# Bucket-Tree Elimination for Automated Reasoning

Kalev Kask, Rina Dechter, Javier Larrosa and Fabio Cozman
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

## Abstract

The paper extends several variable elimination schemes into a two-phase message passing algorithm along a bucket-tree. Our analysis shows that the new algorithm, called Bucket-Tree Elimination (BTE), may provide a substantial speed-up over standard variable-elimination for important probabilistic reasoning tasks. The algorithm is developed and analyzed within a unifying view of tree-clustering methods, making crisp the relationship between the two approaches, and allowing enhancement schemes to be transferred. In particular we show how time-space tradeoffs of BTE are cast within the tree-decomposition framework.

## 1 Introduction

The paper introduces a new algorithm, called *bucket-tree elimination (BTE)*, that extends the Bucket Elimination algorithm [Dechter, 1999] into a message-passing algorithm along a bucket-tree. Furthermore, it shows that BTE can be viewed as an instance of tree-decomposition algorithms appearing in a wide range of automated reasoning tasks.

Bucket-elimination (*BE*) is a unifying algorithmic framework for dynamic-programming algorithms applicable to a wide variety of probabilistic and deterministic reasoning [Bertele and Brioschi, 1972, Dechter, 1999] such as belief updating, finding the most probable explanation (MPE), finding the maximum aposteriori hypothesis (MAP) and finding a collection of decisions that maximize the expected utility in influence diagrams. As well, the scheme is applicable to constraint satisfaction and combinatorial optimization. All bucket-elimination algorithms are close enough, to posses similar performance guarantees, and any single improvement to a single algorithm is immediately applicable to all other algorithms. In particular, an approximation scheme such as the mini-bucket approximation and a variety of time-space tradeoffs are applicable to all algorithms in this framework.

Some important tasks, however are not solved by a single execution of BE, and rather require repeated run of the BE algorithm. One example is belief updating, 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 serve as heuristic guidance of search algorithms. In order to compute such functions for every variable, BE would have to be executed $n$ times, once for each variable.

The bucket-tree elimination algorithm we propose here attempts to overcome this shortcoming. It can sometimes compute the desired unary functions with only twice the time of a single run of BE, thus providing a potential speed-up proportional to the number of variables. The algorithm extends bucket-elimination into a Bucket-Tree Elimination scheme, called *BTE*.

Since BE can be viewed as a one pass message-passing from leaves to root along a tree [Dechter, 1999], an extension to a second pass along the tree, as is typical in tree-clustering algorithms, seems to be warranted. Indeed, such generalized elimination scheme was recently developed by Cozman [Cozman, 2000] for belief updating. The current paper extends Cozman's work, in that it is derived and analyzed in a more general and abstract setting. Specifically, the main contribution of this paper is 1. in extending Bucket-elimination into a bucket-tree elimination scheme for a variety of reasoning tasks and 2. in explicating its relations to tree-decomposition schemes.

To accommodate both aims, the paper presents a unifying, tree-decomposition framework for automated reasoning tasks which captures many existing decomposition schemes, such as join-tree clustering, junction-tree decompositions, and hyper-tree decomposition. We then show that a bucket-tree is a special case of a tree-decomposition yielding the BTE algorithm as a special case of tree-clustering. Correctness and complexity follow therefore from the correctness and complexity of general tree-clustering methods.

Section 2 defines the automated reasoning task and related background concepts. Section 3 presents the tree-decomposition framework and its associated tree-clustering algorithm. Section 4 introduces and analyzes the bucket-tree elimination algorithm, Section 5 relates existing decomposition methods within the context of tree-decomposition. Section 7 discusses time-space tradeoffs and Section 8 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. $D_i$ is the set of values that can be assigned to $X_i$.*
3. *$F = \{f_1, ..., f_r\}$ is a set of functions or relations. Each $f_i$ is defined over a subset of variables $S_i \subseteq X$ and its domain $D_{S_i}$ is the Cartesian products of the domain of variables in $S_i$. The scope of function $f_i$, denoted $scope(f_i) \subseteq X$, is its set of arguments, $S_i$.*
4. *$\bigotimes_i f_i \in \{\prod_i f_i, \sum_i f_i, \bowtie_i f_i\}$ is a combination operator. The scope of $\bigotimes_i f_i$ is $\cup_i S_i$.*
5. *$\Downarrow_Y f \in \{\max_{S-Y} f, \min_{S-Y} f, \Pi_{S-Y} f, \sum_{S-Y} f\}$, where $S$ is the scope of function $f$ and $Y \subseteq X$ is a marginalization operator. The scope of $\Downarrow_Y f_i$ is $Y$.*
6. *The problem is to compute, $\forall$ $Z_i$ $\Downarrow_{Z_1} \bigotimes_{i=1}^r f_i, ..., \Downarrow_{Z_t} \bigotimes_{i=1}^r f_i$*

**Explanation.** We assume that functions are expressed in a tabular form, having an entry for every combination of values from the domains of its variables. Therefore, the specification of such functions is exponential in their scope (the base of the exponent is the maximum domain size). Relations, or clauses can be expressed as functions as well, associating a value of "0" or "1" for each tuple, depending on weather or not the tuple is in the relation (or satisfies a clause). The combination operator takes a set of functions and generates a new function. Note that $\prod_i$ stands for a product when it is a combination operator and $\Pi_i$ for a projection when it is a marginalization operator. The operators are defined by a list of possible specific operators. However they they can be defined axiomatically, as we will elaborate later.

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 from last to first. When node $X$ is processed, all its earlier neighbors are connected.*

### 2.1 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 maximization over all full tuples, $\Downarrow_\emptyset = \max_X$.

**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 projection: $\Downarrow_\emptyset = \Pi_X$.
- **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

Tree clustering schemes are popular both for constraint processing and probabilistic reasoning. The most popular variants are join-tree clustering algorithms, also called junction-trees. The schemes vary somewhat in their graph definition as well as in the way tree-decompositions are processed [Maier, 1983, Dechter and Pearl, 1989, Lauritzen and Spiegelhalter, 1988, Jensen *et al.*, 1990, Georg Gottlob and Scarello, 1999, Shenoy, 1996, Schmidt and Shenoy, 1998]. They all involve a decomposition of a hypergraph into a hypertree.

To allow a coherent discussion and extension of these methods we find it necessary to introduce a unifying perspective. We present a unifying tree-decomposition framework that borrows its notation from the recent hypertree decomposition proposal for constraint satisfaction presented in [Georg Gottlob and Scarello, 1999]. 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 $\chi$ and $\psi$ are labeling functions which associate with each vertex $v \in V$ two sets, $\chi(v) \subseteq X$ and $\psi(v) \subseteq F$, 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. We will often use the term *cluster* referring to a node with its set of functions, We will use the terms tree-decomposition and a cluster-tree interchangeably.

**Algorithm cluster 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 defined by clusters containing the original functions as well as the received messages. A solution computed from the augmented clusters.
**Compute messages:**
Let $m_{(i,u)}$ denote the message sent by vertex $i$ to vertex $u$.
For every node $u$ in the cluster tree, do

- If $u$ has received messages from all adjacent vertices other than $v$, then compute the message to node $v$:

$$m_{(u,v)} = \Downarrow_{sep(u,v)} \Big( \bigotimes_{f \in \psi(u) \cup \{m_{(i,u)} | (i,u) \in T,\ i \neq v\}} f \Big)$$

**Compute solution:** For every $v \in T$ and every $Z_i \subseteq \chi(v)$, compute $\Downarrow_{Z_i} \bigotimes_{f \in cluster(v)} f$
where $cluster(u) = \psi(u) \cup \{m_{(v,u)} | (v,u) \in T\}$

Figure 1: Algorithm cluster-tree elimination (CTE)

DEFINITION **3.2 (tree-width, separator)** *The width (also called tree-width) 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, a separator of $u$ and $v$ is defined as $sep(u, v) = \chi(u) \cap \chi(v)$.*

**Example 3.1** *Consider a problem $P$ over variables $A, B, C, D, F, G$ with functions over scopes of size 2 or 3: $F = \{f(A, B), f(A, C), f(B, C), f(B, F), f(C, F), f(A, B, D)\ f(F, G)\}$. Figure 3b gives its primal graph. Any of the trees in Figure 7 is a tree-decomposition for the problem. The $\chi$ labelings of each node in the tree are explicitly displayed. The $\psi$ labeling that will satisfies condition 1 of a tree-decomposition can be obtained by any partitioning of the functions into nodes whose $\chi$ label contain their arguments.*

A tree-decomposition facilitates a solution to an automated reasoning task. Many variations for processing tree-decompositions exist. The variant we propose, called Cluster-Tree Elimination (CTE), is typical, aiming into minimizing the space complexity of the algorithm. Algorithm CTE 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 its functions and all messages received by $u$ from all adjacent nodes other than $v$, combine them and marginalize onto the separator of $u$ and $v$. The resulting 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 output tree is used to compute the solution (e.g., computing optimal tuple).

The correctness of CTE was proved for the respective tasks in constraint satisfaction and probabilistic reasoning. The extension to the unified framework is immediate. A more general, axiomatic treatment of this subject, applicable for CTE as well can be found in the work of Shenoy [Shafer and Shenoy, 1990, Shenoy, 1992, Schmidt and Shenoy, 1998]. For clarity and completeness we can show explicitly

THEOREM **3.2 (Correctness and completeness)** *Assume that the combination operator $\bigotimes_i$ and marginalization operator $\Downarrow_Y$ satisfy the following properties:*

1. *Associativity:* $f \bigotimes g = g \bigotimes f$
2. *Commutativity:* $f \bigotimes (g \bigotimes h) = (f \bigotimes g) \bigotimes h$
3. *Restricted distributivity:* $\Downarrow_{X-\{z\}} [f(X-\{z\}) \bigotimes g(X)] = f(X-\{z\}) \bigotimes \Downarrow_{X-\{z\}} g(X)$

*Algorithm $CTE$ is sound and complete for any automated reasoning problem whose operators satisfy associativity, commutativity and restricted distributivity.* □

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 $O(N \cdot exp(sep))$.*

**Proof.** The complexity of processing a node $u$ is $deg_u \cdot (|\psi(u)| + deg_u - 1) \cdot exp(|\chi(u)|)$, where $deg_u$ is the degree of $u$. By bounding $deg_u$ by $deg$ and $\chi(u)$ by $w$, and summing over all nodes, we can bound the entire time complexity by $O(deg \cdot (r+N) \cdot exp(w))$.

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 $exp(sep)$, the space complexity is bounded by $O(N \cdot exp(sep))$.

□

## 4 Bucket-Tree Elimination

In this section we extend the bucket-elimination (BE) 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 can be seen as an instance of $CTE$.

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 highest 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 completeness sake we present the BE algorithm for a general reasoning tasks in Figure 2 [Dechter, 1999]. It is well known that the complexity of $BE$ is exponential in the induced-width or the problems graph along the processed ordering.

DEFINITION **4.1 (buckets)** *Let $P =< X, D, F, \bigotimes, \Downarrow Z_1, ..., Z_t >$ 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 $x_i$.*

We will next create a tree structure between the buckets and associate each bucket with a set of variables. A *bucket-tree* of $P$ along 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 function in the bucket once BE terminated. Formally, the tree and the bucket's variables can be characterized using the induced-width.

**Algorithm bucket-elimination (BE)**
**Input:** An automated reasoning task $P =< X, D, F, \bigotimes, \Downarrow, \{x_1, ..., x_n\} >$, an ordering of the variables, $d = X_1, ..., X_n$.
**Output:** A new compiled set of functions from which $\Downarrow_Y \otimes_{i=1}^n f_i$ can be derived in linear time.
1. **Initialize:** Generate an ordered partition of the functions into $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the functions whose highest variable in their scope is $X_i$. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which functions (new or old) are defined.
2. **Backward:** For $p \leftarrow n$ downto 1, do
for all the functions $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$, do

- $U_p \leftarrow \bigcup_{i=1}^j S_i - \{X_p\}$. Generate $\lambda_p = \Downarrow_{U_p} \otimes_{i=1}^j \lambda_i$ and add $\lambda_p$ to the largest-index variable in $U_p$.

3. **Return:** all the functions in each bucket, and for $x_1$, $\Downarrow_{x_1} \otimes_{\lambda \in bucket_i} \lambda$

Figure 2: Algorithm *bucket-elimination*

Definition 4.2 **(bucket tree)** *Let $G_d^*$ be the induced graph along $d$ of a reasoning problem $P$ whose primal graph is $G$. The variables of $B_x$ are $x$ and its earlier neighbors in the induced-graph $G_d^*$. 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 3a. Figure 5 left shows the initial buckets along the ordering $d = A, B, C, D, F, G$, and the $\lambda$ messages that will be passed by $BE$ from top to bottom. On its right, the figure displays the bucket-tree along the ordering.*

Theorem 4.2 *A bucket tree of a problem $P$ is a tree-decomposition of $P$.*

**Proof.** We need to provide mappings $\chi$ and $\psi$ and show that tree-decomposition properties hold for a bucket tree :

1. $\chi(B_x)$ contains $x$ and its earlier neighbors in the induced graph $G_d$ along ordering $d$.
2. $\psi(B_x)$ contains all functions whose highest-ordered argument is $x$.

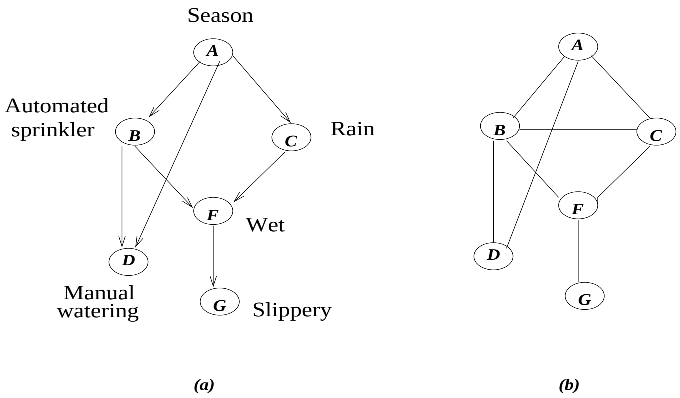


Figure 3: belief network $P(g, f, d, c, b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a)P(a)$

By construction, the tree-decomposition properties other than the connectedness property hold. In order to prove connectedness, let's assume the following: There are two buckets $B_x$ and $B_y$, both containing variable $z$. Also, on the path between $B_x$ and $B_y$ in the bucket-tree, there is a $B_u$ that does not contain $z$. Let $B_i$ ($B_j$) be the first bucket on the path from $B_x$ ($B_y$) to $B_u$ containing $z$, but its parent does not contain $z$. Because $B_u$ is on the path between $B_i$ and $B_j$, it must be that $i \neq j$. Since the parents of $B_i$ and $B_j$ do not contain $z$, it must be that variable $z$ was eliminated at nodes $B_i$ and $B_j$ during the top-down phase of bucket-tree elimination. However, this is impossible, because during the top-down phase, each variable gets eliminated exactly once. Therefore, $B_u$ cannot exist. □

Since the bucket-tree is a tree-decomposition, the cluster-tree elimination algorithm $CTE$ is applicable. 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 BTE algorithm, we will use two types of messages, $\lambda$'s and $\pi$'s as common in the exposition of probabilistic inference.

Algorithm *bucket-tree elimination* (BTE) is given in Figure 4. In the top-down phase, each bucket receives $\lambda$ messages from its children and sends a $\lambda$ message to its parent. This portion is equivalent to $BE$. In the bottom-up phase, each bucket receives a $\pi$ message from its parent and sends $\pi$ messages to each child.

**Example 4.3** *Figure 6 shows the complete execution of $BTE$ along the linear order of buckets and along the bucket-tree. The $\pi$ and $\lambda$ messages are viewed as messages placed on the outgoing arcs.*

Theorem 4.4 *Algorithm BTE is sound and complete*

**Algorithm bucket-tree elimination (BTE)**
**Input:** A problem $P =< X, D, F, \bigotimes, \Downarrow, \{x_1, ..., x_n\} >$, ordering $d$. $G_d$ the induced graph along $d$.
**Output:** Augmented buckets defined by the original functions and all the $\pi$ and $\lambda$ functions received from neighbors in the bucket-tree. A solution to $P$ computed from augmented buckets.
**0. Pre-processing:** Place each function in its latest bucket along $d$, that mentions a variable in its scope. Connect two buckets $B_x$ and $B_y$ if variable $y$ is the latest earlier neighbor of $x$ in the induced graph $G^*{}_d$.
**1. Top-down phase: $\lambda$ 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 $\lambda^y_{x_i}$ sent from $x_i$ to its parent $y$, is computed by

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

where $sep(x_i, y)$ is the separator of $x_i$ and $y$.
**2. Bottom-up phase: sending $\pi$ 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 $\pi$ message received from its parent $y$, $\pi^{x_i}_y$, and computes a message $\pi^{z_j}_{x_i}$ for each child bucket $z_j$ by

$$\pi^{z_j}_{x_i}(sep(x_i, z_j)) = \Downarrow_{sep(x_i,z_j)} \pi^{x_i}_y \bigotimes (\bigotimes_{\lambda_i \neq \lambda^{x_i}_{z_j}} \lambda_i)$$

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

Figure 4: Algorithm Bucket-Tree Elimination

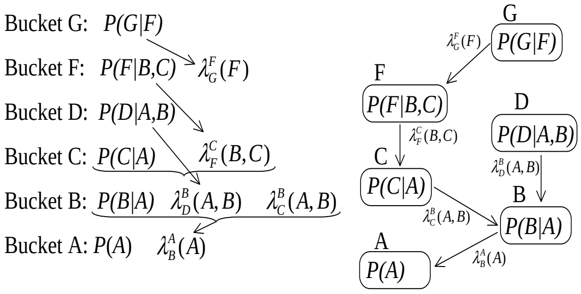


Figure 5: Execution of $BE$ along the bucket-tree

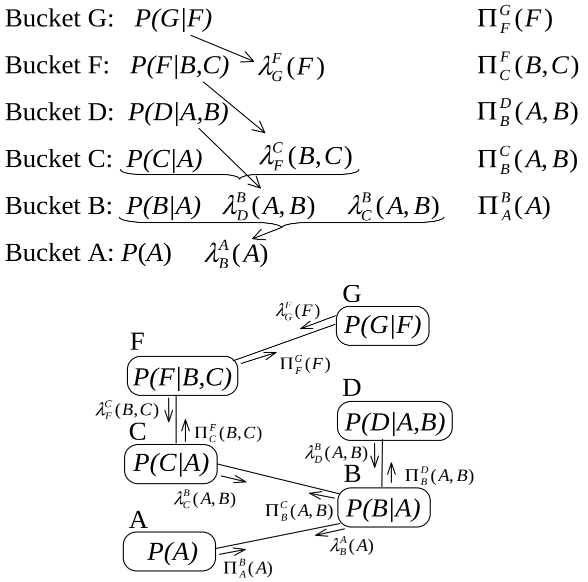


Figure 6: Propagation of $\pi$s and $\lambda$s along the bucket-tree

**Proof:** Since a bucket-tree is a tree-decomposition and since it can be shown that $CTE$ applied to a bucket-tree is equivalent to $BTE$, the soundness and correctness of $BTE$ follows from the soundness and correctness of $CTE$. $\Box$

### 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 $O(n \cdot exp(sep))$.*

**Proof.** During BE, each bucket sends a $\lambda$ message to its parent and since it computes a function defined on all the variables in the bucket, the number of which is bounded by $w^*$, the computed function has domain which is exponential in $w^*$. Since the number of functions that need to be consulted for each tuple in the generated function is bounded by the number of original functions in the bucket, $r_{x_i}$ plus the messages received from its children, which is bounden by $deg_i$, the overall computation, summing over all buckets, is bounded by

$$\sum_{x_i}(r_{x_i} + deg_i - 1) \cdot exp(w^* + 1)$$

The total complexity can be bounded by $O((r + n) \cdot exp(w^* + 1))$. Assuming $r \geq n$, this becomes $O(r \cdot exp(w^* + 1))$. The size of each $\lambda$ message is $O(exp(w^*))$. Since the total number of $\lambda$ messages is $n - 1$, the total space complexity is $O(n \cdot exp(w^*))$. $\square$

THEOREM **4.6 (Complexity of BTE)** *Let $w^*$ be the induced width of $G$ along ordering $d$ and $sep$ its maximum separator ($sep \leq w^*$). 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 $O(n \cdot exp(sep))$.*

**Proof:** Since the number of buckets is $n$, from the analysis of $CTE$ we can derive the time complexity of $BTE$ to be $O((r + n) \cdot deg \cdot exp(w^*))$. Assuming that $r \geq n$ we get the desired bound for time complexity. Since the size of each message is $exp(sep)$ we get space complexity of $O(n \cdot exp(sep))$. $\square$.

The speedup expected from running $BTE$ vs running BE $n$ times (called, $n$-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 speedup of BTE over $n$-BE would be $\Omega(n/deg)$. In a companion paper [Larrosa *et al.*, 2001] we evaluate empirically the speed-up of an approximation scheme based on $BTE$ that show substantial gains. Clearly, for some problems (e.g., chains) the speedup of $BTE$ over $n$-BE is proportional to $n$.

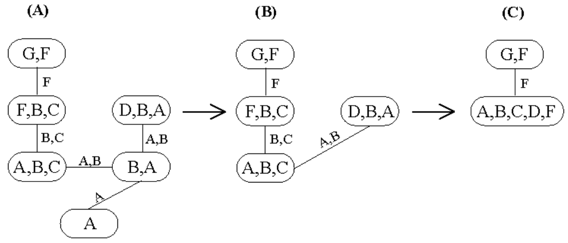


Figure 7: From a bucket-tree (left) to join-tree (middle) to a super-bucket-tree (right)

## 5 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 their width and the hyper-width.

**Proposition 1** *If $T$ is a tree-decomposition, then any tree obtained by merging adjacent nodes in $T$, is a tree-decomposition.* $\square$

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 7.

## 6 Conclusions

By its nature the work here is related to all the work in the past two decades on tree-decompositions for specific tasks, to which we referred sporadically throughout the paper. Unifying framework were presented [Shenoy, 1992, Shenoy, 1996, Bistarelli *et al.*, 1997]. The work here put some of these schemes and formalisms together.

The main novelty of the paper is in 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 was carried out in a general setting explicating the relationship of the proposed algorithm with tree-decomposition schemes. The extension is important for a variety of reasoning tasks. The correctness and complexity of the involving algorithms is analyzed.

## References


[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-99*, 1999.

[Jensen *et al.*, 1990] F.V. Jensen, S.L Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4:269–282, 1990.

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

[Larrosa *et al.*, 2001] J Larrosa, K. Kask, and R. Dechter. Up and down mini-bucket: a scheme for approximating combinatorial optimization tasks. *Submitted*, 2001.

[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.

[Schmidt and Shenoy, 1998] T. Schmidt and P.P. Shenoy. Some improvements to the shenoy-shafer and hugin architecture for computing marginals. *Artificial Intelligence*, 102:323–333, 1998.

[Shafer and Shenoy, 1990] G. R. Shafer and P.P. Shenoy. Axioms for probability and belief-function propagation. volume 4, 1990.

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

[Shenoy, 1996] P.P. Shenoy. Binary join trees. pages 492–499, 1996.