# Unifying Cluster-Tree Decompositions for Reasoning in Graphical models[*]

Kalev Kask[*], Rina Dechter[*], Javier Larrosa[**] and Avi Dechter[***]

[*]Bren School of Information and Computer Science,
University of California, Irvine, CA 92697-3425, USA
[**] Universitat Politcnica de Catalunya (UPC), Barcelona, Spain
[***] Business school, California State University, Northridge, CA, USA

April 7, 2005

**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. Within this framework, we introduce a new algorithm, called bucket-tree elimination ($BTE$), that extends Bucket Elimination ($BE$) to trees, and show that it can provide a speed-up of $n$ over $BE$ for various reasoning tasks. Time-space tradeoffs of tree-decomposition processing are analyzed.

## 1 Introduction

This paper provides a unifying perspective of tree-decomposition algorithms that appear in a variety of automated reasoning areas. Its main contribution

[*]This work was supported in part by NSF grant IIS-0086529 and by MURI ONR award N0014-00-1-0617

is bringing together, within a single coherent framework, seemingly different approaches that have been developed over the years within a number of different communities [1].

The idea of embedding a database that consists of a collection of functions or relations in a tree structure and, subsequently, processing it effectively by a tree-processing algorithm, has been discovered and rediscovered in different contexts. Database researchers observed, almost three decades ago, that relational database schemes that constitute join-trees enable efficient query processing [22]. In fact, many of the important properties of tree-based processing were discussed and proved within the database community [2, 32]. Join tree transformations and their associated variable elimination algorithms were proposed for the efficient solution of constraints satisfaction problems [9] and their role was re-formalized and extended more recently in [14]. In the area of belief networks, junction-tree clustering emerged as the leading strategy for performing probabilistic inference [21]. Variants of this approach were subsequently offered as a means to better address time-space considerations [15, 28, 27]. Tree-decomposition techniques were also introduced in the context of machine learning [12]. Finally, the notion of tree-width, as a means for capturing the decomposition of a hyper-graph into a hyper-tree, is well known in the area of theoretical computer science for quite sometime [25, 1].

The aim of this paper is to present the central concepts and properties of cluster-tree decomposition techniques by way of a single, unified, framework for the purpose of making it more accessible to researchers in diverse areas and facilitating the transfer of this methodology among these areas. Specifically, we demonstrate that join-tree clustering, junction-tree decomposition, and hyper-tree decomposition that, as mentioned above, were developed in different contexts and for different applications, are all instances of our unified scheme. Also, as we show, variable elimination algorithms can be viewed as processing specific cluster-tree decompositions.

Our work is related in aim and content to the earlier work of Shenoy and Shafer [4, 31], who presented a unifying, axiomatic, approach for reasoning tasks. Several of the technical results described in this paper have parallels in these earlier papers. Our work expands on this work in several ways.

[1]Earlier versions of some parts of this paper appear in [6] Chapter 9 (restricted to constraint processing only) and in [19, 24].

First, we use a graph-based language, connecting our approach explicitly to graphical models and using graph-based parameters for capturing algorithmic principles. In particular, we provide graph-based complexity analysis that focuses on the time vs. space issue. Second, we emphasize the distinction between the generation of a structural tree-decomposition and the tree-processing algorithms that are enabled by the decomposition. Third, we show that variable elimination algorithms can be viewed as a specific tree-processing schemes that can be generalized to full processing along special tree-decompositions called bucket trees. Finally, we note how different tree-decompositions of a given problem yield a spectrum of time-space complexity characteristics using superbuckets.

Following a review of background concepts and definitions in Section 2, Section 3 introduces the concept of reasoning problems over graphical models. In Section 4 we introduce the concept of cluster-tree decompositions, present two algorithms for processing tree-decompositions and analyze their time-space complexities. Section 5 introduces and analyzes the bucket-tree elimination algorithm, and Section 5.4 briefly presents the superbuckets, a general method that governs a transition between tree-decompositions and permits a corresponding trade-off between time and space. Section 6 reviews some existing decomposition methods and places them in the context of cluster-tree decomposition and related work. In Section 7 we provide concluding remarks.

## 2 Preliminaries

**Notations:** A reasoning problem is defined in terms of a set of variables that take their values from finite domains and a set of functions defined over these variables. We denote variables or sets of variables by uppercase letters (e.g., $X$, $Y$, $Z$, $S$ ) and values of variables by lower case letters (e.g., $x, y, z, s$). An assignment $(X_1 = x_1, ..., X_n = x_n)$ can be abbreviated as $x = (x_1, ..., x_n)$. For a set of variables $S$, $D_S$ denotes the Cartesian product of the domains of variables in $S$. If $X = \{X_1, ..., X_n\}$ and $S \subseteq X$, $x_S$ denotes the projection of $x = (x_1, ..., x_n)$ over $S$. We denote functions by letters $f$, $g$, $h$, etc., and the scope (set of arguments) of a function $f$ by $scope(f)$.

Definition **2.1 (elimination operators)** *Given a function $h$ defined over a subset of variables $S$, the functions* $(\min_X h)$, $(\max_X h)$, *and* $(\sum_X h)$ *where*

*$X \in S$, are defined over $U = S - \{X\}$ as follows: For every $U = u$, and denoting by $(u, x)$ the extension of tuple $u$ by assignment $X = x$, $(\min_X h)(u) = \min_x h(u, x)$, $(\max_X h)(u) = \max_x h(u, x)$, and $(\sum_X h)(u) = \sum_x h(u, x)$. Given a set of functions $h_1, ..., h_k$ defined over the sets $S_1, ..., S_k$, the product function $\Pi_j h_j$ and the sum function $\sum_j h_j$ are defined over $U = \cup_j S_j$ such that for every $u \in D_U$, $(\Pi_j h_j)(u) = \Pi_j h_j(u_{S_j})$ and $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.*

Definition **2.2 (graph concepts)** *A* **directed graph** *is a pair $G = \{V, E\}$, where $V = \{V_1, ..., V_n\}$ is a set of vertices and $E = \{(V_i, V_j)|V_i, V_j \in V\}$ is the set of edges (arcs). If $(V_i, V_j) \in E$, we say that $V_i$* points to *$V_j$. The degree of a variable is the number of arcs incident to it. For each variable $V_i$, the set of parent vertices $pa(V_i)$ or $pa_i$, is the set of variables pointing to $V_i$ in $G$, while the set of child vertices of $V_i$, denoted $ch(V_i)$ or $ch_i$, comprises the variables that $V_i$ points to. The family of $V_i$, $F_i$, consists of $V_i$ and its parent variables. A directed graph is acyclic if it has no directed cycles.*

Definition **2.3 (hyper-graph)** *A* **hyper-graph** *is a pair $H = (V, S)$ where $S = \{S_1, ..., S_t\}$ is a set of subsets of $V$, called hyper-edges.*

Definition **2.4 (primal graph, dual graph)** *The* **primal graph** *of a hyper-graph $H = (V, S)$ is an undirected graph $G = (V, E)$ such that there is an edge $(u, v) \in E$ for any two vertices $u, v \in V$ that appear in the same hyper-edge (namely, there exists $S_i$, s.t., $u, v \in S_i$). The* **dual graph** *of a hyper-graph $H = (V, S)$ is an undirected graph $G = (S, E)$ that has a vertex for each hyper-edge, and there is an edge $(S_i, S_j) \in E$ when the corresponding hyper-edges share a vertex ($S_i \cap S_j \neq \emptyset$).*

Definition **2.5 (hyper-tree)** *A hyper-graph is a* **hyper-tree***, also called* **acyclic hyper-graph** *if and only if its dual graph has an edge subgraph (one that has the same set of vertices as the dual graph, but a subset of the edges) that is a tree and that satisfies the connectedness property, namely all the nodes in the dual graph that contain a common variable, form a connected subgraph (see also definition in Section 4).*

Definition **2.6 (induced-width)** *An* **ordered graph** *is a pair $(G, d)$ denoted $G_d$ where $G$ is an undirected graph, and $d = (X_1, ..., X_n)$ is an ordering of the vertices. The* width of a vertex *in an ordered graph is the number of its earlier neighbors. The* width of an ordered graph*, $w(G_d)$, is the maximum* *width of all its vertices. The* **induced width of an ordered graph**, $w^*(G_d)$, *is the width of the induced ordered graph, denoted* $G_d^*$, *obtained by processing the vertices recursively, from last to first; when vertex* $X$ *is processed, all its earlier neighbors are connected. The* induced width of a graph, $w^*(G)$, *is the minimal induced width over all its orderings [9].*

## 3 Reasoning Tasks over Graphical Models

A graphical model is defined by a set of real-valued functions $F$ over a set of variables $X$, conveying probabilistic and deterministic information, whose structure can be captured by a graph.

Definition **3.1** *A* **graphical model** $\mathcal{R}$ *is a 4-tuple* $\mathcal{R} =< X, D, F, \bigotimes,>$ *where:*

1. $X = \{X_1, \ldots, X_n\}$ *is a set of variables.*
2. $D = \{D_1, \ldots, D_n\}$ *is a set of finite domains.*
3. $F = \{f_1, \ldots, f_r\}$ *is a set of real-valued functions over subsets of* $X$.
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 function* $f_i$, *denoted* $scope(f_i) \subseteq X$, *is the set of arguments of* $f_i$.

*The graphical model represents the combination of all its functions or relations, namely the set* $\bigotimes_{i=1}^{r} f_i$.

Definition **3.2 (cost of a partial assignment)** *The* cost of a partial assignment $\bar{x} = (\langle X_1, x_1\rangle, \ldots, \langle X_i, x_i\rangle)$ *in a graphical model* $\mathcal{R} =< X, D, F, \bigotimes,>$ *is the combination of all the functions whose scopes are included in the partial assignment, evaluated at the assigned values. Namely,* $c(\bar{x}) = \bigotimes_{f|scope(f)\subseteq\{X_1,\ldots X_i\}} f(\bar{x}[scope(f)])$.

There are various queries/tasks that can be posed over graphical models. We refer to them all as *reasoning problems*. We need one more functional operator, *marginalization*, to express most of the common queries.

Definition **3.3 (A reasoning problem)** *A* **reasoning problem** *over a graphical model is defined by a marginalization operator and a set of variable subsets. It is therefore a triplet $\mathcal{P} = (\mathcal{R}, \Downarrow_Y, \{Z_1, \ldots, Z_t\})$ where $\mathcal{R} = \langle X, D, F, \bigotimes \rangle$ is a graphical model and $Z = \{Z_1, \ldots, Z_t\}$ is a set of subsets of variables of $X$. $\Downarrow_Y f \in \{\overset{max}{_{S-Y}} f, \overset{min}{_{S-Y}} f, \overset{\Pi}{_{S-Y}} f, \sum_{S-Y} f\}$, is a marginalization operator, where $S$ is the scope of function $f$ and $Y \subseteq X$. The reasoning problem is to compute:*

$$\Downarrow_{Z_1} \bigotimes_{i=1}^{r} f_i, \ldots, \Downarrow_{Z_t} \bigotimes_{i=1}^{r} f_i$$

*For optimization tasks we have $Z = \{Z_1\}$, $Z_1 = \emptyset$ and $S = X$. Often we also seek an assignment to all the variables that optimizes (maximizes or minimizes) the combined cost function $f$. Namely, we need to find $x = (x_1, ..., x_n)$ such that $f(x) = \Downarrow_\emptyset \bigotimes_{i=1}^r f_i$, where $\Downarrow \in \{min, max\}$.*

For convenience sake we will sometime combine the reasoning problem with its graphical model. In that case a reasoning problem $P$ denotes a six-tuple $P =< X, D, F, \bigotimes, \Downarrow_Y, \{Z_1, ..., Z_t\} >$.

We assume that functions are expressed in tabular form, having an entry for every combination of values from the domains of their variables. Therefore, the specification of such functions is exponential in their scopes (the base of the exponent is the maximum domain size). Relations, or clauses, can be expressed as functions that associate a value of "0" or "1" with each tuple, depending on whether 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 explicitly as a list of possible specific operators. However, they can be defined axiomatically, as we discuss later.

Definition **3.4** *The* **hyper-graph of a graphical model** *has the variables as its vertices and the scopes of functions as its hyper-edges. The* **primal graph of a graphical model** *is the primal graph of the problem's hyper-graph. Namely, the variables are the vertices and any two variables that belong to a function's scope are connected by an edge.*

We next elaborate on the special cases of reasoning tasks defined over constraint networks and belief networks.

## 3.1 Constraint Networks

**Constraint Satisfaction** [6] is a framework for formulating real-world problems, such as scheduling, planning, etc., as a set of constraints between variables. For example, one approach for formulating a scheduling problem as a constraint problem is to create a variable for each resource and time slice. Values of variables would be tasks that need to be scheduled. Assigning a task to a particular variable (corresponding to a resource at some time slice) means that this resource starts executing the given task at the specified time. Various constraints (such as that a given task takes a certain amount of time to execute, or that a task can be executed at most once) can be modeled as constraints between variables.

The Constraint Satisfaction problem is to find an assignment of values to variables that does not violate any constraint, or else to conclude that the problem is inconsistent. Such problems are graphically represented by vertices corresponding to variables and edges corresponding to constraints between variables.

Definition **3.5 (Constraint Networks, Constraint Satisfaction Problems)** *A* **constraint network (CN)** *is defined by a triplet $(X, D, C)$ where $X$ is a set of variables $X = \{X_1, ..., X_n\}$ associated with a set of discrete-valued domains, $D = \{D_1, ..., D_n\}$, and a set of constraints $C = \{C_1, ..., C_r\}$. Each constraint $C_i$ is a pair $(S_i, R_i)$, where $R_i$ is a relation $R_i \subseteq D_{S_i}$ defined on a subset of variables $S_i \subseteq X$ called the scope of $C_i$. The relation denotes all tuples of $D_{S_i}$ allowed by the constraint. The primal graph of a constraint network is called a* **constraint graph**. *A* **solution** *is an assignment of values to variables $x = (x_1, ..., x_n)$, $x_i \in D_i$, such that each constraint is satisfied, namely $\forall\ C_i \in C\ x_{S_i} \in R_i$. The Constraint Satisfaction Problem (CSP) is to determine whether a constraint network has a solution, and if it does, to find a solution. A binary CSP is one where each constraint involves at most two variables, namely $|S_i| \leq 2$. Sometimes (e.g., the Max-CSP problem defined below), we express the relation $R_i$ as the cost function $C_i(X_{i_1} = x_{i_1}, ..., X_{i_k} = x_{i_k}) = 0$ if $(x_{i_1}, ..., x_{i_k}) \in R_i$, and $1$ otherwise.*

A constraint satisfaction problem is a reasoning task $P = (\mathcal{R}, \Pi, Z = \emptyset)$, where $\mathcal{R} =< X, D, C, \bowtie>$ is a constraint network, the combination operator is the join operator and the marginalization operator is the projection operator. Namely, the problem is to find $\Downarrow_\emptyset \otimes_i f_i = \Pi_X \bowtie_i f_i$.

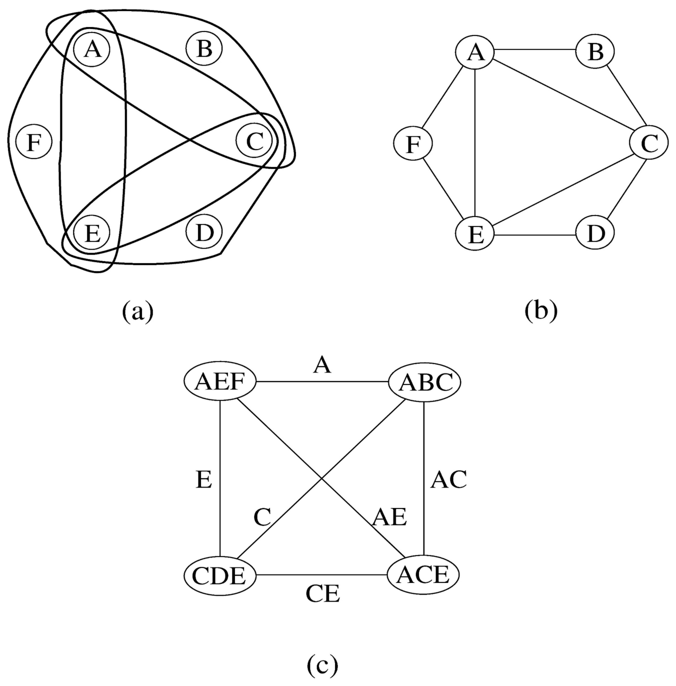

Figure 1: (a) Hyper, (b) Primal, (c) Dual graphs of a CSP.

**Example 3.1** Figure 1 depicts the *hyper-graph* (a), the *primal graph* (b) and the *dual graph* (c) representations of a constraint network with variables $A, B, C$, $D, E, F$ and with constraints on the scopes $(ABC)$, $(AEF)$, $(CDE)$ and $(ACE)$. The specific constraints are irrelevant to the current discussion; they can be arbitrary relations over domains of $\{0, 1\}$, such as $C = A \vee B$, $F = A \vee E$, and so on.

Real-world problems are often over-constrained and do not have a solution. In such cases, it is desirable to find an assignment that satisfies a maximum number of constraints, called a Max-CSP assignment. A Max-CSP problem as the name suggests is a maximization problem, it can also be defined as a minimization problem. Instead of maximizing the number of constraints that are satisfied, we minimize the number of constraints that are violated.

Definition **3.6 (Max-CSP)** *Given a constraint network,* $\mathcal{R} =< X, D, C, \bowtie>$ *the Max-CSP task means finding an assignment* $x^0 = (x^0_1, .., x^0_n)$ *that minimize the number of violated constraints, namely* $\sum_{i=1}^r C_i(x^0_{scope(C_i)}) = min_x \sum_{i=1}^r C_i(x_{scope(C_i)})$*.*

A Max-CSP is a reasoning task $P =< \mathcal{R}, min, Z = \emptyset >$, where $\mathcal{R} =< X, D, F, \sum >$ where $F$ is the set of cost functions assigning 0 to all allowed tuples and 1 to all non-allowed tuples. The combination operator is summation and the marginalization operator is the minimization operator. Namely, the task is to find $\Downarrow_\emptyset \bigotimes_i f_i = \min_X \sum_i f_i$. It also requires an optimizing assignment.

## 3.2 Belief Networks

*Belief Networks* [23] provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over vertices representing variables of interest (e.g., the temperature of a device, the gender of a patient, a feature of an object, the occurrence of an event). The arcs signify the existence of direct causal influences between linked variables quantified by conditional probabilities that are attached to each cluster of parents-child vertices in the network.

Definition **3.7 (Belief Networks)** *Given a set* $X = \{X_1, \ldots, X_n\}$ *of variables over multi-valued domains* $D = \{D_1, ..., D_n\}$*, a* **belief network** *is a pair* $(G, P)$*, where* $G$ *is a directed acyclic graph over* $X$ *and* $P = \{P_i\}$ *are conditional probability matrices* $P_i = \{P(X_i \,|\, pa\,(X_i)\,)\,\}$ *associated with each* $X_i$ *and its parents. Given a subset of variables* $S$*, we will denote by* $P(s)$ *the probability* $P(S = s)$*, where* $s \in D_S$*. A belief network represents a probability distribution over* $X$*,* $P(x_1, ...., x_n) = \Pi_{i=1}^n P(x_i|x_{pa_i})$*. An evidence set* $e$ *is an instantiated subset of variables. The primal graph of a belief network is called a moral graph. It can be obtained by connecting the parents of each vertex in* $G$ *and making every directed arc, undirected. Equivalently, it connects any two variables appearing in the same family.*

Definition **3.8 (Belief Updating)** *Given a belief network and evidence* $e$*, the* **belief updating** *task is to compute the posterior marginal probability of assignment* $X_i = x_i$*, conditioned on the evidence, namely,*

$$Bel(X_i = x_i) = P(X_i = x_i|e) = \alpha \sum_{\{x=(x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_n)\}} \prod_{k=1}^n P(x_k, e|x_{pa_k, X_i=x_i})$$

*where $\alpha$ is a normalization constant.*

When formulated as a reasoning task, functions in $F$ denote conditional probability tables and the scopes of these functions are determined by the directed acyclic graph (DAG): Each function $f_i$ ranges over variable $X_i$ and its parents in the DAG. The combination operator is $\bigotimes_j = \prod_j$, the marginalization operator is $\Downarrow_{X_i} = \sum_{X-X_i}$, and $Z_i = \{X_i\}$. Namely, $\forall Z_i, \Downarrow_{Z_i} \bigotimes_i f_i = \sum_{X-\{X_i\}} \prod_i f_i$.

Definition **3.9 (Most Probable Explanation)** *Given a belief network and evidence $e$, the* **most probable explanation** *(MPE) task is to find a complete assignment which agrees with the available evidence, and which has the highest probability among all such assignments, namely, to find an assignment $(x_1^o, \ldots, x_n^o)$ such that*

$$P(x_1^o, \ldots, x_n^o) = P(x_1, ..., x_n, e) = max_{(x_1,...,x_n)} \prod_{k=1}^{n} P(x_k, e | x_{pa_k})$$

When MPE is formalized as a reasoning task, the combination operator is multiplication and the marginalization operator is maximization. An MPE task is to find $\Downarrow_\emptyset \bigotimes_i f_i = \max_X \prod_i f_i$, where $X$ is the set of variables and $f_i$ is the set of conditional probability tables. It also requires an optimizing assignment.

**Example 3.2** Consider a belief network in Figure 2a. It contains variables $A, B, C, D, F, G$ and functions $f(A, B)$, $f(A, C)$, $f(B, C, F)$, $f(A, B, D)$, $f(F, G)$, modelling the dependency of the lawn being wet on various other phenomena, such as rain, sprinkler system, etc. All variables, except *Season* have two values. The domain of variable *Season* is {Winter, Spring, Summer, Fall} and the prior probability associated with *Season* is $P(Season) = \{0.25, 0.25, 0.25, 0.25\}$. All other variables are associated with a conditional probability. For example, $P(Rain|Winter) = 0.01$, $P(Rain|Spring) = 0.10$, $P(Rain|Summer) = 0.25$, $P(Rain|Fall) = 0.35$; $P(Sprinkler|Winter) = P(Sprinkler|Spring) = 0.3$, $P(Sprinkler|Summer) = P(Sprinkler|Fall) = 0.9$; $P(Wet|Rain, Sprinkler) = 0.95$, $P(Wet|Rain, \neg Sprinkler) = 0.5$, $P(Wet|\neg Rain, Sprinkler) = 0.75$, $P(Wet|\neg Rain, \neg Sprinkler) = 0.05$. It was observed that the lawn is wet and we want to know what is the probability that it was raining and the probability that the sprinkler was on. We can compute $P(Rain|Wet) = 0.38$

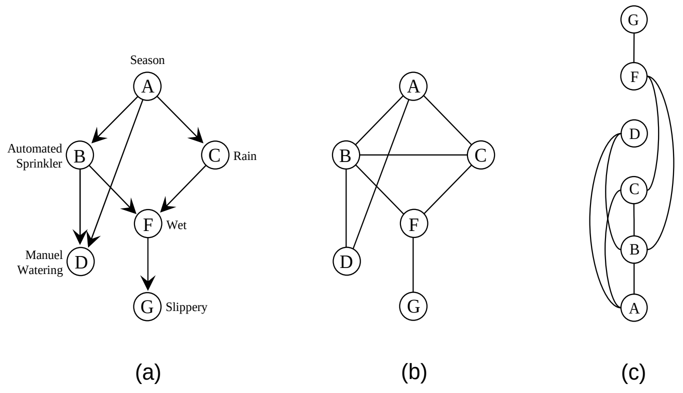

Figure 2: (a) Belief network $P(g, f, d, c, b, a)$, (b) its moral graph and (c) its induced graph.

and $P(Sprinkler|Wet) = 0.59$. Figure 2c gives the induced-graph in (b) along the ordering $d = A, B, C, D, F, G$.

## 4 Cluster-Tree Decomposition

Tree clustering schemes have been widely used for constraint processing, probabilistic reasoning and for graphical models in general. The most popular variants are join-tree clustering algorithms, also called junction-trees. The schemes vary somewhat in their graph definitions as well as in the way tree-decompositions are processed [22, 10, 21, 15, 14, 29, 30]. However, they all involve a decomposition of a hyper-graph into a hyper-tree.

To allow a coherent discussion and extension of these methods, we present a unifying (cluster-)tree-decomposition framework that borrows its notation from the recent hyper-tree decomposition proposal for constraint satisfaction presented in [14]. The exposition is declarative, separating the desired target output from its generative process.

Definition **4.1** *Let* $P = < \mathcal{R} \Downarrow, \{Z_i\} >$ *be a reasoning problem over a graphical model* $< X, D, F, \bigotimes >$. *A* **tree-decomposition** *for* $P$ *is a triple* $< T, \chi, \psi >$, *where* $T = (V, E)$ *is a tree and* $\chi$ *and* $\psi$ *are labelling functions that 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 $f_i \in \psi(v)$.*

2. *If $f_i \in \psi(v)$, then scope$(f_i) \subseteq \chi(v)$.*

3. *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 or the connectedness property.*

4. *$\forall\ i\ Z_i \subseteq \chi(v)$ for some $v \in T$.*

The "exactly one" requirement in the first condition of Definition 4.1 is needed to guarantee the correctness of the Cluster-Tree Elimination algorithm we present next for problems like belief updating, or optimization problems, since each occurrence of each function might change the value of the combined function. For CSPs (when the combination operator is join), this requirement may be relaxed to "at least one", because adding a constraint to more than one vertex is safe since this does not eliminate any solutions. In general, if the combination operator $\bigotimes$ is idempotent ($f \bigotimes (f \bigotimes g) = f \bigotimes g$, for all $f$ and $g$), "exactly one" can be relaxed to "at least one".

Definition **4.2 (tree-width, hyper-width, separator)** *The width (also called tree-width) of a tree-decomposition $< T, \chi, \psi >$ is $\overset{max}{_{v \in V}} |\chi(v)|$, and its hyper-width is $\overset{max}{_{v \in V}} |\psi(v)|$. Given two adjacent vertices $u$ and $v$ of a tree-decomposition, a separator of $u$ and $v$ is defined as sep$(u, v) = \chi(u) \cap \chi(v)$.*

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

**Example 4.1** Consider the belief network in Figure 2a. Any of the trees in Figure 3 is a tree-decomposition for this problem where the functions can be partitioned into clusters that contain their scopes. The labeling $\chi$ are the sets of variables in each node. For example, Figure 3C shows a cluster-tree decomposition with two vertices, and labelling $\chi(1) = \{G, F\}$

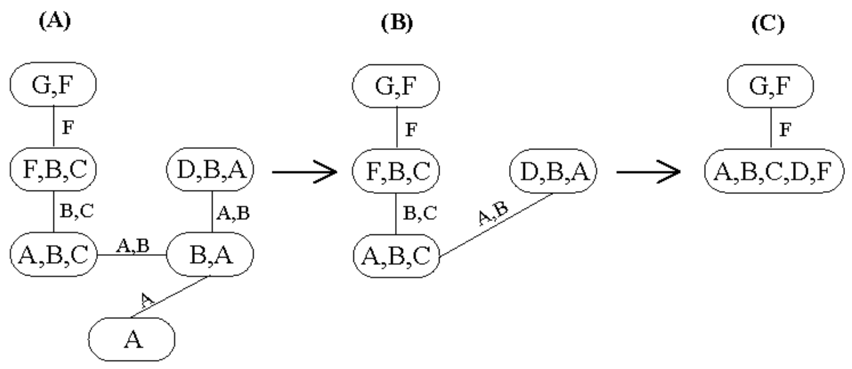

Figure 3: Several tree-decompositions of the same belief network

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 3 are drawn upside-down, namely, the leaves are at the top and the root is at the bottom.

A tree-decomposition facilitates a solution to an automated reasoning task. Cluster-tree elimination algorithm for processing a tree-decomposition is presented as a message-passing algorithm Figure 4. Each vertex of the tree sends a function or a relation 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. 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 4.2** Consider a graphical model whose primal graph appears in

**Algorithm cluster-tree elimination (CTE)**
**Input:** A tree decomposition $< T, \chi, \psi >$ for a problem $P =< X, D, F, \bigotimes, \Downarrow, \{Z_1, ...Z_t\} >$, $X = \{X_1, ..., X_n\}$, $F = \{f_1, ..., f_r\}$.
**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 $v$,

$$m_{(u,v)} = \Downarrow_{sep(u,v)} \Big( \bigotimes_{f \in cluster(u), f \neq m_{(v,u)}} f \Big) \qquad (1)$$

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

Figure 4: Algorithm Cluster-Tree Elimination (CTE)

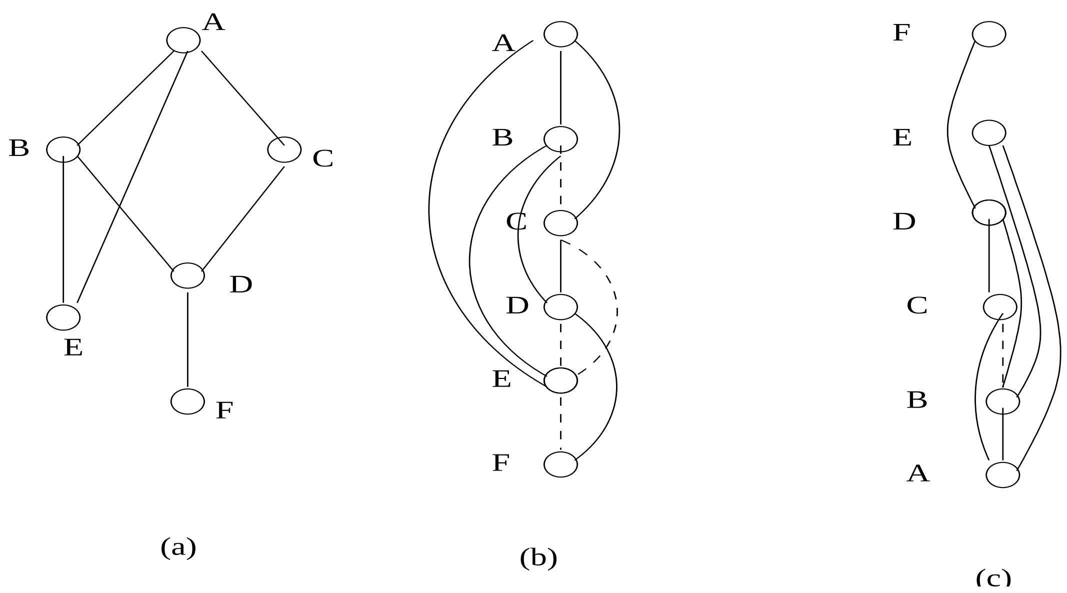

Figure 5: A graph (a) and two of its induced graphs (b) and (c).

Figure 2(a). Assume all functions are on pairs of variables. Two tree-decomspositions are described in Figure 6. The induced-graphs that correspond to the two decompositions are given in Figure 5b,c respectively.

**Example 4.3** Figure 7 shows the messages propagated for the tree-decomspoition in Figure 6b. Assume that it expresses a constraint problem, so the functions are relations. Since cluster 1 contains only one relation, the message from cluster 1 to 2 is the projection of $R_{FD}$ over the separator between cluster 1 and 2, which is variable $D$. The message $m_{(2,3)}$ from cluster 2 to cluster 3 joins the relations 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) in output linear time. For some tasks the whole output tree is used to compute the solution (e.g., computing an optimal tuple).

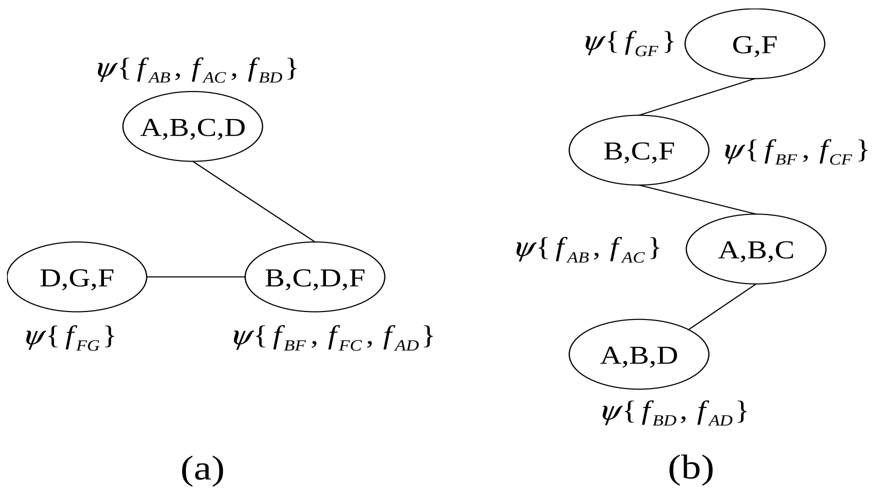

Figure 6: Two tree-decompositions of a graphical model

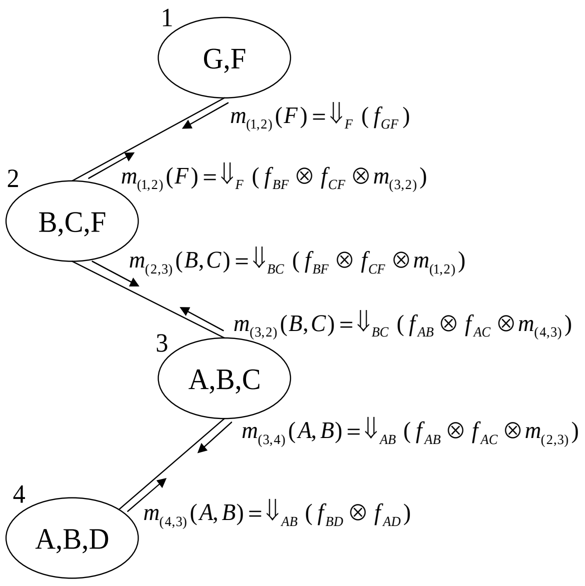
1
G,F
$m_{(1,2)}(F) = \Downarrow_F (f_{GF})$
2
$m_{(1,2)}(F) = \Downarrow_F (f_{BF} \otimes f_{CF} \otimes m_{(3,2)})$
B,C,F
$m_{(2,3)}(B,C) = \Downarrow_{BC} (f_{BF} \otimes f_{CF} \otimes m_{(1,2)})$
$m_{(3,2)}(B,C) = \Downarrow_{BC} (f_{AB} \otimes f_{AC} \otimes m_{(4,3)})$
3
A,B,C
$m_{(3,4)}(A,B) = \Downarrow_{AB} (f_{AB} \otimes f_{AC} \otimes m_{(2,3)})$
4
$m_{(4,3)}(A,B) = \Downarrow_{AB} (f_{BD} \otimes f_{AD})$
A,B,D

Figure 7: Example of messages sent by CTE

.

## 4.1 Correctness of CTE

THEOREM **4.4 (soundness and Completeness)** *Assuming that the combination operator $\bigotimes_i$ and the marginalization operator $\Downarrow_Y$ satisfy the following properties (these properties were first formulated by [31, 30]):*

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.*

A proof of this theorem follows from the work of Shenoy [31, 30]. For completeness we provide a proof which is different and we believe to be clearer.

**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. $\Box$

## 4.2 Complexity of CTE

Algorithm $CTE$ can be subtly varied to influence its time and space complexities. The description in Figure 4 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 1 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 1, 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 combination. This can be implemented by enumeration (or search) as follows: For each assignment $a$ to $\chi(u)$, we can compute its combined functional value, and use this for accumulating the marginalization value on the separator, $sep$, updating $a_{sep}$, of the message function $m(sep)$.

THEOREM **4.5 (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 $O((r+N) \cdot deg \cdot k^w)$ 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, $Time(CTE)$ 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 $w$, we get

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

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

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

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

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 $k^{sep}$, 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 $O(deg \cdot r \cdot k^w)$. $\Box$

## 4.3 Using more space to save time by ICTE

Algorithm CTE presented in Figure 4 is time inefficient in that when a vertex is processed, many computations are performed repeatedly. By precomputing intermediate functions we can reduce the time complexity of the algorithm by a factor of the tree degree. This fact was first observed by Shenoy who proposed a binary tree-architecture [29]. Here we give an alternative formulation of the same idea.

When vertex $u$ is processed, it contains two kinds of functions - original functions (the number of which is $|\psi(u)|$) and messages that $u$ received from its neighbors (there are $deg_u$ of these, one from each neighbor). When a vertex $u$ computes a message to be sent to an adjacent vertex $v$, it combines all original functions $\psi(u)$ with the $deg_u - 1$ messages received from its neighbors other than $v$, and marginalizes over the separator between $u$ and $v$.

Let the neighbors of $u$ be enumerated as $v_1, \ldots, v_{deg_u}$. We can define a set of intermediate functions:

1. Let $f^u = \bigotimes \psi(u)$.
2. Let $m^{(i,j)} = \bigotimes_{k=i}^{j} m_{(v_k,u)}$.

A message that $u$ sends to $v_k$ can be defined as

$$m_{(u,v_k)} = \Downarrow_{sep(u,v_k)} (f^u \bigotimes m^{(1,k-1)} \bigotimes m^{(k+1,deg_u)})$$

In Figure 8 we present an improved version of the CTE algorithm (called ICTE) that precomputes intermediate functions for each vertex. The following theorem proves that ICTE is faster than CTE by a factor of $deg$. However, because ICTE needs to store intermediate functions, its space complexity is exponential in the tree-width, and not in the separator size, as the case is with CTE.

THEOREM **4.6 (Complexity of ICTE)** *Let $N$ be the number of vertices in the tree decomposition, $w$ be its tree-width, $r$ be the number of input functions in $F$, and $k$ be the maximum domain size of a variable. The time complexity of ICTE is $O((r + N) \cdot k^w)$ and its space complexity is $O(N \cdot k^w)$.*

**Proof:** For each vertex $u$, ICTE has to first compute intermediate functions $f^u$, $m^{(1,j)}$ and $m^{(j,deg_u)}$, $j = 2, \ldots, deg_u - 1$, and then messages $m_{(u,v_k)}$ for each adjacent vertex $v$. Computing intermediate functions takes time $O((|\psi(u)| + 2deg_u) \cdot k^w)$ (note that $m^{(1,k)}$ can be computed as $m^{(1,k)} = m^{(1,k-1)} \bigotimes m_{(v_k,u)}$). Once intermediate functions are computed, we can compute messages to all neighbors in time $O(3deg_u \cdot k^w)$ ($deg_u$ neighbors and $O(3 \cdot k^w)$ per neighbor).

**Algorithm improved-cluster-tree elimination (ICTE)**
**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 vertices are clusters containing the original functions as well as messages received from neighbors. A solution computed from the augmented clusters.
**1. Compute messages:**
For every edge $(u, v_l)$ in the cluster tree, such that neighbors of $u$ are enumerated $v_1, \ldots, v_l, \ldots, v_{deg_u}$, do

- If vertex $u$ has received messages from all adjacent vertices other than $v_l$, then Compute $f^u = \bigotimes \psi(u)$, if not yet computed.
- For all $j$, $1 < j < deg_u$, compute $m^{(1,j)} = \bigotimes_{k=1}^{j} m_{(v_k,u)}$ and $m^{(j,deg_u)} = \bigotimes_{k=j}^{deg_u} m_{(v_k,u)}$, if not yet computed.
- Compute $m_{(u,v)}$, the message that vertex $u$ sends to vertex $v$,

$$m_{(u,v_l)} = \Downarrow_{sep(u,v_l)} (f^u \bigotimes m^{(1,l-1)} \bigotimes m^{(l+1,deg_u)})$$

**2. Return:** The cluster-tree augmented with messages and for every $v \in T$ and every $Z_i \subseteq \chi(v)$, compute $\Downarrow_{Z_i} \bigotimes_{f \in cluster(v)} f$.

Figure 8: Algorithm Improved-Cluster-Tree Elimination (ICTE)

Therefore the time complexity of processing vertex $u$ is $O((|\psi(u)| + 5deg_u) \cdot k^w)$. The time complexity of ICTE is

$$\sum_u O((|\psi(u)| + 5deg_u) \cdot k^w)$$

Since $\sum_u |\psi(u)| = r$ and $\sum_u deg_u = 2(N-1)$ time complexity of ICTE is $= O((r+N) \cdot k^w)$

For each vertex $u$, we need to store $O(2deg_u)$ intermediate functions of size $k^w$. By summing over all vertices, the space complexity of storing all intermediate functions is $O(N \cdot k^w)$. Also, for each edge, ICTE has to store two messages of size $k^{sep}$. Since the total number of edges is $N-1$, the space complexity of storing messages is $O(N \cdot k^{sep})$. However, since $sep \leq w$ the total space complexity of ICTE is $O(N \cdot k^w)$. $\Box$

As we have mentioned, Shenoy [29] introduced binary join trees to organize computations more efficiently. For any cluster-tree, there exists a binary cluster-tree such that CTE has the same time and space complexity on the binary tree as ICTE has on the original tree. So, our ICTE algorithm can be viewed as a reformulation and rederivation of Shenoy's result without the actual construction of the binary tree. Our derivation also pinpoints the associated space-time complexity tradeoff.

## 5 Bucket-Tree Elimination

This section extends the bucket elimination scheme into a message passing algorithm along a bucket-tree, and shows that the extended algorithm is an instance of the cluster-tree elimination scheme.

### 5.1 Bucket Elimination

*Bucket elimination (BE)* is a unifying algorithmic framework for dynamic-programming algorithms applicable to any graphical model such as probabilistic and deterministic networks. The input to a BE algorithm consists of a collection of functions or relations of a reasoning problem. Given a variable ordering, the algorithm partitions the 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,

```
Algorithm BE
Input: A problem description $P =< X, D, F, \bigotimes, \Downarrow, \emptyset >$; $F = \{f_1, ..., f_r\}$, an ordering of the variables $d = (X_1, ..., X_n)$.
Output: Augmented buckets containing the original functions and all the message-functions received.
1. Initialize: Partition the functions in $F$ into buckets denoted $B_{X_1}, \ldots, B_{X_n}$, where initially $B_{X_i}$ contains all input functions whose highest variable is $X_i$. (ignore instantiated variables).
During the algorithm's execution $B_{X_i} = \{h_1, h_2, ..., h_j\}$
2. Backward: For $p \leftarrow n$ down-to 1, process $B_{X_i}$:
    • Generate the function $\lambda^p$
          $\lambda^p = \Downarrow_{U_p} \bigotimes_{h \in B_{X_i}} h$
      where $U_p = \bigcup_{h \in B_{X_i}} scope(h) - \{X_p\}$.
    • Add $\lambda^p$ to the bucket of the largest-index variable in $U_p$.
3. Return: The set of augmented buckets and the function computed in the first bucket.
```

Figure 9: Bucket Elimination Algorithm

from the last variable to the first, by a variable elimination procedure that computes a new function using combination and marginalization operators in each bucket. The new function is placed in the closest lower bucket whose variable appear in the function's scope. When the solution of the problem requires a complete assignment (e.g., finding the most probable explanation in belief 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 information see [5].

For the sake of completeness we present in Figure 9 the BE algorithm [5]. It is well known that the complexity of *BE* is exponential in the induced-width of the problem's graph along the the order of processing. We provide a formal result for the complexity of BE in Section 5.3.

## 5.2 Bucket-Tree Elimination

Definition **5.1 (singleton-optimality tasks)** *An automated reasoning problem $P =< X, D, F, \bigotimes, \Downarrow, \{Z_1, ...Z_t\} >$, where $F = \{f_1, ..., f_r\}$, is a* **singleton-optimality problem** *if $t = n$ and for all $i$, $Z_i = \{X_i\}$. In this case, we write $Opt(X_i) = \Downarrow_{X_i} \bigotimes_{i=1}^{r} f_i$.*

Singleton-optimality tasks, require repeated execution of the BE algorithm, for example, when the belief distribution is required for every variable in a belief network. Another example is computing the optimal cost associated with each value of every variable that is used to guide a search algorithm [8]. In order to compute the singleton-optimality task, BE would have to be run $n$ times, each initiated by a different variable. We next propose a more efficient alternative, extending bucket-elimination into a bucket-tree elimination (BTE) scheme. While the essence of this extension can be found in [30], its derivation within the tree-decomposition framework adds clarity. It extends BE's view as message-passing from leaves to root along a bucket-tree [5] with a root-to-leaves message-passing phase similar to the recent suggestion for probabilistic inference [3].

Let $P =< X, D, F, \bigotimes, \Downarrow, \{Z_i\} >$ be a reasoning problem and $d$ be an ordering $d = (X_1, ..., X_n)$. Let $B_{X_1}, ..., B_{X_n}$ denote a set of buckets, one for each variable. Each bucket $B_{X_i}$ contains those functions in $F$ whose latest variable in $d$ is $X_i$. A *bucket-tree* of a problem $P$ 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 $P$ along $d$ using the following definition.

Definition **5.2 (bucket tree, graph-based)** *Let $G_d^*$ be the induced graph along $d$ of a reasoning problem $P$ whose primal graph is $G$. The vertices of the bucket-tree are the $n$ buckets each associated with a variable. 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*

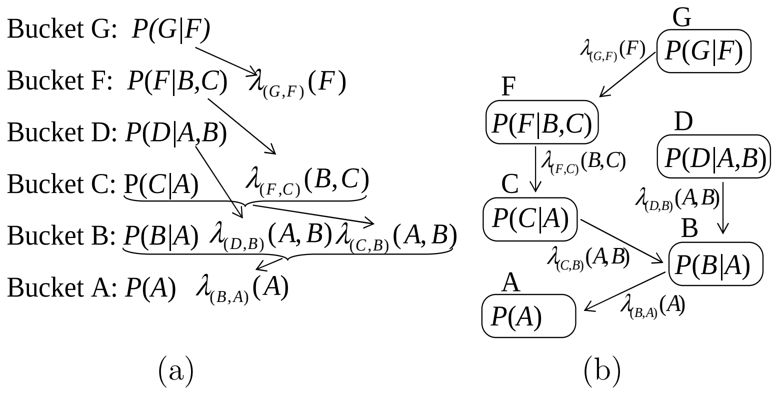

Figure 10: Execution of $BE$ along the bucket-tree

*$B_X$ 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 5.1** Consider the Bayesian network defined over the DAG in Figure 2a. Figure 10a shows the initial buckets along the ordering $d = A, B, C, D, F, G$, and the messages (labelled $\lambda$'s in this case) that will be passed by $BE$ from top to bottom. The message from $B_X$ to $B_Y$ is denoted $\lambda_{(X,Y)}$. Notice that the ordering is displayed bottom-up and messages are passed top-down in the figure. Figure 10b displays the same computation as a message-passing along its bucket-tree.

THEOREM **5.2** *A bucket tree of a reasoning problem $P$ is a tree-decomposition of $P$.*

**Proof.** We need to provide two mappings, $\chi$ and $\psi$, and show that the following two 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 $X$.

By construction, the conditions 1,2 and 4 of tree-decomposition property holds. In order to prove connectedness, let's assume to the contrary that there are two buckets $B_X$ and $B_Y$, both containing variable $Z$, but that on the path between $B_X$ and $B_Y$, there is a bucket $B_U$ that does not contain $Z$. Let $B_i$ be the first bucket on the path from $B_X$ to $B_U$ containing $Z$, but whose parent does not contain $Z$ but whose parent does not contain $Z$. Let $B_j$ be the first bucket on the path from $B_Y$ to $B_U$ containing $Z$. 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 $Z$, variable $Z$ must have been eliminated at nodes $B_i$ and $B_j$ during the top-down phase of bucket-tree elimination. This is impossible, however, because each variable is eliminated exactly once during the top-down phase. Therefore, $B_U$ cannot exist. $\Box$

Since the bucket-tree is a tree-decomposition, algorithm CTE is applicable. The correctness of the extension of BE to that adds a bottom-up message passing can be established by showing equivalence with $CTE$ when applied to the bucket-tree. Algorithm *bucket-tree elimination* (BTE) is given in Figure 11. We describe the algorithm using two types of messages, $\lambda$s and $\pi$s, as is common in some message propagation schemes. 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 a $\pi$ message to each child.

**Example 5.3** Figure 12 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 in the lower portion of the figure.

THEOREM **5.4** *Algorithm BTE is a special case of CTE applied to the bucket tree using a specific order of message computation.*

**Proof:** It is easy to see that messages computed by BTE are exactly the same as those computed by CTE. Since a bucket-tree is a specific case of a cluster-tree decomposition and CTE is correct and complete for cluster-tree decompositions, it follows that BTE is correct and complete. Notice that the actual computation of a message $\lambda_{(X,Y)}$ in the BTE algorithm is identical to the computation of message $m_{(B_X,B_Y)}$ in CTE . We need only to show that the order of message computation by BTE is also a valid order of message computation for the CTE algorithm.

**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 $\pi$ 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 the latest bucket along $d$ that mentions a variable in its scope. Connect two bucket nodes $B_X$ and $B_Y$ if variable $Y$ is the latest among the earlier neighbors 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 $h_1, ..., h_j$ be all the functions in $B_{X_i}$ at the time $B_{X_i}$ is processed. The message $\lambda_{(X_i,Y)}$ sent from $X_i$ to its parent $Y$, is computed over the separator by $sep(X_i, Y)$

$$\lambda_{(X_i,Y)} =\Downarrow_{sep(X_i,Y)} \bigotimes_{h_i \in B_{X_i}} h_i$$

**2. Bottom-up phase: $\pi$ messages**
For $i = 1$ to $n$, process bucket $B_{X_i}$:
Let $h_1, ..., h_j$ 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 $Y$, $\pi_{(Y,X_i)}$, and computes a message $\pi_{(X_i,Z_j)}$ for each child bucket $Z_j$ over the separator $sep(X_i, Z_j)$ by

$$\pi_{(X_i,Z_j)} =\Downarrow_{sep(X_i,Z_j)} \pi_{(Y,X_i)} \bigotimes ( \bigotimes_{h_i \in B_{X_i}, h_i \neq h_{(Z_j,X_i)}} h_i$$

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

Figure 11: Algorithm Bucket-Tree Elimination

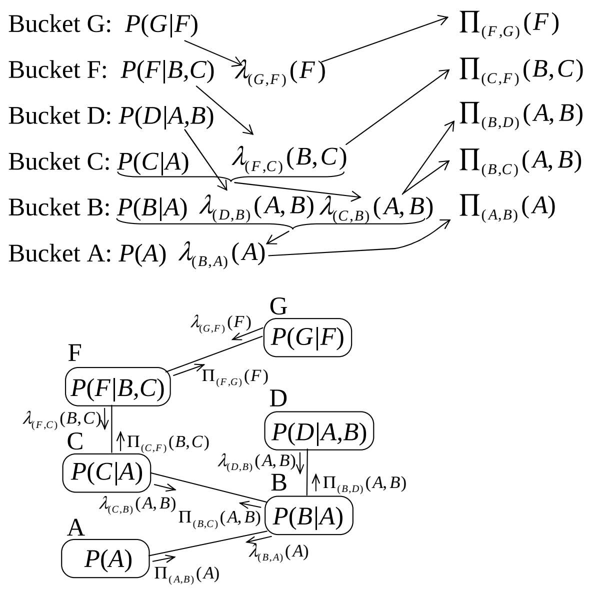

Figure 12: Propagation of $\pi$'s and $\lambda$'s along the bucket-tree

Algorithm CTE specifies that a message $m_{(B_X,B_Y)}$ can be computed when vertex $B_X$ has received messages from all neighbors other than $B_Y$. Therefore, the top-down phase of the BTE, where a bucket $B_X$ sends a message to its parent $B_Y$ after it has received messages from all its children, is a valid order of message computation for CTE. Also, the top-down phase is a valid order of message computation for CTE, since by the time a bucket $B_X$ is processed by BTE, it has received messages from all neighbors. $\Box$

## 5.3 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 $BTE$ and $IBTE$.

THEOREM **5.5 (Complexity of BE)** *Given a reasoning problem $P =< X, D, F, \bigotimes, \Downarrow, \{X_1, ..., X_n\} >$, let $w^*$ be the induced width of its primal graph $G$ along ordering $d$, let $k$ be the maximum domain size of any variable and $r$ the number of* *functions. The time complexity of $BE$ is $O(r \cdot k^{w^*+1})$ and its space complexity is $O(n \cdot k^{w^*})$.*

**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 size of the computed function is exponential in $w^*$. Since the number of functions that need to be consulted for each tuple in the generated function in bucket $B_{X_i}$ is bounded by the number of its original functions, denoted $r_{X_i}$ plus the number of messages received from its children, which is bounded by $deg_i$, the overall computation, summing over all buckets, is bounded by

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

The total complexity can be bound by $O((r+n) \cdot k^{w^*+1})$. Assuming $r > n$, this becomes $O(r \cdot k^{w^*+1})$. The size of each $\lambda$ message is $O(k^{w^*})$. Since the total number of $\lambda$ messages is $n-1$, the total space complexity is $O(n \cdot k^{w^*})$. $\Box$

THEOREM **5.6 (Complexity of BTE)** *Let $P =< X, D, F, \bigotimes, \Downarrow, \{X_1, ..., X_n\} >$ be a reasoning problem, let $w^*$ be the induced width along ordering d, k be the maximum size and let r be the number of functions. The time complexity of $BTE$ is $O(r \cdot deg \cdot k^{w^*+1})$, where deg is the maximum degree of any vertex 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 $w^*$ equals $w-1$, where $w$ is the tree-width, it follows from Theorem 5.4 that the time complexity of $BTE$ is $O((r+n) \cdot deg \cdot k^{w^*+1})$. Assuming that $r > n$ we get the desired bound for time complexity. Since the size of each message is $k^{sep}$, and since here $sep = w^*$, we get space complexity of $O(n \cdot k^{w^*})$. $\Box$

We can apply the idea of precomputing intermediate functions described in Section 8 to BTE, resulting in new algorithm IBTE. However, in this case, we have an improvement in speed with no increase in space complexity.

THEOREM **5.7 (Complexity of IBTE)** *Let $w^*$ be the induced width of $G$ along ordering d of a reasoning problem $P =< X, D, F, \bigotimes, \Downarrow, \{X_1, ..., X_n\} >$ and let k be the maximum size and r the number of functions. The time complexity of $IBTE$ is $O(r \cdot k^{w^*+1})$ and the space complexity is $O(n \cdot k^{w^*+1})$.*

**Proof:** Follows from Theorems 4.6 and 5.4. □.

**Speed-up of BTE vs n-BE.** Next we will compare the complexity of BTE and IBTE against running BE n times ($n$-BE) for solving the singleton optimality task. While both BTE and $n$-BE have the same space complexity, the space needs of IBTE is larger by a factor of $k$, where $k$ is the domain size of any variable.

In theory, the speedup expected from running $BTE$ vs running $n$-BE is at most $n$. This may seem insignificant compared with the exponential complexity in $w^*$. However, in practice it can be significant, especially when these computations are used as a procedure within more extensive search algorithms [17]. 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 $O(deg \cdot r \cdot k^{w^*+1})$. These two bounds cannot be compared directly because we do not know how tight the $n$-BE bound is. However, there are classes of problems (e.g., $w$-trees) for which the complexity of $n$-BE is $\Theta(n \cdot r \cdot k^{w+1})$, and the maximum degree of a vertex in the bucket tree can be bounded by $w$. Therefore, the speedup of BTE over $n$-BE for these classes of problems would be $\Omega(n/deg)$ (also $\Omega(n/w)$). Similar considerations appear when comparing IBTE with $n$-BE. Clearly, the speedup of IBTE over $n$-BE is at least as the speedup of BTE over $n$-BE.

## 5.4 Using more time to save space : Superbuckets

The main drawback of $CTE$ is its memory demands. The space complexity of $CTE$ is exponential in the largest separator size. In Section 4.3 we showed how we can save some time by using more space. Here we will go in the opposite direction. We will show how we can save space if we are willing to allow weaker time bounds for the algorithm. This is because in practice, the space complexity that is exponential in the separator size may be computationally prohibitive. To overcome this limitation time-space tradeoffs were introduced [7]. The idea is to trade off space for time by combining adjacent vertices (that is, combining the variable/funcion labels), thus reducing separator sizes, while increasing their width.

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

**Proof:** It is straightforward to verify that all the conditions of Definition 4.1 still hold. □

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 various tree-decompositions starting at a bucket-tree and merging adjacent vertices. For illustration see Figure 3.

## 6 Comparing Tree-Decomposition Methods

In this section we will discuss the relationships between several known tree-decomposition structuring schemes, their processing schemes and related work.

### 6.1 Join-Tree Clustering

In both the constraint satisfaction and the Bayesian networks communities the common tree-clustering methods, called join-tree (or junction-tree) clustering ([10, 21]), are based on a triangulation algorithm that transforms the primal graph $G = (V, E)$ of a problem instance $P$ into a chordal graph $G'$. A graph is chordal, if any cycle of length 4 or more has a chord. To transform a primal graph $G$ into a chordal graph $G'$, the triangulation algorithm processes $G$ along the reverse order of an ordering $d$ and connects any two non-adjacent vertices if they are connected through a vertex later in the ordering. A join-tree clustering is defined as 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 [22]. 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 2** *Every join-tree clustering is a tree-decomposition.* □

Join-trees correspond to minimal tree-decompositions, where minimality, as we defined earlier, means that separators are always strict subsets of their adjacent clusters, thus excluding some decompositions that can be useful (see [14]). Moreover, they are *cluster-minimal*, no vertex and its variables can be partitioned further to yield a more refined tree-decomposition. This restriction exclude the super-bucket methods that accomodates time-space trade-offs.

**Example 6.1** Consider a graphical model having functions defined on all pairs of variables, whose graph is complete. Clearly, the only possible join-tree will have one vertex containing all the variables and all the functions. An alternative tree-decomposition has vertex $C_1$ 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 vertex, $C_{i,j}$, for each other function that is not contained in $C_1$, and the tree connects $C_1$ with each other vertex. While this is a legitimate tree-decomposition, it is not a legitimate join-tree. This is an example of a hyper-tree decomposition, discussed next.

## 6.2 Hyper-tree Decomposition

Recently, Gottlob et.al [14] 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.

Definition **6.1 (hyper-tree decomposition)** *[14] A (complete) hyper-tree decomposition of a hyper-graph $HG = (X, S)$ is a triple $< T, \chi, \psi >$, where $T = (V, E)$ is a rooted tree, and $\chi$ and $\psi$ are labelling functions which associate with each vertex $v \in V$ two sets $\chi(v) \subseteq X$ and $\psi(v) \subseteq S$, and which satisfies the following conditions:*

1. *For each edge $h \in S$, there exists $v \in V$ such that $h \in \psi(v)$ and $scope(h) \subseteq \chi(v)$ (we say that $v$ strongly covers $h$).*
2. *For each variable $X_i \in X$, the set $\{v \in V | X_i \in \chi(v)\}$ induces a (connected) subtree of $T$.*
3. *For each $v \in V$, $\chi(v) \subseteq scope(\psi(v))$.*
4. *For each $v \in V$, $scope(\psi(v)) \cap \chi(T_v) \subseteq \chi(v)$, where $T_v = (V_v, E_v)$ is the subtree of $T$ rooted at $v$ and $\chi(T_v) = \cup_{u \in V_v} \chi(u)$.*

*The hyper-width hw, of a hyper-tree decomposition is $hw = max_v |\psi(v)|$.*

A hyper-tree decomposition of a graphical model $\mathcal{R}$ is a hyper-tree-decomposition of its hyper-graph that has the variables of $\mathcal{R}$ as its vertices and the scopes of functions as its hyper-edges. In [14] the complexity of processing hyper-tree decomposition for solving a constraint satisfaction problem is analyzed as a function of the hyper-width $hw$. The processing algorithm used for this analysis is similar to tree-clustering *(T-C)* algorithm presented in [10] described as follows. Once a hyper-tree decomposition is available, 1. join all the relations in each cluster, yielding a single relation on each cluster. This step takes $O((m+1) \cdot t^{hw})$ time and space, where $t$ bounds relation size and $m$ is the number of edges in the hyper-tree decomposition, and it creates an acyclic constraint satisfaction problem. 2. Process the acyclic problem by arc-consistency. This step can be accomplished in $O(m \cdot hw \cdot t^{hw} \cdot logt)$ time.

Theorem **6.2** *[13] Let $m$ be the number of edges in the hyper-tree decomposition of a constraint network $\mathcal{R}$, $hw$ its hyper-width and $t$ be a bound on the relation size. A hyper-tree decomposition of a constraint problem can be processed in time*

$$O(m \cdot hw \cdot logt \cdot t^{hw}) \tag{2}$$

*and in space $O(t^{hw})$.*

Not every hyper-tree decomposition is a tree-decomposition. Hyper-tree decomposition allows a function to be placed in more than one vertex. This cannot be allowed for general graphical models and it is therefore made illegal in our definition of tree-clustering 4.1. We therefore define a restricted form of hyper-tree decomposition.

Definition **6.2** *A restricted hyper-tree decomposition is a complete hyper-tree decomposition such that for every hyperedge $h \in S$ there is exactly one $v \in V$ s.t. $h \in \psi(v)$.*

It is easy to see that,

**Proposition 3** *Any restricted hyper-tree decomposition of a reasoning problem $P$ is a tree-decomposition of $P$.* $\square$

Notice that the opposite is not true. There are tree-decompositions that are not (restricted) hyper-tree decompositions, because hyper-tree decompositions require that the variables labelling a vertex will be contained in the scope of its labelling functions (Condition 3 of Definition 6.1). This is not required by the tree-decomposition definition 4.1. For example, consider a single n-ary function $f$. It can be mapped into a bucket-tree with n vertices. Vertex $i$ contains all variables $\{1, 2, ...i\}$ but no functions, while vertex $n$ contains all the variables and the input function. Both join-tree and hyper-tree decomposition will allow just one vertex that include the function and all its variables.

The complexity bound in EQ. (2) can be extended to any graphical model that is absorbing relative to 0 element (a graphical model is absorbing relative to 0 element if its combination operator has the property that $x \bigotimes 0 = 0$; for example, multiplication has this property while summation has not) assuming we use a restricted hyper-tree decomposition, and if the relational nature of constraints is extended to functions in general graphical models. The tabular representation of functions can be converted into relations by removing their zero-cost tuples. For examples, in probabilistic networks all the rows in a CPT that have zero probability can be removed (an idea that was explored computationally in [18]).

In order to apply the hyper-width bound we will consider a specific implementation of the message computation expressed in equation 1 of algorithm CTE. Recall that given a hyper-tree decomposition, each node $u$ has to send a single message to each neighbor $v$. We can compute $m_{(u,v)}$ as follows:

1. Combine all functions $\psi(u)$ in node $u$ yielding function $h(u)$. namely,

$$h(u) = \bigotimes_{f \in \psi(u)} f$$

   This step can be done in time and space $O(t^{|\psi(u)|})$.

2. For each neighbor $c$ of $u$, $c \neq v$ iterate the following:

$$h(u) \leftarrow h(u) \bigotimes \Downarrow_{\chi(u) \cap \chi(c)} m_{(c,u)}$$

   This step can be accomplished in $O(deg \cdot hw \cdot logt \cdot t^{hw})$ time and $O(t^{hw})$ space.

3. Take $m_{(u,v)} \leftarrow h(u)$.

The complexity of the second step can be derived as follows. The marginalization step can be done in linear time in the size of the message sent from $c$ to $u$ whose size is $O(t^{hw})$. The combination of a relation with one that is defined on a subset of its scope can be done in a brute force way quadratically in the size of the respective relations, namely $O(t^{2hw})$. Or, we can sort each relation first in $O(t^{hw} log(t^{hw}))$ time and then combination can be accomplished in linear time in the largest relation, yielding, $O(hw \cdot logt \cdot t^{hw})$. The space required to store the result is bounded by $O(t^{hw})$. Since this computation must be done for every neighbor $c$, we get complexity of $O(deg \cdot hw \cdot logt \cdot t^{hw})$ time and $O(t^{hw})$ space. Finally, the above computation must be accomplished for every neighbor $v$ of $u$ yielding overall complexity of CTE of $O(m \cdot deg \cdot hw \cdot logt \cdot t^{hw})$ time and $O(t^{hw})$ space. The discrepancy between this bound and the one in 6.4 is that the later requires message passing in one direction only. We can conclude:

THEOREM **6.3** *A (restricted) hyper-tree decomposition of a reasoning problem absorbing relative to 0 element can be processed in time*

$$O(m \cdot deg \cdot hw \cdot logt \cdot t^{hw})$$

*and $O(t^{hw})$ space, where $m$ is the number of edges in the hyper-tree decomposition $hw$ its hyper-width and $t$ is a bound on the size of the relational representation of each function in $\mathcal{R}$.*

Theorem 6.3 does not apply for the general definition of tree-decomposition 4.1, even if we use relational representation. The main problem is that the complexity analysis assumed Condition 3 of Definition 6.1 (which can be thought of as "every variable in a vertex of a tree must be covered by a function" in that node). We can remedy this problem if we think of all uncovered variables in a node as having a unit-cost universal relation associated with their scope. For a constraint problem this is the universal relation that allows all combinations of values. For a general graphical model the universal relation will assign each tuple a unit cost of "1" assuming that combining with "1" is not changing the cost. Provided this, we can show

THEOREM **6.4** *[26] A tree-decomposition of a reasoning problem absorbing relative to 0 element can be processed in time*

$$O(m \cdot deg \cdot hw^* \cdot logt \cdot t^{hw^*})$$

*by CTE, where $N$ is the number of vertices in the hyper-tree decomposition, $t$ is a bound on the relation size, and $hw^*(v) = (|\psi(v)| + |\{X_i | X_i \notin scope(\psi(v)\}|)$ and $hw^* = \underset{v \in V}{\overset{max}{}} hw^*(v)$.*

**Proof.** Once we add the universal relation on uncovered variables we have a restricted hyper-tree decomposition to which we can apply the bound in Theorem 6.3 assuming the same implementation of CTE. The number of uncovered variables in a node $v$ is $n(v) = |\{X_i | X_i \notin scope(\psi(v)\}|$. So time processing of a node is $O(t^{hw} \cdot k^{n(v)})$ when $k$ bounds the domain size, yielding $O((max(t,k)^{hw^*})$. Assuming that $t > k$ we can use the bound $O(t^{hw^*})$ time and space. Subsequently, message passing between all nodes yields overall complexity as in 6.3 when $hw$ is replaced by $hw^*$. $\Box =$

Notice that in some cases the tree-width would provide a far better bound on the complexity of CTE than the hyper-width while at other cases it does not. To exploit both parameters when bounding the complexity of CTE we can define $hw^*{}_{\psi'}(v)$ relative to any subset of the functions $\psi' \subseteq \psi(v)$ as follows. $hw^*_{\psi'}(v) = |\psi'(v)| + |\{X_i | X_i \notin scope(\psi'(v)\}|$. CTE's performance will be bounded by any selection of a subset $\psi'(v)$ from $\psi(v)$. In particular, if we choose the empty function set, $n(v)$ becomes equal to $\chi(v)$ yielding $hw^* = w^*$. Clearly finding $\psi'_v$ that minimize $hw^*_{\psi'}(v)$ can be hard. In any case for restricted hyper-tree decomposition the bound is obtained using $\psi'(v) = \psi(v)$, and $hw^* = hw$, while when $\psi' = \emptyset$ $hw^* = w^*$.

## 6.3 Comparing the tree-processing algorithms

Algorithms processing a tree-decomposition are of two primary styles. One class compiles all the functions in a cluster into a single function, and then sends messages between clusters. The other class avoids precompilation into a single function, and works similarly to CTE. Algorithm join-tree clustering [9] for processing constraint networks first creates a single joined function or relation from all the functions in each cluster and then applies message-passing. The same idea is used in *junction-tree* algorithm [21] for probabilistic networks. In the later case, the combined function is called "potential". The time and space complexity of these algorithms are both exponential in the tree-width. On the other hand, the so-called Shafer and Shenoy architecture

[31] for probabilistic networks is similar to CTE, and its time-space algorithm presented in [11]. Like CTE, these variants have a more efficient space complexity which is exponential in the separator's width only.

## 6.4 More on Related Work

As noted, join-tree (junction-tree) in LS ([21], [20]) and Hugin ([16], [15]) architectures is a special case of a tree-decomposition, constructed from the cliques of the triangulated moral graph of the problem's graph. Each vertex contains exactly one potential after combining all its functions, while Hugin also records a potential on the separator. When a message is received by a vertex, it updates its potential by incorporating the new message. To avoid overcounting, LS architecture divides the potential of the vertex by the outgoing inward message, while Hugin divides messages by the separator. Both are designed for computing marginals in the Bayesian network.

Shenoy-Shafer architecture executes what is called the Fusion algorithm, on a binary join-tree ([30]), for computing marginal probabilities in a Bayesian network. As noted earlier, a binary join-tree is constructed by taking a join-tree (or a collections of potentials) and then transforming it into a binary tree.

# 7 Discussion and conclusions

The paper unifies and clarifies the language and algorithms of various computational approaches associated with tree-decomposition. We present these classes of algorithms by harnessing the formal notation appearing in [14], (which is restricted there to constraint satisfaction). This allows separating tree-structuring from tree-processing. In particular, we provide two variants of tree-processing algorithms (CTE and ICTE) that have different time-space performance qualities. We also show that the bucket-elimination algorithm, BE, can be extended to a message propagation along a specialized tree-decomposition called bucket-tree decomposition yielding algorithm BTE.

The main novelty of this work is that it provides a graph-based unifying framework for tree-decomposition algorithms 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 adds clarity which will benefit researchers in different communities and will accommodate technology transfer. Several of the technical results described here have parallels in these earlier papers, which we cite throughout.

## References

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

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database ochemes. *Journal of the ACM*, 30(3):479–513, 1983.

[3] 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.

[4] R. Dechter. Decomposing a relation into a tree of binary relations. *Journal of Computer and System Sciences, Special Issue on the Theory of Relational Databases*, 41:2–24, 1990.

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

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

[7] R. Dechter and Y. El Fattah. Topological parameters for time-space tradeoff. *Artificial Intelligence*, 125:93–118, 2001.

[8] R. Dechter, K. Kask, and J. Larrosa. A general scheme for multiple lower-bound computation in constraint optimization. *Principles and Practice of Constraint Programming (CP-2001)*, pages 346–360, 2001.

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

[10] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[11] Y. El-Fattah and R. Dechter. An evaluation of structural parameters for probabilistic reasoning: results on benchmark circuits. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 244–251, 1996.

[12] B.J. Frey. *Graphical models for machine learning and digital communication.* MIT press, 1998.

[13] G. Gottlob, N. Leone, and F. Scarello. A comparison of structural csp decomposition methods. *IJCAI-99*, pages 394–399, 1999.

[14] G. Gottlob, N. Leone, and F. Scarello. A comparison of structural csp decomposition methods. *Artificial Intelligence*, pages 243–282, 2000.

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

[16] F.V. Jensen, K.G. Olesen, and S.K. Andersen. An algebra of bayesian belief universes for knowledge based systems. *Networks*, 20:637–659, 1990.

[17] K. Kask and R. Dechter. A general scheme for automatic generation of search heuristics from specification dependencies. *Artificial Intelligence*, 129:91–131, 2001.

[18] D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. *AI and Statistics(AISTAT03)*, 2003.

[19] J. Larrosa. On the time complexity of bucket elimination algorithms. *UCI Technical Report*, 2000.

[20] S.L. Lauritzen and F.V. Jensen. Local computation with valuations from commutative semigroups. *Annals of Mathematics and Artificial Intelligence*, 21:51–69, 1997.

[21] 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.

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

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

[24] K. Kask R. Dechter and R. Mateescu. Iterative join-graph propagation. In *UAI 2002*, pages 128–136, 2002.

[25] N. Robertson and P. Seymour. Graph minors a survey. *In Anderson, editor, Surveys in Combinatorics*, pages 153–171, 1985.

[26] F. Scarello. Private communication. 2004.

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

[28] G.R. Shafer and P.P. Shenoy. Probability propagation. *Anals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

[29] P.P. Shenoy. Binary join trees. *Porceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI96)*, pages 492–499, 1996.

[30] P.P. Shenoy. Binary join trees for computing marginals in the shenoy-shafer architecture. *International Journal of Approximate Reasoning*, 17(2-3):239–263, 1997.

[31] P.P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. *R.D. Shachter, T.S. Levitt, J.F. Lemmer and L.N. Kanal (eds.), Uncertainty in Artificial Intelligence, North-Holland, Amsterdam*, 4:169–198, 1990.

[32] 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.