator is $\bigotimes_j = \prod_j$ and the marginalization operator is $\Downarrow_y = \sum_{X-y}$.
- **Most probable explanation** requires computing the most probable tuple in a given Bayesian network. Here the combination operator is $\bigotimes_j = \prod_j$ and marginalization operator is $\Downarrow_\emptyset = \max_X$.

Constraint Satisfaction and Optimization
For CSPs, the functions in F are deterministic relations over subsets of variables. In constraint optimization, the functions in F are real-valued cost functions. The primal graph is the constraint graph.

- **Deciding consistency of a CSP** requires determining if a constraint satisfaction problem has a solution and, if so, to find all its solutions. Here the combination operator is $\bigotimes_j = \bowtie_j$ and the marginalization operator is $\Downarrow_\emptyset = \Pi_X$.
- **Max-CSP, Combinatorial optimization** Max-CSP problems seek to find a solution that minimizes the number of constraints violated. Combinatorial optimization assumes real cost functions in F. Both tasks can be formalized using the combination operator $\bigotimes_j = \sum_j$ and the marginalization operator is $\Downarrow_\emptyset = \min_X$ (the constraints can be expressed as cost functions of cost 0, or 1).

3 Tree-Decomposition schemes

This section introduces a unifying tree-decomposition framework. The exposition is declarative, separating the desired target output from its generative process.

DEFINITION **3.1** *Let $P =< X, D, F, \bigotimes, \Downarrow, \{Z_i\} >$ be an automated reasoning problem. A* tree-decomposition *for P 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 $\psi(v) \subseteq F$*[1]*, that satisfy the following conditions:*

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 $x \in X$, the set $\{v \in V | x \in \chi(v)\}$ induces a connected subtree of T. This is also called the running intersection or connectedness property.*
3. *$\forall\ i\ Z_i \subseteq \chi(v)$ for some $v \in T$.*

When the combination operator is join, as in constraint satisfaction, Condition 1 can be relaxed to require that each function will be in *at least* one node, thus allowing multiple appearances of a function in nodes.

DEFINITION **3.2 (tree-width, hyper-width, separator)** *The width (also called tree-width) of a tree-decomposition $< T, \chi, \psi >$ is $\max_{v \in V} |\chi(v)|$, and its hyper-width is $\max_{v \in V} |\psi(v)|$. Given two adjacent vertices u and v of a tree-decomposition, a separator of u and v is defined as $sep(u, v) = \chi(u) \cap \chi(v)$.*

Algorithm tree-elimination (CTE)
Input: A tree decomposition $< T, \chi, \psi >$ for a problem $P =< X, D, F, \bigotimes, \Downarrow, \{Z_1, ...Z_t\} >$.
Output: An augmented tree whose nodes 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 cluster tree, do

- Let (u, v) be an edge in the cluster tree such that vertex u has received messages from all adjacent vertices other than v.
- Let $m_{(i,u)}$ denote the message sent by vertex i to vertex u. Compute:

$$m_{(u,v)} = \Downarrow_{sep(u,v)} \Big(\bigotimes_{f \in cluster(u), f \neq m_{(v,u)}} f \Big)$$

where $cluster(u) = \psi(u) \cup \{m_{(v,u)} | (v, u) \in T\}$

Compute solution: For every $v \in T$ and every $Z_i \subseteq \chi(v)$, compute $\Downarrow_{Z_i} \bigotimes_{f \in cluster(v)} f$.

Figure 1: Algorithm cluster-tree elimination (CTE)

Example 3.1 *Consider a problem P over variables A, B, C, D, F, G with functions over scopes of size 2: $F = \{f(A, B), f(A, C), f(B, C), f(B, F), f(C, F), F(A, B, D)\ F(F, G)\}$. Figure 2b gives its primal graph. Any of the trees in Figure 6 is a tree-decomposition for the problem where the functions can be partitioned into clusters that contain their scopes.*

A tree-decomposition facilitates a solution to an automated reasoning task. Algorithm cluster-tree elimination for processing a tree-decomposition is given in Figure 1. It works by having each vertex of the tree send a function to each of its neighbors. If the tree contains m edges, then a total of $2m$ messages will be sent. Node u takes all the functions in node u and all messages received by u from all adjacent nodes other than v, joins them using the combination operator and projects the combined function onto the separator of u and v using the marginalization operator. The projected function is then sent to v.

Node activation can be asynchronous. Convergence is guaranteed, but it may take as long as the diameter of the tree in the worst case. 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.

Once all nodes have received messages from all neighbors, a solution to the problem can be generated using the output augmented tree (as described in the algorithm), in output linear time. For some tasks the whole

[1]We follow the notation provided in [Georg Gottlob and Scarello, 1999].

output tree is used to compute the solution (e.g., computing optimal tuple).

THEOREM **3.2 (Correctness and completeness)** *Algorithm CTE is sound and complete.*

THEOREM **3.3 (Complexity)** *Let N be the number of nodes in the tree decomposition, w be its tree-width, sep be its maximum separator size, r be the number of input functions in F, and deg be the maximum degree in T. The time complexity of CTE is $O((r+N) \cdot deg \cdot exp(w))$ and its space complexity is $O(N \cdot exp(sep))$.*

4 Bucket-Tree Elimination

In this section we first extend the bucket-elimination scheme into a message passing algorithm along a bucket-tree. We then show that a bucket-tree is an instance of tree-decomposition and that the extended algorithm coincides with CTE.

Bucket-elimination (BE) is a unifying algorithmic framework for dynamic-programming algorithms applicable to probabilistic and deterministic reasoning [Bertele and Brioschi, 1972; Dechter, 1999]. The input to a BE algorithm consists of a collection of functions or relations (e.g., clauses for propositional satisfiability, constraints, or conditional probability matrices for belief networks). Given a variable ordering, the algorithm partitions functions into buckets, each associated with a single variable. A function is placed in the bucket of its latest argument in the ordering. The algorithm processes each bucket, top-down, from the last variable to the first, by a variable elimination procedure that computes a new function using combination and marginalization operators. The new function is placed in the closests lower bucket whose variable appears in the new function's scope.

When the solution of the problem requires a complete assignment (e.g., solving the most probable explanation (mpe) problem in Bayesian networks), a second, bottom-up phase, assigns a value to each variable along the ordering, consulting the functions created during the top-down phase. For more details see [Dechter, 1999].

Some tasks, however, require repeated execution of the BE algorithm, for example, when a belief for every variable in a Bayesian network is required. Another example is computing the optimal cost associated with each value of each variable, to guide search algorithms. In order to compute belief for every variable, BE would have to be run n times, each initiated by a different variable. We next propose a more efficient approach, by extending bucket-elimination into a bucket-tree elimination scheme, called BTE.

The idea is based on a recent result in the context of belief updating. It is known that BE can be viewed as message-passing from leaves to root along a bucket-tree [Dechter99]. A generalized elimination scheme was recently developed by Cozman [Cozman, 2000] in the context of probabilistic inference, where a second pass along the bucket tree can update every bucket in the tree. Here this scheme is derived and analyzed in a more general and abstract setting.

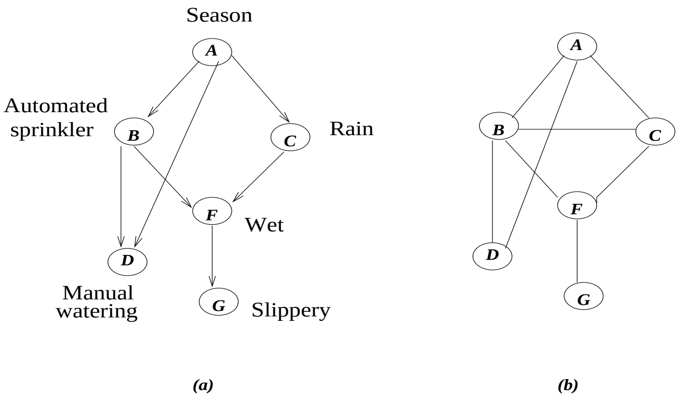

Figure 2: belief network $P(g,f,d,c,b,a) = P(g|f)P(f|c,b)P(d|b,a)P(b|a)P(c|a)P(a)$

We next present this idea for any automated reasoning task, show that the BTE algorithm is an instance of tree-decomposition, and derive correctness and complexity from this relationship.

DEFINITION **4.1 (buckets)** *Let $P =< X, D, F, \bigotimes, \Downarrow>$ be an automated reasoning problem and d an ordering of its variables $d = (x_1, ..., x_n)$. Let $B_{x_1}, ..., B_{x_n}$ be a set of buckets, one for each variable. Each bucket B_{x_i} contains those functions in F whose latest variable in d is x_i.*

DEFINITION **4.2 (bucket-tree)** *A* bucket-tree *of P and an ordering d, has buckets as its nodes, and 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_i} are those appearing in the scopes of any of its new and old functions.*

Therefore, in a bucket tree, every node B_x has one parent node B_y and several child nodes $B_{z_1}, ...B_{z_t}$. The structure of the bucket-tree can also be extracted from the induced-ordered graph of P along d using the following equivalent definition.

DEFINITION **4.3 (bucket tree, graph-based)** *Let G_d be the induced graph along d of a reasoning problem P whose primal graph is G. Each variable x and its earlier neighbors in the induced-graph is a variable in bucket B_x. The nodes of the bucket-tree are the n buckets. Each node B_x points to B_y (or, B_y is the parent of B_x) if y is the latest earlier neighbor of x in G_d. 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 $B_x \cap B_y$.*

Example 4.1 *Consider the Bayesian network defined over the DAG in Figure 2a. Figure 4 left shows the initial buckets along the ordering $d = A, B, C, D, F, G$, and the λ messages that will be passed by BE from top to bottom. On its right, the figure displays the same computation as a message-passing along its bucket-tree.*

THEOREM **4.2** *A bucket tree of a reasoning problem P is a tree-decomposition of P.*

Since the bucket-tree is a tree-decomposition, the cluster-tree elimination algorithm CTE is applicable.

Algorithm bucket-tree elimination (BTE)
Input: A problem $P =< X, D, F, \bigotimes, \Downarrow, \{x_1, ..., x_n\} >$, ordering d.
Output: Augmented buckets containing the original functions and all the π and λ functions received from neighbors in the bucket-tree. A solution to P computed from augmented buckets.
0. Pre-processing:
Place each function in the latest bucket, along d, that mentions a variable in its scope. Connect two buckets B_x and B_y if variable y is the lastest earlier neighbor of x in the induced graph G_d. **1. Bottom-up phase: λ messages** (BE)
For $i = n$ to 1, process bucket B_{x_i}:
Let $\lambda_1, ...\lambda_j$ be all the functions in B_{x_i} at the time B_{x_i} is processed, including original functions of P. The message $\lambda_{x_i}^y$ sent from x_i to its parent y, is computed by

$$\lambda_{x_i}^y(sep(x_i, y)) = \Downarrow_{sep(x_i,y)} \bigotimes_{i=1}^{j} \lambda_i$$

where $sep(x_i, y)$ is the separator of x_i and y.
2. Top-down phase: π messages
For $i = 1$ to n, process bucket B_{x_i}:
Let $\lambda_1, ...\lambda_j$ be all the functions in B_{x_i} at the time B_{x_i} is processed, including the original functions of P. B_{x_i} takes the π message received from its parent y, $\pi_y^{x_i}$, and computes a message $\pi_{x_i}^{z_j}$ for each child bucket z_j by

$$\pi_{x_i}^{z_j}(sep(x_i, z_j)) = \Downarrow_{sep(x_i,z_j)} \pi_y^{x_i} \bigotimes (\bigotimes_i \lambda_i / \lambda_{z_j}^{x_i})$$

3. Compute solution: In each augmented bucket compute: $\Downarrow_{x_i} \bigotimes_{f \in bucket_i} f$,

Figure 3: Algorithm Bucket-Tree Elimination

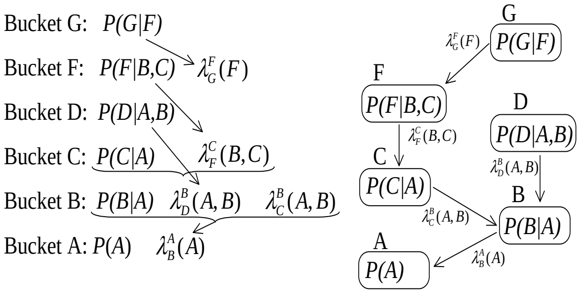

Figure 4: Execution of BE along the bucket-tree

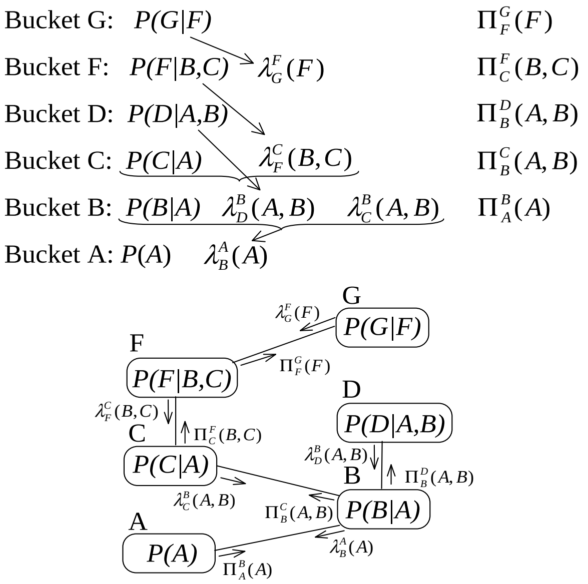

Figure 5: Propagation of πs and λs along the bucket-tree

Indeed, as we show, the correctness of the extension of BE to BTE that adds a bottom-up message passing is established by showing equivalence with CTE when applied to the problem's bucket-tree. To present the algorithm, we will use two types of messages, λs and πs as common in the exposition of the bucket-elimination scheme.

Algorithm *bucket-tree elimination* (BTE) is given in Figure 3. In the top-down phase, each bucket receives λ messages from its children and sends a λ message to its parent. This portion is equivalent to BE. In the bottom-up phase, each bucket receives a π message from its parent and sends π messages to each child.

Example 4.3 *Figure 5 shows the complete execution of BTE along the linear order of buckets and along the bucket-tree. The π and λ messages are viewed as messages placed on the outgoing arcs.*

THEOREM **4.4** *Algorithm BTE is sound and complete*

4.1 Complexity

Clearly, the induced-width w^* along d is identical to the tree-width of the bucket-tree when viewed as a tree-decomposition. We next provide a refined complexity analysis of BE followed by complexity analysis of BTE.

THEOREM **4.5 (Complexity of BE)** *Let w^* be the induced width of G along ordering d and sep its maximum separator. The time complexity of BE is $O(r \cdot exp(w^* + 1))$ and its space complexity is $O(n \cdot exp(sep))$.*

THEOREM **4.6 (Complexity of BTE)** *Let w^* be the induced width of G along ordering d and sep its maximum spearator ($sep \leq w^*$). The time complexity of BTE is $O(r \cdot deg \cdot exp(w^* + 1))$, where deg is the maximum degree in the bucket-tree. The space complexity of BTE is $O(n \cdot exp(sep))$.*

In theory the speedup expected from running BTE vs running BE n times (n-BE) is at most n. This may seem insignificant compared with the exponential complexity in w^*, however in practice it can be very significant. In particular, when these computations are used as a procedure within more extensive search algorithms [Kask and Dechter, 1999]. The actual speedup of BTE relative to n-BE may be smaller than n, however. We know that the complexity of n-BE is $O(n \cdot r \cdot exp(w^* + 1))$, whereas the complexity of running BTE is $O(deg \cdot r \cdot exp(w^* + 1))$. These two bounds cannot be directly compared because we do not know how tight the n-BE bound is. We can hypothesize as follows: If the complexity of n-BE was $\Theta(n \cdot r \cdot exp(w^* + 1))$, then the speeup of BTE over n-BE would be $\Omega(n/deg)$. In a companion paper [anonymous, 2001] we evaluate empirically the speed-up of an approximation scheme based on BTE that show substantial gains.

5 Relating tree-decomposition methods

5.1 Join-Tree Clustering

In both constraint satisfaction and in Bayesian network's communities the most used clustering methods called join-tree clustering ([Dechter and Pearl, 1989; Lauritzen and Spiegelhalter, 1988]) are based on a triangulation algorithm which transforms the primal graph $G = (V, E)$ of a problem instance P into a chordal graph G'. Therefore, a join-tree clustering is a tree $T = (V, E)$, where V is a set of maximal cliques of G' and E is a set of edges that form a tree between cliques satisfying the connectedness property [Maier, 1983]. The width of a join-tree clustering is the cardinality of its maximal clique, which coincides with the induced-width (plus 1), along the order of triangulation. Subsequently, every function is placed in one clique containing its scope.

It is easy to see that a join-tree satisfies the properties of tree-decomposition.

Proposition 1 *Every join-tree clustering is a tree-decomposition.* □

Join-trees correspond to minimal tree-decompositions only, where separators are always strict subsets of their adjacent clusters, thus excluding some decompositions that can be useful (see [Georg Gottlob and Scarello, 1999]). Moreover, they are *cluster-minimal*; no node and its variables can be partitioned further to yield a more refined tree-decomposition.

Example 5.1 *Consider a problem having functions defined on all pairs of variables whose graph is complete. Clearly, the only possible join-tree will have one node containing all the variables and all the functions. An alternative tree-decomposition has node C_1 whose variables are $\{1, ..., n\}$ and whose functions are defined over the pairs of variables: $\{(1,2)(3,4), ...(i, i+1)(i+2, i+3)....\}$. Then, there is a node, $C_{i,j}$, for each other function that is not contained in C_1, and the tree connects C_1 with each other node. While this is a legitimate tree-decomposition, it is not a legitimate join-tree. This example is an instance of a hyper-tree decomposition, discussed next.*

5.2 Hypertree Decomposition

Recently, Gottlob et.al [Georg Gottlob and Scarello, 1999] presented the notion of hyper-tree decompositions for Constraint Satisfaction, and showed that for CSPs the hyper-width parameter can capture tractable classes that are not captured by tree-width. The exposition in [Georg Gottlob and Scarello, 1999] of hypertree-decomposition, as is, is *not* an instance of tree-decomposition because it allows a function to label more than a single node in the tree. While this will not hurt the solution of constraint problems it is not legal for the general case. Since our interest is in general reasoning tasks, tree-decomposition must be restricted. We will therefore augment the definition of hypertree-decomposition in [Georg Gottlob and Scarello, 1999] with a restriction, and will show that such a restricted hypertree-decomposition is an instance of tree-decomposition.

A *hypertree for a hypergraph H* [Georg Gottlob and Scarello, 1999] is a triple $< T, \chi, \psi >$, where $T = (N, E)$ is a rooted tree, and χ and ψ are labeling functions which associate with each node $p \in N$ two sets $\chi(p) \subseteq scope(H)$ and $\psi(p) \subseteq edges(H)$. If $T' = (N', E')$ is subtree of T, we define $\chi(T') = \cup_{v \in N'} \chi(v)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

DEFINITION **5.1** *[Georg Gottlob and Scarello, 1999] A (restricted) complete hypertree decomposition of a hypergraph H is a hypertree $< T, \chi, \psi >$ for H which satisfies the following conditions:*

1. *For each edge $h \in edges(H)$, there exists $p \in vertices(H)$ such that $h \in \psi(p)$and $scope(h) \subseteq \chi(p)$ (we say that p strongly covers h);*
2. *For each variable $x \in scope(H)$, the set $\{p \in vertices(T) | x \in \chi(p)\}$ induces a (connected) subtree of T.*
3. *For each $p \in vertices(H)$, $\chi(p) \subseteq scope(\psi(p))$.*
4. *For each $p \in vertices(T)$, $scope(\psi(p)) \cap \chi(T_p) \subseteq \chi(p)$.*

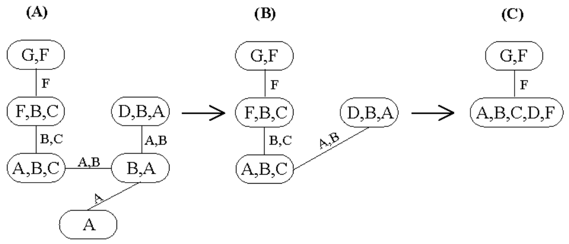

Figure 6: From a bucket-tree (left) to join-tree (middle) to a super-bucket-tree (right)

5. *(the restricting condition) For every $h \in H$ there is exactly one $p \in vertices(T)$ s.t. $h \in \psi(p)$.*

Conditions 1-4 correspond to complete hypertree-decompositions in [Georg Gottlob and Scarello, 1999].

DEFINITION **5.2** *A complete hypertree-decomposition of a reasoning problem P is obtained from a hypertree-decomposition of its hypergraph by replacing hyperedges with the functions having the hyperedges as their scope.*

Proposition 2 *Any (restricted) complete hypertree decomposition of P is a tree-decomposition of P.* □

Notice that the opposite is not true. There are tree-decompositions that are not (restricted) complete hypertree decompositions, because hypertree decompositions require that the variables labeling a node will be contained in the scope of its labeling functions.

For example, consider a single n-ary function f. It can be mapped into a bucket-tree with n nodes. Node i contains all variables $\{1, 2, ...i\}$ but no functions, while node n contains all the variables and the input function. Both join-tree and hyper-tree decomposition will allow just one node that include the function and all its variables.

6 Space-Time Tradeoff : Superbuckets

The main drawback of CTE is its memory needs. The space complexity of CTE is exponential in the largest separator size. In practice this may be too prohibitive and therefore time-space tradeoffs were introduced [Dechter, 1996]. The idea is to trade space for time by combining adjacent nodes, thus reducing separator sizes, while increasing theire width and the hyper-width.

Proposition 3 *If T is a tree-decomposition, then any tree obtained by merging adjacent nodes in T, is a tree-decomposition.* □

Since a bucket tree is a tree-decomposition, by merging adjacent buckets, we get what we call a *super-bucket-tree* (SBT). This means that in the top-down phase of processing SBT, several variables are eliminated at once. Note that one can always generate a join-tree from a bucket-tree by merging adjacent nodes. For illustration see Figure 6.

7 Related work and conclusions

By its nature the work here is related to all the work in the past two decades on tree-docmpositions for specific tasks, to which we referred sporadically throughout the paper. Unifying framework were also presented [Shenoy, 1992; Bistarelli *et al.*, 1997]. The work here put all these schemes and formalisms together.

The work presented here has two novelties. First, it provides a unifying framework for tree-decomposition that draws on notations and formalizations that appear in wide sources and in diverse communities, such as probabilistic reasoning, optimization, constraint satisfaction and graph theory. We believe that the current exposition add clarity and will allow technology transfer.

The second novelty is extending the general variable-elimination algorithm called bucket elimination, into a two phase algorithm along a bucket-tree making explicit the connection between these type of algorithms and tree-decompositions. The extension is important for a variety of reasoning tasks. The correctness and complexity of the involving algorithms is analyzed.

References

[anonymous, 2001] anonymous. Up and down mini-bucket: a scheme for approximating combinatorial optimization tasks. *Submitted to Ijcai-2001*, 2001.

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

[Bistarelli *et al.*, 1997] 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.

[Cozman, 2000] F. G. Cozman. Generalizing variable-elimination in bayesian networks. In *Workshop on Probabilistic reasoning in Bayesian networks at SBIA/Iberamia 2000*, pages 21–26, 2000.

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

[Dechter, 1996] R. Dechter. Topological parameters for time-space tradeoffs. In *Uncertainty in Artificial Intelligence (UAI'96)*, pages 220–227, 1996.

[Dechter, 1999] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[Georg Gottlob and Scarello, 1999] Nicola Leone Georg Gottlob and Francesco Scarello. A comparison of structural csp decomposition methods. *Ijcai 1999*, 1999.

[Kask and Dechter, 1999] K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. *Proc. IJCAI-99*, 1999.

[Lauritzen and Spiegelhalter, 1988] 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.

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

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

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