Weighted Hypertree Decompositions and Optimal Query Plans

[Extended Abstract]

Francesco Scarcello
DEIS
Università della Calabria
87030 Rende - Italy
scarcello@deis.unical.it

Gianluigi Greco
DEIS
Università della Calabria
87030 Rende - Italy
ggreco@si.deis.unical.it

Nicola Leone
Dip. Di Matematica
Università della Calabria
87030 Rende - Italy
leone@mat.unical.it

ABSTRACT

Hypertree width [22, 25] is a measure of the degree of cyclicity of hypergraphs. A number of relevant problems from different areas, e.g., the evaluation of conjunctive queries in database theory or the constraint satisfaction in AI, are tractable when their underlying hypergraphs have bounded hypertree width. However, in practical contexts like the evaluation of database queries, we have more information besides the structure of queries. For instance, we know the number of tuples in relations, the selectivity of attributes and so on. In fact, all commercial query-optimizers are based on *quantitative methods* and do not care about structural properties.

In this paper, we define the notion of *weighted hypertree decomposition*, in order to combine structural decomposition methods with quantitative approaches. Weighted hypertree decompositions are equipped with cost functions, that can be used for modelling many situations where we have further information on the given problem, besides its hypergraph representation. We analyze the complexity of computing the hypertree decompositions having the smallest weights, called minimal hypertree decompositions. We show that, in many cases, adding weights we loose tractability. However, we prove that, under some – not very severe – restrictions on the allowed cost functions and on the target hypertrees, optimal weighted hypertree decompositions can be computed in polynomial time. For some easier hypertree weighting functions, this problem is also highly parallelizable. Then, we provide a cost function that models query evaluation costs and show how to exploit weighted hypertree decompositions for determining (logical) query plans for answering conjunctive queries. Finally, we present the results of an experimental comparison of this query optimization technique with the query optimization of a commercial DBMS. These preliminary results are very promising, as for some large queries (with many joins) our hybrid technique clearly outperforms the commercial optimizer.

PODS 2004 June 14-16, 2004, Paris, France.

1. INTRODUCTION

1.1 Structural Decomposition Methods

Conjunctive queries (CQs) have been studied for a long time in database theory. This class of queries, equivalent in expressive power to the class of Select-Project-Join queries, is probably the most fundamental and most thoroughly analyzed class of database queries. While the evaluation of conjunctive queries is known to be an NP-complete problem in general [8], and in PTIME for the restricted class of acyclic queries [45, 24], several recent papers [9, 32, 23, 13, 29, 28, 30] exploit structural query properties to identify and analyze very large parameterized classes of conjunctive queries whose evaluation is tractable. Note that all these results refer to the *combined complexity* of database queries, where both the database and the query are given in input. In the restricted cases where either the query or the database is fixed, the problem may be easier [43].

The recent renewed interest in tractable classes of conjunctive queries has two main motivations. Firstly, it is well-known that the problem of *conjunctive query containment* is essentially the same as the problem of CQ evaluation [8]. Conjunctive query containment is of central importance in *view-based query processing* [2] which arises, e.g., in the context of data warehousing. Secondly, conjunctive query evaluation is essentially the same problem as *constraint satisfaction*, one of the major problems studied in the field of AI, and there has been a lot of recent interaction between the areas of query optimization and constraint satisfaction (see Vardi's survey paper [44]).

In this paper, we adopt the logical representation of a relational database [42, 1], where data tuples are identified with logical ground atoms, and conjunctive queries are represented as datalog rules. In particular, a *Boolean* conjunctive query (BCQ) is represented by a rule whose head is variable-free, i.e., propositional. Some relevant polynomially solvable classes of conjunctive queries are determined by structural properties of the query hypergraph (or of query graph – the primal graph of the query hypergraph, also called *Gaifman graph*). The *hypergraph* $\mathcal{H}(Q)$ associated to a conjunctive query Q is defined as $\mathcal{H}(Q) = (V, H)$, where the set V of vertices consists of all variables occurring in the body of Q, while the set H of hyperedges contains, for each atom A in the rule body, the set $var(A)$ of all variables occurring in A. As an example, consider the following query
Q_0: $ans \leftarrow s_1(A,B,D) \ \wedge \ s_2(B,C,D) \ \wedge \ s_3(B,E) \ \wedge \ s_4(D,G) \ \wedge s_5(E,F,G) \ \wedge \ s_6(E,H) \ \wedge \ s_7(F,I) \ \wedge \ s_8(G,J)$.
Figure 1 shows its associated hypergraph $\mathcal{H}(Q_0)$.

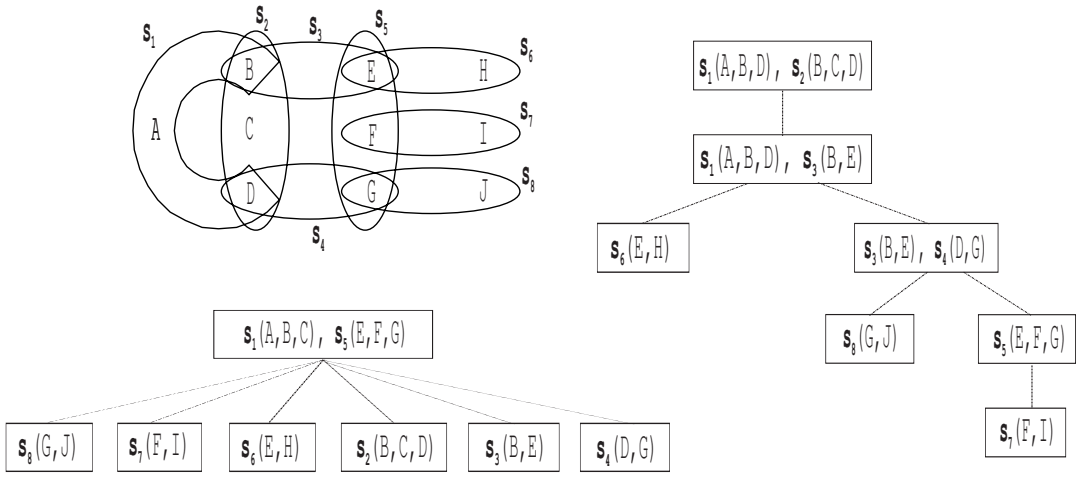

Figure 1: Hypergraph $\mathcal{H}(Q_0)$ (left), two hypertree decompositions of width 2 of $\mathcal{H}(Q_0)$ (right and bottom).

A *structural query decomposition method*[1] is a method of appropriately transforming a conjunctive query into an equivalent tree query (i.e., acyclic query given in form of a join tree) by organizing its atoms into a polynomial number of clusters, and suitably arranging the clusters as a tree (see Figure 1). Each cluster contains a number of atoms. After performing the join of the relations corresponding to the atoms jointly contained in each cluster, we obtain a join tree of an acyclic query which is equivalent to the original query. The resulting query can be answered in output-polynomial time by Yannakakis's well-known algorithm [45]. In case of a Boolean query, it can be answered in polynomial time. The tree of atom-clusters produced by a structural query decomposition method on a given query Q is referred to as the *decomposition* of Q. Figure 1 also shows two possible decompositions of our example query Q_0. A decomposition of Q can be seen as a query plan for Q, requiring to first evaluate the join of each cluster, and then to process the resulting join tree bottom-up (following Yannakakis's algorithm).

The efficiency of a structural decomposition method essentially depends on the maximum size of the produced clusters, measured (according to the chosen decomposition method) either in terms of the number of variables or in terms of the number of atoms. For a given decomposition, this size is referred-to as the *width* of the decomposition. For example, if we adopt the number of atoms, then the width of both decompositions shown in Figure 1 is 2. Intuitively, the complexity of transforming a given decomposition into an equivalent tree query is exponential in its width w. In fact, the evaluation cost of each of the (polynomially many) clusters is bounded by the cost of performing the (at most) w joins of its relations. The overall cost (transformation+evaluation of the resulting acyclic query) is $O(n^{w+1} \log n)$, where n is the size of the input problem, that is, the size of the query and of the database encoding [20].

Therefore, once we fix a bound k for such a width, any structural method D identifies a class of queries that can be answered in polynomial time, namely, all those queries having k-bounded D-width.[2]

[1] In the field of constraint satisfaction, the same notion is known as *structural CSP decomposition method*, cf. [23].

[2] Intuitively, the D-width of a query Q is the minimum width of the decompositions of Q obtainable by method D.

The main structural decomposition methods are based on the notions of Biconnected Components [14], Tree Decompositions [36, 9, 32, 11, 13], Hinge Decompositions [30], and Hypertree Decompositions [22]. Among them, the Hypertree Decomposition Method (HYPERTREE) seems to be the most powerful method, as a large class of cyclic queries has a low hypertree-width, and in fact it strongly generalizes all other structural methods [23]. More precisely, this means that every class of queries that is recognized as tractable according to any structural method D (has k-bounded D-width), is also tractable according to HYPERTREE (has k-bounded HYPERTREE-width), and that there are classes of queries that are tractable according to HYPERTREE, but not tractable w.r.t. D (have unbounded D-width). Moreover, for any fixed $k > 0$, deciding whether a hypergraph has hypertree width at most k is feasible in polynomial time, and is actually highly parallelizable, as this problem belongs to LOGCFL [22] (See Appendix B, for properties and characterizations of this complexity class).

1.2 Limits to the Applicability of Structural Decomposition Methods

Despite their very nice computational properties, all the above structural decomposition methods, including Hypertree Decomposition, are often unsuited for some real-world applications. For instance, in a practical context, one may prefer query plans (i.e., minimum-width decompositions) which minimize the number of clusters having the largest cardinality.

Even more importantly, structural decompositions methods focus "only" on structural query features, while they completely disregard "quantitative" aspects of the query, which may dramatically affect the query-evaluation time. For instance, while answering a query, the computation of an *arbitrary* hypertree decomposition (having minimum width) could not be satisfactory, since it does not take into account important quantitative factors, such as relations sizes, attributes selectivities, and so on. These factors are flattened in the query hypergraph (which considers only the query structure), while their suitable exploitation can significantly reduce the cost of query evaluation. On the other hand, query optimizers of commercial DBMSs are based solely on quantitative methods and do not care of structural properties at all. Indeed, all the commercial DBMSs restrict the search space of query plans to very simple structures (e.g., left-deep trees), and then try to find the best plans

among them, by estimating their evaluation costs according to some cost-model, and exploiting the quantitative information on the input database. It follows that, on some low-width queries with a guaranteed polynomial-time evaluation upper-bound, they may also take time $O(n^\ell)$, which is exponential in the length ℓ of the query, rather than on its width. On some relevant applications with many atoms involved, as for the queries used for populating datawarehouses, this may lead to unacceptable costs. In fact, very often such queries are not very intricate and have low hypertree width, though not necessarily acyclic.

1.3 Contribution of the Paper

To overcome the above mentioned drawbacks, we aim at combining the structural decomposition methods with quantitative approaches.

We thus generalize the notion of HYPERTREE, by equipping hypertree decompositions with polynomial-time weight functions that may encode quantitative aspects of the query database, or other additional requirements. Computing a minimal weighted-decomposition is in general harder than computing a standard decomposition and tractability may be lost. We extensively study the computational complexity of this problem, we prove hardness results and we identify useful tractable cases. In summary, the main contributions of this paper are:

- ▷ We define the notion of *hypertree weighting function* (short: HWF) and of *minimal hypertree decompositions* of a hypergraph $\mathcal{H}$ w.r.t. a given HWF $\omega_{\mathcal{H}}$ and a class of decompositions $\mathcal{C}_{\mathcal{H}}$.
- ▷ We show that computing such a minimal decomposition is NP-hard for general HWFs, even for acyclic hypergraphs.
- ▷ We show that this problem remains NP-hard also if we consider very simple weighting functions, called *vertex aggregation functions*, and restrict the search space to the class of k-bounded hypertree decompositions, for a fixed $k \geq 4$.
- ▷ We prove that, surprisingly, computing minimal hypertree decompositions is tractable if we consider normal-form hypertree decompositions, that are equivalent to the general ones in the unweighted framework. Furthermore, this tractability result holds for a class of functions, called *tree aggregation functions* (TAFs) and defined over semirings, that are much larger than the vertex aggregation functions. We design an algorithm for the efficient computation of such minimal decompositions.
- ▷ We investigate what happens in the (frequent) case where the given TAF is logspace computable, and hence not inherently sequential. We show that deciding whether the minimum weight of normal-form hypertree decompositions is below some given threshold is in LOGCFL and hence highly parallelizable. Moreover, we show that this problem is also hard for this class and we thus get another natural complete problem for this nice complexity class. Note that this result is not obvious, as the LOGCFL-hardness of recognizing (unweighted) k-bounded hypertree decompositions is still unknown. We then prove that computing these minimal decompositions is in the functional version of LOGCFL, that is, $\mathrm{L}^{\mathrm{LOGCFL}}$.
- ▷ We show how the notion of weighted hypertree decomposition can be used for generating effective query plans for the evaluation of conjunctive queries, by combining structural and quantitative information. To this end, we describe a suitable TAF $cost_{\mathcal{H}(Q)}$, which encodes traditional query-plan cost estimations, based on the size of database relations and attributes selectivity.
- ▷ We implemented the algorithm for the computation of the minimal decompositions with respect to $cost_{\mathcal{H}(Q)}$, that correspond to query plans that are optimal w.r.t. our cost-model.
- ▷ We made some preliminary experimental comparisons of the query plans computed by our algorithm with those generated by the internal optimization module of the well known commercial DBMS *Oracle 8.i*. This is not intended to be a thorough comparison with such systems, as we only tried a small set of queries. Our aim here is just to show that exploiting the structure of the query may lead to significant computational savings, and in fact the preliminary results confirm this intuition.

2. HYPERTREE DECOMPOSITIONS AND NORMAL FORMS

We next recall some basic definitions of hypergraphs and hypertree decompositions. For detailed descriptions of the latter notion, see [22, 25].

A *hypergraph* $\mathcal{H}$ is a pair (V, H), where V is a set of vertices and H is a set of hyper-edges such that for each $h \in H$, $h \subseteq V$. For the sake of simplicity, we always denote V and H by $var(\mathcal{H})$ and $edges(\mathcal{H})$, respectively. We use the term *var* because, in our context, hypergraph vertices correspond to query variables. Moreover, for a set of hyperedges S, $var(S)$ denotes the set of variables occurring in S, that is $\bigcup_{h \in S} h$.

A *hypertree for a hypergraph* $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labelling functions which associate each vertex $p \in N$ with two sets $\chi(p) \subseteq var(\mathcal{H})$ and $\lambda(p) \subseteq edges(\mathcal{H})$. If $T' = (N', E')$ is a subtree of T, we define $\chi(T') = \bigcup_{v \in N'} \chi(v)$. We denote the set of vertices N of T by $vertices(T)$, and the root of T by $root(T)$. Moreover, for any $p \in N$, T_p denotes the subtree of T rooted at p.

Definition 2.1 A *hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for $\mathcal{H}$ which satisfies all the following conditions:

1. for each edge $h \in edges(\mathcal{H})$, there exists $p \in vertices(T)$ such that $h \subseteq \chi(p)$ (we say that p *covers* h);
2. for each variable $Y \in var(\mathcal{H})$, the set $\{p \in vertices(T) \mid Y \in \chi(p)\}$ induces a (connected) subtree of T;
3. for each $p \in vertices(T)$, $\chi(p) \subseteq var(\lambda(p))$;
4. for each $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

An edge $h \in edges(\mathcal{H})$ is *strongly covered* in HD if there exists $p \in vertices(T)$ such that $var(h) \subseteq \chi(p)$ and $h \in \lambda(p)$. In this case, we say that p strongly covers h. A hypertree decomposition HD of hypergraph $\mathcal{H}$ is a *complete decomposition* of $\mathcal{H}$ if every edge of $\mathcal{H}$ is strongly covered in HD.

The *width* of a hypertree decomposition $\langle T, \chi, \lambda \rangle$ is $max_{p \in vertices(T)} |\lambda(p)|$. The HYPERTREE *width* $hw(\mathcal{H})$ of

$\mathcal{H}$ is the minimum width over all its hypertree decompositions. A c-width hypertree decomposition of $\mathcal{H}$ is *optimal* if $c = hw(\mathcal{H})$. □

Example 2.2 Consider the following conjunctive query Q_1:

$$\begin{aligned} ans \;\leftarrow\; & a(S,X,X',C,F) \;\wedge\; b(S,Y,Y',C',F') \\ & \wedge\, c(C,C',Z) \;\wedge\; d(X,Z) \;\wedge \\ & e(Y,Z) \;\wedge\; f(F,F',Z') \;\wedge\; g(X',Z') \;\wedge \\ & h(Y',Z') \;\wedge\; j(J,X,Y,X',Y'). \end{aligned}$$

Let $\mathcal{H}_1$ be the hypergraph associated to Q_1. Since $\mathcal{H}_1$ is cyclic, $hw(\mathcal{H}_1) > 1$ holds. Figure 2.a shows a (complete) hypertree decomposition HD_1 of $\mathcal{H}_1$ having width 2, hence $hw(\mathcal{H}_1) = 2$.

In order to help the intuition, Figure 2.b shows an alternative representation of this decomposition, called *atom* (or *hyperedge*) *representation* [22]: each node p in the tree is labeled by a set of atoms representing $\lambda(p)$; $\chi(p)$ is the set of all variables, distinct from '_', appearing in these hyperedges. Thus, in this representation, possible occurrences of the anonymous variable '_' take the place of variables in $var(\lambda(p)) - \chi(p)$.

Another example is depicted in Figure 1, which shows two hypertree decompositions of the query Q_0 in Section 1. Both decompositions have width two and are complete decompositions of Q_0. □

Given any hypertree decomposition HD of $\mathcal{H}$, we can easily compute a complete hypertree decomposition of $\mathcal{H}$ having the same width [22]. Note that the acyclic hypergraphs are precisely those hypergraphs having hypertree width one.

It has been observed that, according to the above definition, a hypergraph may have some (usually) undesirable hypertree decompositions [22]. For instance, a decomposition may contain two vertices with exactly the same labels. Thus, following [22], we next define a normal form for hypertree decompositions that avoids such kind of redundancies.

Let $\mathcal{H}$ be a hypergraph, and let $V \subseteq var(\mathcal{H})$ be a set of variables and $X, Y \in var(\mathcal{H})$. X is $[V]$-adjacent to Y if there exists an edge $h \in edges(\mathcal{H})$ such that $\{X, Y\} \subseteq (h - V)$. A $[V]$-path π from X to Y is a sequence $X = X_0, \ldots, X_\ell = Y$ of variables such that: X_i is $[V]$-adjacent to X_{i+1}, for each $i \in [0...\ell\text{-}1]$. A set $W \subseteq var(\mathcal{H})$ of variables is $[V]$-connected if $\forall X, Y \in W$ there is a $[V]$-path from X to Y. A $[V]$-component is a maximal $[V]$-connected non-empty set of variables $W \subseteq (var(\mathcal{H}) - V)$. For any $[V]$-component C, let $edges(C) = \{h \in edges(\mathcal{H}) \mid h \cap C \neq \emptyset\}$.

Let $HD = \langle T, \chi, \lambda \rangle$ be a hypertree for a hypergraph $\mathcal{H}$. For any vertex v of T, we will often use v as a synonym of $\chi(v)$. In particular, $[v]$-*component* denotes $[\chi(v)]$-*component*; the term $[v]$-path is a synonym of $[\chi(v)]$-path; and so on.

Definition 2.3 A hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of a hypergraph $\mathcal{H}$ is in *normal form* (*NF*) if for each vertex $r \in vertices(T)$, and for each child s of r, all the following conditions hold:

1. there is (exactly) one $[r]$-*component* C_r such that $\chi(T_s) = C_r \cup (\chi(s) \cap \chi(r))$;
2. $\chi(s) \cap C_r \neq \emptyset$, where C_r is the $[r]$-component satisfying Condition 1;
3. for each $h \in \lambda(s)$, $h \cap var(edges(C_r)) \neq \emptyset$, where C_r is the $[r]$-component satisfying Condition 1;
4. $\chi(s) = var(edges(C_r)) \cap var(\lambda(s))$, where C_r is the $[r]$-component satisfying Condition 1. □

Note that, from Condition 4 above, the label $\chi(s)$ of any vertex s of a hypertree decomposition in normal form can be computed just from its label $\lambda(s)$ and from the $[r]$-component satisfying Condition 1 associated with its parent r.

Observe that the above normal form is slightly more stronger than the normal form defined in [22], and allows us to compute hypertree decompositions more efficiently, as some kind of useless vertices are discarded by the new Condition 3. However, we can show that, as for the original definition, the following fundamental result holds.

Theorem 2.4 *For each k-width hypertree decomposition of a hypergraph $\mathcal{H}$ there exists a k-width hypertree decomposition of $\mathcal{H}$ in normal form.*

3. WEIGHTED HYPERTREE DECOMPOSITIONS

In this section, we consider hypertree decompositions with an associated weight, and we analyze the complexity of the main problems related to the computation of the best decompositions.

Formally, given a hypergraph $\mathcal{H}$, a *hypertree weighting function* (short: HWF) $\omega_{\mathcal{H}}$ is any polynomial-time function that maps each hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of $\mathcal{H}$ to a real number, called the *weight* of HD.

For instance, a very simple HWF is the function $\omega^{w}_{\mathcal{H}}(HD) = \max_{p \in vertices(T)} |\lambda(p)|$, that weights a hypertree decomposition HD just on the basis of its worse vertex, that is the vertex with the largest λ label, which also determines the width of the decomposition.

Example 3.1. In many applications, finding a decomposition having the minimum width is not the best we can do. We can think of minimizing the number of vertices having the largest width w and, for decompositions having the same numbers of such vertices, minimizing the number of vertices having width $w-1$, and continuing so on, in a lexicographical way. To this end, we can define the HWF $\omega^{lex}_{\mathcal{H}}(HD) = \sum_{i=1}^{w} |\{p \in N \text{ such that } |\lambda(p)| = i\}| \times B^{i-1}$, where $N = vertices(T)$, $B = |edges(\mathcal{H})|+1$, and w is the width of HD. Note that any output of this function can be represented in a compact way as a radix B number of length w, which is clearly bounded by the number of edges in $\mathcal{H}$.

Consider again the query Q_0 of the Introduction, and the the hypergraph, say HD', of $\mathcal{H}(Q_0)$ shown on the right of Figure 1. Then, $\omega^{lex}_{\mathcal{H}}(HD') = 4 \times 9^0 + 3 \times 9^1$. HD' is not the best decomposition w.r.t. $\omega^{lex}_{\mathcal{H}}$ and the class of hypertree decompositions in normal form; for instance, the decomposition HD'' shown on the bottom of Figure 1 is better than HD', as $\omega^{lex}_{\mathcal{H}}(HD'') = 6 \times 9^0 + 1 \times 9^1$.

Note that these examples are still based on structural criteria only, as no information on the input database is exploited. In fact, in the

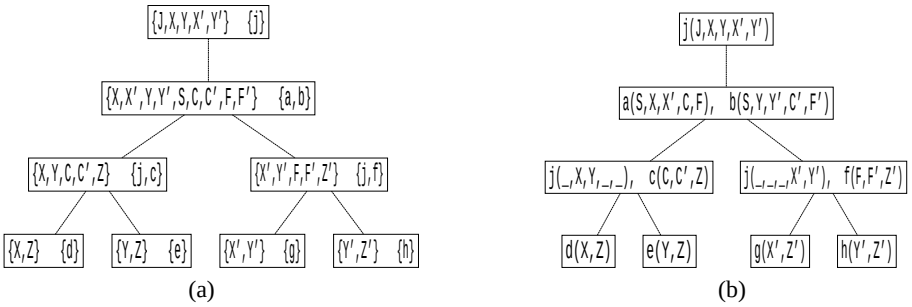

Figure 2: (a) A width 2 hypertree decomposition of $\mathcal{H}_1$, and (b) its atom representation

next section, we will define a more sophisticated hypertree weighting function, specifically designed for query evaluation. □

Since computing a minimum-width hypertree decomposition is NP-hard, it is easy to see that, in the general case, finding the hypertree decompositions having the minimum weight is NP-hard, too. However, we are usually interested only in decompositions belonging to some tractable class, for instance the hypertree decompositions having small width.

Let $k > 0$ be a fixed integer and $\mathcal{H}$ a hypergraph. We define $kHD_{\mathcal{H}}$ (resp., $kNFD_{\mathcal{H}}$) as the class of all hypertree decompositions (resp., normal-form hypertree decompositions) of $\mathcal{H}$ having width at most k. We recall that, given a hypergraph $\mathcal{H}$, deciding whether $kNFD_{\mathcal{H}} \neq \emptyset$ (and hence $kHD_{\mathcal{H}} \neq \emptyset$, after Theorem 2.4) is in LOGCFL [22].

Definition 3.2 Let $\mathcal{H}$ be a hypergraph, $\omega_{\mathcal{H}}$ a weighting function, and $\mathcal{C}_{\mathcal{H}}$ a class of hypertree decompositions of $\mathcal{H}$. Then, a hypertree decomposition $HD \in \mathcal{C}_{\mathcal{H}}$ is *minimal* w.r.t. $\omega_{\mathcal{H}}$ and $\mathcal{C}_{\mathcal{H}}$, denoted by $[\omega_{\mathcal{H}}, \mathcal{C}_{\mathcal{H}}]$-$minimal$, if there is no $HD' \in \mathcal{C}_{\mathcal{H}}$ such that $\omega_{\mathcal{H}}(HD') < \omega_{\mathcal{H}}(HD)$. □

For instance, the $[\omega^{w}_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decompositions are exactly the k-bounded optimal hypertree decompositions in Definition 2.1, while the $[\omega^{lex}_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decompositions correspond to the lexicographically minimal decompositions described above.

It is not difficult to show that, for general weighting functions, the computation of minimal hypertree decompositions is a difficult problem even if we consider just bounded hypertree decompositions.

Theorem 3.3 *Given a hypergraph $\mathcal{H}$ and an* HWF *$\omega_{\mathcal{H}}$, computing a $[\omega_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decomposition (if any) is NP-hard. Hardness holds even for acyclic hypergraphs ($k = 1$).*

PROOF SKETCH. The reduction is from the graph coloring problem. Given a graph G, we can build an acyclic hypergraph $\mathcal{H}(G)$ and a HWF $\omega_{\mathcal{H}(G)}$ such that the $[\omega_{\mathcal{H}(G)}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decompositions correspond to legal colorings of G. Intuitively, the acyclic hypergraph has many possible join trees, and the idea is to exploit their shapes in such a way that join trees corresponding to legal colorings have weight 0, while all other join trees have weight 1. □

In this case, the main source of complexity is the HWF, which can evaluate hypertree decompositions looking at the whole tree, and weighting for instance the shape of the tree or other arbitrary relationships among vertices.

One may thus wonder whether, restricting our attention to simpler HWFs our problem becomes any easier. Let $\mathcal{H}$ be a hypergraph. A *vertex aggregation function* is a weighting function of the form $\Lambda^{v}_{\mathcal{H}}(HD) = \sum_{p \in vertices(T)} v_{\mathcal{H}}(p)$, where $v_{\mathcal{H}}$ is any polynomial time function that associates a real number to any vertex of the hypertree decomposition HD.

Therefore, in vertex aggregation functions, all the power is in the local (restricted to single vertices) function $v_{\mathcal{H}}$, and the weighting function just returns the sum of such values.

For instance, if we let $v^{lex}_{\mathcal{H}}(p) = B^{|\lambda(p)|-1}$, where $B = |edges(T)| + 1$, then the vertex aggregation function $\Lambda^{v^{lex}}_{\mathcal{H}}$ is exactly the HWF $\omega^{lex}_{\mathcal{H}}$ described in Example 3.1, that allows us to single out the lexicographically minimal hypertree decompositions.

Unfortunately, the next result shows that even in this restricted setting computing minimal decompositions is NP-hard.

Theorem 3.4 *Given a hypergraph $\mathcal{H}$ and a vertex aggregation function $\Lambda^{v}_{\mathcal{H}}$, computing a $[\Lambda^{v}_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decomposition of $\mathcal{H}$ (if any) is NP-hard.*

PROOF SKETCH. Let $\mathcal{H}$ be a hypergraph. We reduce the problem of deciding whether $\mathcal{H}$ has a query decomposition of width at most k [9] to the problem of computing a $[\Lambda^{v}_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decomposition of $\mathcal{H}$. Recall that, for any fixed $k \geq 4$, deciding the existence of a query decomposition for $\mathcal{H}$ of width at most k is NP-hard [22] (see Appendix A).

We can show that there is a query decomposition for $\mathcal{H}$ of width w if and only if there is a hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of $\mathcal{H}$ having width w and such that, for each $p \in vertices(T)$, $\chi(p) = var(\lambda(p))$.

Then, for any hypertree decomposition $HD' = \langle T', \chi', \lambda' \rangle$ of $\mathcal{H}$, we define the vertex evaluation function $v_{\mathcal{H}}$ as follows: for each $p \in vertices(T')$, $v_{\mathcal{H}}(p) = |var(\lambda'(p)) - \chi'(p)|$.

It follows that the weight of the $[\Lambda^{v}_{\mathcal{H}}, kHD_{\mathcal{H}}]$-$minimal$ hypertree decompositions of $\mathcal{H}$ is 0 if and only if there is a query decomposition for $\mathcal{H}$ of width at most k. Therefore computing any such a minimal decomposition amounts to solving the query decomposi-

tion problem instance. □

Thus, restricting the kind of hypertree weighting functions is not sufficient to ensure tractability. Indeed, the above result shows that even the search space of all k-bounded hypertree decompositions is too wide to be efficiently explored. We need a further restriction of this class of decompositions.

4. EFFICIENTLY-COMPUTABLE MINIMAL HYPERTREE DECOMPOSITIONS

In this section, we show that, by slightly restricting the class of allowed decompositions, it is possible to compute in polynomial time any minimal weighted hypertree decomposition, even with respect to hypertree weighting functions more general than the vertex aggregation functions described above.

In fact, we would like to use HWFs whose evaluation of a hypertree decomposition does not depend only on the vertices as isolated entities, but also on the relationships between any vertex and its children in the tree. Moreover, we can think of other operators besides the simple summation.

Let $\langle \mathbb{R}^+, \oplus, \min, \bot, +\infty \rangle$ be a *semiring*, that is, $\oplus$ is a commutative, associative, and closed binary operator, $\bot$ is the neuter element for $\oplus$ (e.g., 0 for $+$, 1 for $\times$, etc.) and the absorbing element for $\min$, and $\min$ distributes over $\oplus$.[3] Given a function g and a set of elements $S = \{p_1, ..., p_n\}$, we denote by $\bigoplus_{p_i \in S} g(p_i)$ the value $g(p_1) \oplus \ldots \oplus g(p_n)$.

Definition 4.1 Let $\mathcal{H}$ be a hypergraph. Then, a *tree aggregation function* (short: TAF) is any hypertree weighting function of the form

$$\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}(HD) = \bigoplus_{p \in N} \big(v_{\mathcal{H}}(p) \ \oplus \bigoplus_{(p,p') \in E} e_{\mathcal{H}}(p,p')\big),$$

associating an $\mathbb{R}^+$ value to the hypertree decomposition $HD = \langle (N, E), \chi, \lambda \rangle$, where $v_{\mathcal{H}} : N \mapsto \mathbb{R}^+$ and $e_{\mathcal{H}} : N \times N \mapsto \mathbb{R}^+$ are two polynomial functions evaluating vertices and edges of hypertrees, respectively. □

Note that every vertex aggregation function corresponds to a tree aggregation function with the same $v_{\mathcal{H}}$, with $\oplus = +$, and the constant function $\bot$ as the edge evaluation function $e_{\mathcal{H}}$.

Example 4.2. Another simple example of tree aggregation function is $\mathsf{F}^{\max,v^w,\bot}_{\mathcal{H}}(HD)$, where $v^w_{\mathcal{H}}(p) = |\lambda(p)|$. Observe that this TAF is equal to ω^w, and thus its minimal decompositions are those having the minimum possible width.

In some applications it could also be useful to minimize the size of the largest vertex separator in HD, where a separator $sep(p,q)$ is defined as $\chi(p) \cap \chi(q)$ [11]. This can be easily obtained by using the tree aggregation function $\mathsf{F}^{\max,\bot,e^{sep}}_{\mathcal{H}}(HD)$, where $e^{sep}_{\mathcal{H}}(p,q) = |sep(p,q)|$. Of course, a more sophisticated minimization of the size of separators may be obtained using a lexicographical criterion as for the width, through the TAF $\mathsf{F}^{+,\bot,e^{lsep}}_{\mathcal{H}}(HD)$, where $e^{lsep}_{\mathcal{H}}(p,q) = (|N|+1)^{|sep(p,q)|-1}$, and N is the set of vertices of the decomposition tree of HD. □

[3] For the sake of presentation, we refer to $\min$ and hence to minimal hypertree decompositions. However, it is easy to see that all the results presented in this paper can be generalized easily to any semiring, possibly changing $\min$, $\mathbb{R}^+$, and $+\infty$.

Observe that in the above examples we used either the vertex evaluation function or the edge evaluation function. By exploiting both functions $v_{\mathcal{H}}$ and $e_{\mathcal{H}}$, we can obtain more sophisticated and powerful tree aggregation functions.

Example 4.3 Given a query Q over a database **DB**, let $HD = \langle T, \chi, \lambda \rangle$ be a hypertree decomposition in normal form for $\mathcal{H}(Q)$. For any vertex p of T, let $E(p)$ denote the relational expression $E(p) = \ \bowtie_{h \in \lambda(p)} \ \prod_{\chi(p)} rel(h)$, i.e., the join of all relations in **DB** corresponding to hyperedges in $\lambda(p)$, suitably projected onto the variables in $\chi(p)$. Given also an incoming node p' of p in the decomposition HD, we define $v^*_{\mathcal{H}(Q)}(p)$ and $e^*_{\mathcal{H}(Q)}(p,p')$ as follows:

- $v^*_{\mathcal{H}(Q)}(p)$ is the estimate of the cost of evaluating the expression $E(p)$, and
- $e^*_{\mathcal{H}(Q)}(p,p')$ is the estimate of the cost of evaluating the semi-join $E(p) \ltimes E(p')$.

Let $cost_{\mathcal{H}(Q)}$ be the TAF $\mathsf{F}^{+,v^*,e^*}_{\mathcal{H}(Q)}(HD)$, determined by the above functions. Intuitively, $cost_{\mathcal{H}(Q)}$ weights the hypertree decompositions of the query hypergraph $\mathcal{H}(Q)$ in such a way that minimal hypertree decompositions correspond to "optimal" query evaluation plans for Q over **DB**. We will come back to this TAF in Section 5, which is devoted to the relationship between query optimization and minimal hypertree decompositions.

Note that any method for computing the estimates for the evaluation of relational algebra operations, from the quantitative information on **DB** (relations sizes, attributes selectivity, and so on), may be employed for v^* and e^*. In particular, in our experiments with such minimal hypertree decompositions reported in Section 5, we adopt the standard techniques described, e.g., in [16, 18]. □

Clearly, all this powerful weighting functions would be of limited practical applicability, without a polynomial time algorithm for the computation of minimal hypertree decompositions. Surprisingly, we show that, unlike the traditional (non-weighted) framework, working with normal-form hypertree decompositions, rather than with any kind of bounded-width hypertree decomposition, does matter. Indeed, it turns out that computing such minimal hypertree decompositions with respect to any tree aggregation function is a tractable problem. A polynomial time algorithm for this problem, called `minimal-`k`-decomp`, is shown in Figure 3.

Theorem 4.4 *Given a hypergraph $\mathcal{H}$ and a* TAF $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$*, computing an* $[\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}, kNFD_{\mathcal{H}}]$*-minimal hypertree decomposition of $\mathcal{H}$ (if any) is feasible in polynomial time.*

PROOF SKETCH. We first illustrate the method for computing minimal hypertree decompositions implemented in Algorithm `minimal-`k`-decomp`. Then, we show that this algorithm has a polynomial-time complexity, but we omit here the (quite involved) formal proof of its soundness and completeness.

Input: A hypergraph $\mathcal{H}$, a tree aggregation function $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$.
Output: An $[\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}, kNFD_{\mathcal{H}}]$-$minimal$ hypertree decomposition of $\mathcal{H}$, if any; otherwise, *failure*.
Var $CG = (N_{sol} \cup N_{sub}, A, \mathit{weight})$: weighted directed bipartite graph;
$HD = \langle (N_{sol}, E), \chi, \lambda \rangle$: hypertree of $\mathcal{H}$;
Begin
(*Build the Candidates Graph*)
$N_{sub} := \{(\emptyset, var(\mathcal{H}))\} \cup \{(R, C) \mid R$ is a k-vertex and C is an $[R]$-component $\}$;
$N_{sol} := \{(S, C) \mid S$ is a k-vertex, C is any $[R]$-$component$ (for some R), $var(S) \cap C \neq \emptyset$ and, $\forall h \in S, h \cap var(edges(C)) \neq \emptyset\}$;
$A := \emptyset$;
For each $(R, C) \in N_{sub}$ **Do**
For each $(S, C) \in N_{sol}$ such that $var(edges(C)) \cap var(R) \subseteq var(S)$ **Do**
Add an arc from (S, C) to (R, C) in A;
For each $(S, C') \in N_{sub}$ s.t. $C' \subset C$ **Do** (* Connect its subproblems *)
Add an arc from (S, C') to (S, C) in A;
(* Evaluate the Candidates Graph *)
For each $p = (S, C) \in N_{sol}$ **Do**
$\lambda(p) := S$; $\chi(p) := var(edges(C)) \cap var(S)$, and $\mathit{weight}(p) := v_{\mathcal{H}}(p)$;
$weigthed := \{p \in N_{sol} \mid incoming(p) = \emptyset\}$;
$toBeProcessed := N_{sub}$;
While $toBeProcessed \neq \emptyset$ **Do**
Extract a node q from $toBeProcessed$ such that $incoming(q) \subseteq weighted$;
If $incoming(q) = \emptyset$ **Then** (* no way to solve subproblem q *)
Remove all $p' \in outcoming(q)$;
Else (* success, for this subproblem *)
For each $p' \in outcoming(q)$ **Do**
$\mathit{weight}(p') := \mathit{weight}(p') \oplus \min_{p \in incoming(q)} (\mathit{weight}(p) \oplus e_{\mathcal{H}}(p', p))$;
If $incoming(p') \cap toBeProcessed = \emptyset$ **Then**
Add p' to $weigthed$;
EndWhile (* $toBeProcessed$ *)
(* Identify a minimal weighted hypertree decomposition (if any)*)
If $incoming((\emptyset, var(\mathcal{H}))) = \emptyset$ **Then**
Output *failure*;
Else
$E := \emptyset$; (* the tree of the decomposition has no edges, initially *)
Choose a minimum-weighted $p \in incoming((\emptyset, var(\mathcal{H})))$;
Select-hypertree(p);
Remove all isolated vertices in the tree (N_{sol}, E);
Output HD;
End.

Procedure *Select-hypertree*($p \in N_{sol}$)
For each $q \in incoming(p)$ **Do**
Choose a minimum-weighted $p' \in incoming(q)$;
Add the edge $\{p, p'\}$ to E;
Select-hypertree(p');
EndProcedure;

Figure 3: ALGORITHM `minimal-`k`-decomp`

The algorithm `minimal-`k`-decomp` maintains a weighted directed bipartite graph CG, called the *Candidates Graph*, that collects all information we need for computing the desired decompositions. Its nodes are partitioned in two sets N_{sub} and N_{sol}, representing the subproblems that we have to solve and the candidates to their solutions, respectively. Nodes in N_{sub} have the form (R, C), where R is a set of at most k edges of $\mathcal{H}$, called a k-vertex, and C is an $[R]$-$component$. Moreover, there is a special node $(\emptyset, var(\mathcal{H}))$ that represents the whole problem. Nodes in N_{sol} have the form (S, C'), where S is a k-vertex, C' is a component to be decomposed, $var(S) \cap C' \neq \emptyset$ and, $\forall h \in S$, $h \cap var(edges(C')) \neq \emptyset$. Intuitively, this node could be the root of a hypertree decomposition for the sub-hypergraph induced by $var(edges(C'))$. The node (S, C') has an arc pointing to all nodes of the form $(R', C') \in N_{sub}$ for which it is a candidate solution, that is, for which $var(edges(C')) \cap var(R') \subseteq var(S)$ holds. Moreover, it has a number of incoming nodes of the form $(S, C'') \in N_{sub}$, for each $[S]$-$component$ C'' that is included in C', as each of these nodes represents a subproblem of (S, C') (or, more precisely, of any $(R', C') \in N_{sub}$ the node (S, C') is connected to).

For every node $p \in N_{sol}$, we initially set $\mathit{weight}(p) := v_{\mathcal{H}}(p)$. Then, since $\oplus$ is associative, commutative, and closed, we can update this weight by setting $\mathit{weight}(p) := \mathit{weight}(p) \oplus e_{\mathcal{H}}(p, p')$, as soon as we know any descendant p' of p in the decomposition tree. These descendants are obtained by a suitable filtering of the nodes connected to its incoming nodes in N_{sub}, corresponding to its subproblems.

If a node $q \in N_{sub}$ has no candidate solutions, i.e., if $incoming(q) = \emptyset$, then it is not solvable. We immediately exploit this information by removing all of its outcoming nodes, for which it was a subproblem. On the other hand, if it has some candidates, whenever all of them have been completely evaluated, it can propagate this information to its outcoming nodes. Then, since $\min$ distributes over $\oplus$, we can safely select as its solution its minimum-weighted incoming node in N_{sol}.

Once all nodes have been processed, the information encoded in the weighted graph CG is enough to compute every minimal hypertree decomposition of $\mathcal{H}$ in normal form having width at most k, if any. One of these hypertrees is eventually selected through the simple recursive procedure *Select-hypertree*.

We next briefly discuss the complexity of `minimal-`k`-decomp`. Let $\mathcal{H}$ be a hypergraph and $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$ a TAF, and let n and m be the number of edges and the number of vertices of $\mathcal{H}$, respectively. Moreover, let $c_{\oplus}$, $c_{\min}$, c_v, and c_e be the maximum costs of evaluating the operators $\oplus$ and $\min$, and the functions $v_{\mathcal{H}}$ and $e_{\mathcal{H}}$, respectively, for the given problem instance. We denote by Ψ the number of k-vertices of $\mathcal{H}$, that is, $\Psi = \sum_{i=1}^{k} \binom{n}{i} = \sum_{i=1}^{k} \frac{n!}{i!(n-i)!}$. Note that, for each k-vertex R, there are at most $O(m)$ $[R]$-components. Therefore, the graph CG has $O(\Psi\ m)$ nodes in N_{sub} and $O(\Psi^2 m)$ nodes in N_{sol}. Moreover, each node in N_{sub} has $O(\Psi)$ incoming arcs at most, and each node in N_{sol} has $O(m)$ incoming arcs at most. Then, it can be checked that building CG costs $O(\Psi^2 m^2)$, and computing the weights according to `minimal-`k`-decomp` costs $O(\Psi^2 m^2 c_{\oplus} + \Psi^2 m c_e c_{\min} + \Psi^2 m c_v)$. Thus, an upper bound of the overall complexity is given by the latter expression. A very inaccurate but more readable upper bound can be obtained by observing that, clearly, Ψ is $O(n^k)$. However, for practical purposes, it is worthwhile noting that these two values differ significantly. (E.g., for $k = 3$ and $n = 5$, $n^k = 125$, while $\Psi = 25$; for $k = 4$ and $n = 10$, $n^k = 10000$, while $\Psi = 385$.) □

Furthermore, we observe that, by restricting a little bit the power of tree aggregation functions, the problem becomes also highly parallelizable.

Of course, this is not possible without any restriction, as the weighting function may be a P-complete function, in general. We say that a TAF $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$ is *smooth* if can be evaluated in logspace. More precisely, if $\oplus$, $v_{\mathcal{H}}$ and $e_{\mathcal{H}}$ are computable in $O(\log(\| \mathcal{H} \|))$ and the size of their outputs is $O(\log(\| \mathcal{H} \|))$.

Note that this is a wide class of functions, comprising many interesting TAFs. In fact, all the hypertree weighting functions described so far in this paper, but $cost_{\mathcal{H}(Q)}$, are smooth. For instance, counting the size of a separator is feasible in logspace, and encoding such a number requires logspace in the size of the given hypergraph.

We first consider the problem of deciding whether there is a normal-form hypertree decomposition HD of $\mathcal{H}$ such that $\mathsf{F}^{\oplus,v,e}(HD) \leq t$, for some given threshold $t \geq 0$. Interestingly, we show that this problem is LOGCFL-complete and hence in NC^2 (see Appendix B, for details on this complexity class, and for its characterization in terms of alternating Turing machines). We thus have a new nice natural complete problem for this class, after the recent results in [20]. We remark that we miss such a result for traditional (unweighted) hypertree decompositions (and, similarly, for the notion of bounded treewidth). Indeed, we know that deciding whether $\mathcal{H} \in kHD_{\mathcal{H}}$ is in LOGCFL [22], but the hardness for this class has not been proven, and is still an open problem.

Theorem 4.5 *Given a hypergraph $\mathcal{H}$, a smooth* TAF $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$*, and a number $t \geq 0$, deciding whether there is a hypertree decomposition $HD \in kNFD_{\mathcal{H}}$ such that $\mathsf{F}^{\oplus,v,e}(HD) \leq t$ is LOGCFL-complete. Hardness holds even for acyclic hypergraphs.*

PROOF SKETCH. *Membership.* We build a logspace alternating Turing machine M with a polynomially-bounded computation tree that decides this problem. Intuitively, any existential configuration of M is used for guessing both a set of k-edges that is candidate to belong to some decomposition, and a maximum weight for the sub-hypertree rooted at this node. The only exception is the starting configuration, where the weight is not guessed, but just initialized with the given threshold t (suitably encoded through a logspace pointer to the input tape).

Hardness. We describe a logspace reduction from the LOGCFL-complete problem of answering an acyclic Boolean conjunctive query Q over a database **DB**. We build a hypergraph $\mathcal{H}$ that represents not only the structure of Q (as $\mathcal{H}(Q)$), but also the database **DB**. In particular, $\mathcal{H}$ has an edge for each tuple in **DB**, and the edge function $e_{\mathcal{H}}$ will be used to check whether two given nodes encode matching tuples or not. Minimal hypertree decompositions of this hypergraph will correspond to the choice of exactly one tuple from each query atom, and the minimum weight will be 0 is and only if the answer of Q over **DB** is true. □

By using the alternating Turing machine in the proof above as an oracle of a logspace machine, and by exploiting the results in [26] on the computation of LOGCFL certificates, we get the analogous result for the problem of computing minimal hypertree decompositions w.r.t. smooth TAFs.

Theorem 4.6 *Given a hypergraph $\mathcal{H}$ and a smooth* TAF $\mathsf{F}^{\oplus,v,e}_{\mathcal{H}}$*, computing an $[\mathsf{F}^{\oplus,v,e}, kNFD_{\mathcal{H}}]$-minimal hypertree decomposition of $\mathcal{H}$ (if any) is feasible in* $\mathrm{L}^{\mathrm{LOGCFL}}$.

It is easy to see that all the above tractability results also hold for some further restrictions of the class of NF bounded-width hypertree decompositions, such as the bounded-width decompositions in *reduced normal form*, recently defined in [17].

5. MINIMAL DECOMPOSITIONS AND OPTIMAL QUERY PLANS: SOME EXPERIMENTS

In this section, we exploit the previous results for the efficient evaluation of database queries and we carry out some experiments.

Recall that $cost_{\mathcal{H}(Q)}$ is the TAF $\mathsf{F}^{+,v^*,e^*}_{\mathcal{H}(Q)}$ described in Example 4.3, and let `cost-`k`-decomp` the specialization of `minimal-`k`-decomp` implementing $cost_{\mathcal{H}(Q)}$. Then, any $[cost_{\mathcal{H}(Q)}, kNFD_{\mathcal{H}(Q)}]$-*minimal* weighted hypertree decomposition, which can be computed by the algorithm `cost-`k`-decomp`, represents an effective plan for evaluating Q and is in fact an optimal plan according to the given cost model (and to the class of k-bounded NF hypertree decompositions).

In [22], it is evidenced that for answering queries we need complete decompositions, where each atom occurs at least once (see Definition 2.1), but NF decompositions are not necessarily complete. Nevertheless, this property can be easily guaranteed by adding a fresh variable to each query atom. The algorithm `cost-`k`-decomp`, available at the hypertree decomposition homepage [41], also deals with these issues, by suitably modifying the query and then filtering such fresh variables, in its output phase.

Example 5.1 Consider again the following conjunctive query Q_1,

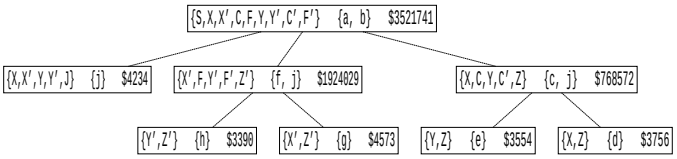

Figure 4: A minimal weighted optimal hypertree decomposition of width 2 for Q_1

defined in Example 2.2:

$$ans \leftarrow a(S,X,X',C,F) \wedge b(S,Y,Y',C',F') \wedge c(C,C',Z) \wedge d(X,Z) \wedge e(Y,Z) \wedge f(F,F',Z') \wedge g(X',Z') \wedge h(Y',Z') \wedge j(J,X,Y,X',Y').$$

Assume we have to evaluate this query on a database **DB**, whose quantitative statistics information are reported in Table 1, which shows for each atom p occurring in Q_1, the number of tuples in the relation $rel(p)$ associated with p, denoted by $|p|$, and for each variable X occurring in p, the selectivity of the attribute corresponding to X in $rel(p)$, i.e., the number of *distinct* values X takes on $rel(p)$. These data are obtained by means of the command *ANALYZE TABLE* in *Oracle 8.i*.

atom a, $\|a\| = 4606$	S	X	X'	C	F
SELECTIVITY	14	24	16	21	15

atom b, $\|b\| = 2808$	S	Y	Y'	C'	F'
SELECTIVITY	17	5	12	20	7

atom c, $\|c\| = 1748$	C	C'	Z'
SELECTIVITY	18	7	19

atom d, $\|d\| = 3756$	X	Z
SELECTIVITY	18	7

atom d, $\|d\| = 3554$	Y	Z
SELECTIVITY	21	13

atom f, $\|f\| = 2892$	F	F'	Z'
SELECTIVITY	20	7	6

atom g, $\|g\| = 4573$	X'	Z'
SELECTIVITY	22	16

atom h, $\|h\| = 3390$	Y'	Z'
SELECTIVITY	15	12

atom j, $\|j\| = 4234$	J	X	Y	X'	Y'
SELECTIVITY	18	8	18	22	10

Table 1: Cardinality and selectivity for relations in query Q_1.

Figure 4 shows a minimal hypertree decomposition of Q_1 (w.r.t. **DB**) computed by `cost-`k`-decomp`, if we fix the bound k to 2. In the figure, each vertex v has a new label marked by the symbol **$** that represents the estimated cost for evaluating the subtree rooted at v. In particular, for each leaf ℓ, this number will be equal to the estimate for the cost of the relational expression $E(\ell)$ (see Example 4.3); for the root r of the hypertree, this number gives the estimated cost of the whole evaluation of Q_1(**DB**). In our example, this cost is 3521741.

Recall that Q_1 is a cyclic query, having hypertree width 2, thus there is no hypertree decomposition of width 1 for Q_1, and $k = 2$ is the lowest bound such that `cost-`k`-decomp` is able to compute decomposition for Q_1. However, any value larger than 2 is feasible, and so we have run `cost-`k`-decomp` with k ranging from 2 to 5. For $k = 3$, we obtain the estimated cost 1373879, while for both $k = 4$ and $k = 5$ we obtain 854867. Figure 5 shows a minimal hypertree decomposition computed by `cost-`k`-decomp`, with $k = 4$.

It turns out that, even if the hypertree width of Q_1 is 2, for the given quantitative information on **DB** and the cost model we have chosen,

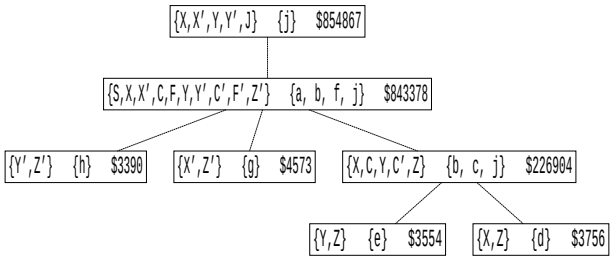

Figure 5: A minimal weighted hypertree decomposition of width 4 for Q_1

the bound 4 leads to the best query plans. This is not surprising, as a larger bound k allows us to explore more decompositions, containing larger vertices with more associated join operations. □

In order to assess the efficacy of the query plans generated by `cost-`k`-decomp`, we compare their performances against the ones of the plans generated by the internal optimization module of the well known commercial DBMS *Oracle 8.i*, for different types of queries. In our experiments, we mainly use randomly generated synthetic data. Moreover, in this preliminary experimentation activity, we did not allow indices on the relations, in order to focus just on the less-physical aspects of the optimization task.

The experiments have been carried out by using *Oracle 8.i* as evaluation engine. Thus, each query used for the comparison is evaluated over *Oracle 8.i* twice: (i) by using its internal optimization module, allowing the exploitation of all the quantitative information available for the database (by means of the command *ANALYZE TABLE*, invoked for each table); (ii) with the query plan generated by `cost-`k`-decomp`, whose execution is enforced by supplying a suitable translation in terms of views and hints (*NO_MERGE, ORDERED*) to *Oracle 8.i*. Experiments with `cost-`k`-decomp` have been conducted for k ranging over $(2..5)$. The experiments have been performed on a 1600MHz/256MB Pentium IV machine running *Windows XP Professional*.

The results of experiments are displayed in Figure 6.

We use the algorithm `cost-`k`-decomp` for computing a cost-based hypertree decomposition HD whose associated query plan for answering Q_1 on **DB** guarantees the minimum estimated cost over all (normal form) k-bounded hypertree decomposition of Q_1. We also report experiments for queries q_2 and Q_3. Query Q_2 consists of 8 atoms and 9 distinct variables, and query Q_3 is made of 9 atoms, 12 distinct variables, and 4 output variables. On all considered queries, the evaluation of the query plans generated by our approach is significantly faster than the evaluation which exploits the internal query optimization module of *Oracle 8.i*.

We experimented `cost-`k`-decomp` for different values of the hypertree width k (2..5). A higher value of k considers a larger number of hypertree decompositions; it can therefore generate a better plan, but obviously pays a computational overhead. For query Q_1, we report the costs of constructing the plan, the weight of the plan, and the evaluation time (sec), for different values of k (2..5). The Figure 6.(A) displays the ratio between the query evaluation times (Oracle vs `cost-`k`-decomp`). Such a ratio increases for higher values of k. Queries Q_2 and Q_3 show a similar behaviour of this

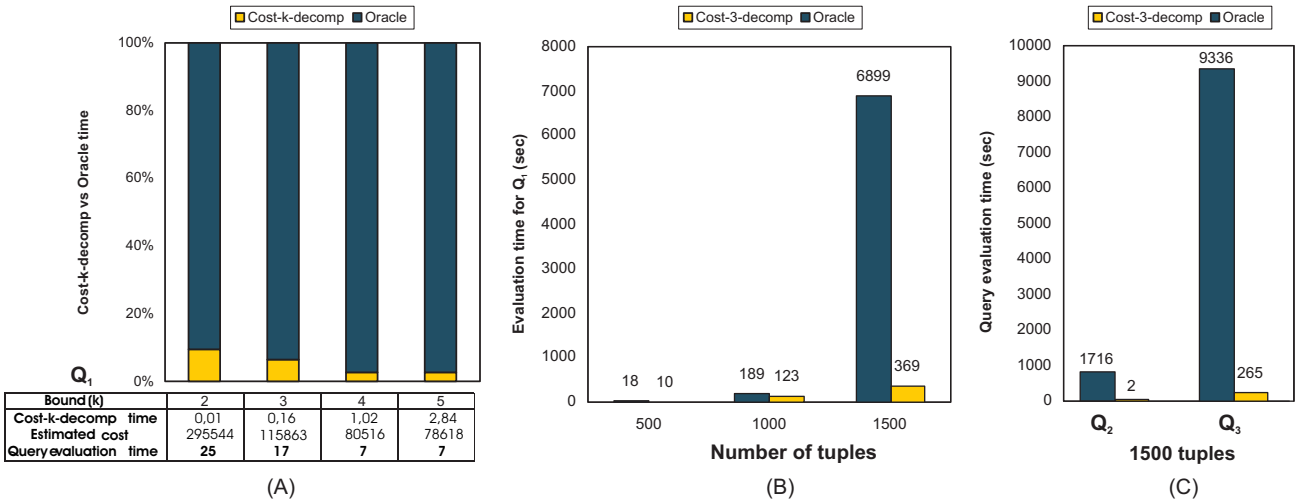

Figure 6: *Cost-k-decomp* vs *Oracle*: (A) Results for different values of k. (B) Scaling of the algorithm for query Q_1. (C) Evaluation time for queries Q_2 and Q_3.

ratio, and we report only the (absolute) query evaluation times for $k = 3$ (*Cost-3-decomp*), which appears to be a good trade-off between the better quality of the plan and the overhead needed for its computation.

6. CONCLUSION

We have presented an extension of the notion of hypertree decomposition, where hypertrees are weighted by some suitable functions, and we want to compute the hypertree decompositions having the minimum weight. This is a natural generalization of the optimal hypertree decompositions in [22], that are the smallest width decompositions of a given query hypergraph.

The new notion have many possible applications, in all the areas where structural decomposition methods may be useful. Unfortunately, we prove that even for very simple weighting functions, the computation of a minimal decomposition is an NP-hard task. However, if we restrict our search space to bounded width decompositions in normal form, the problem is feasible in polynomial time (and in some cases also parallelizable), for a large and interesting class of weighting functions.

In particular, we then focus on a weighting function such that minimal hypertree decompositions of a query Q correspond to the best plans for evaluating Q. Thus, we get a new hybrid technique for query planning, which combines structural decomposition methods with the quantitative methods of commercial DBMS. The idea is to take advantage of both the information on the data and the structure of the query, in order to have a statistically good execution plan with a polynomial-time upper bound on the execution cost, guaranteed by the bounded hypertree-width of the query. We described and implemented an algorithm for computing query plans according to this new technique. Also, we made some preliminary experiments, to show that hybrid methods may lead to significant computational savings. In fact, the results are very promising, showing that the proposed approach clearly outperforms the traditional quantitative-only optimizers, for many queries involving a certain number of atoms (more than five) and not very intricate (that is, having low hypertree width).

Future work will concern a thorough experimentation activity, to compare our technique against other approaches presented in the literature, or implemented in the most advanced commercial database systems. In particular, we plan to experiment with real queries and databases, loaded with non-random data.

Another interesting question to be experimentally evaluated is the true impact of restricting our attention to the tractable class of normal form hypertree decompositions. That is, we want to investigate how the minimum weight in this restricted case is far from the minimum weight over all hypertree decompositions, in real applications.

Acknowledgments

We sincerely thank Alfredo Mazzitelli for his valuable work in implementing and testing the algorithm proposed in the paper, and for designing the tools for experimenting with Oracle.

The research was supported by the European Commission under the INFOMIX project (IST-2001-33570).

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Abiteboul and O.M. Duschka. Complexity of Answering Queries Using Materialized Views. In *Proc. of the PODS'98, Seattle, Washington*, pp. 254–263, 1998.

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

[4] P.A. Bernstein and N. Goodman. The power of natural semijoins. *SIAM Journal on Computing*, 10(4), pp. 751–771, 1981.

[5] H.L. Bodlaender and F.V. Fomin. Tree decompositions with small cost. In *In Proc. of SWAT'02*, Springer-Verlag LNCS 2368, pp. 378–387, 2002.

[6] J. Cai, V.T. Chakaravarthy, R. Kaushik, and J.F. Naughton. On the Complexity of Join Predicates. In *Proc. of PODS'01*, 2001.

[7] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the ACM*, 26:114–133, 1981.

[8] A.K. Chandra and P.M. Merlin. Optimal Implementation of Conjunctive Queries in relational Databases. In *Proc. of the STOC'77*, pp.77–90, 1977.

[9] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2), pp. 211–229, 2000.

[10] R. Dechter. Constraint networks. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pp. 276–285. Wiley, 1992. Volume 1, second edition.

[11] R. Dechter. Constraint Processing. Morgan Kaufmann, 2003.

[12] R. Fagin, A.O. Mendelzon, and J.D. Ullman. A simplified universal relation assumption and its properties. *ACM Transactions on Database Systems*, 7(3), pp. 343–360, 1982.

[13] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *Journal of the ACM*, 49(6), pp. 716–752, 2002.

[14] E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of ACM*, 32(4), pp. 755–761, 1985.

[15] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *Journal of the ACM*, 48(6), pp. 1184–1206, 2001.

[16] H. Garcia-Molina, J. Ullman, and J. Widom. *Database system implementation*. Prentice Hall, 2000.

[17] P. Harvey and A. Ghose. Reducing Redundancy in the Hypertree Decomposition Scheme, In *Proc. of 5th IEEE International Conference on Tools with Artificial Intelligence*, pp. 474–481, 2003.

[18] Y.E. Ioannidis. Query Optimization. *The Computer Science and Engineering Handbook*, pp. 1038–1057, 1997.

[19] Y.E. Ioannidis. The History of Histograms (abridged). In *In Proc. of VLDB'03, Berlin, Germany*, pp. 19-30, 2003.

[20] G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2), pp. 243–282, 2000.

[21] N. Goodman and O. Shmueli. Tree queries: a simple class of relational queries. *ACM Transactions on Database Systems*, 7(4), pp. 653–6773, 1982.

[22] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3), pp. 579–627, 2002.

[23] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124(2), pp.243–282, 2000.

[24] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3), pp. 431–498, 2001.

[25] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions: A survey. In *In Proc. of MFCS'2001*, pp. 37–57, 2001.

[26] G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL Certificates. *Theoretical Computer Science*, 270(1-2), pp. 761–777, 2002.

[27] S.H. Greibach. The Hardest Context-Free Language. *SIAM Journal on Computing*, 2(4):304–310, 1973.

[28] M. Grohe. The Complexity of Homomorphism and Constraint Satisfaction Problems Seen from the Other Side. In *Proc. of FOCS'03, Cambridge, MA, United States*, pp. 552–561, 2003.

[29] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *Proc. of STOC'01, Heraklion, Crete, Greece*, pp. 657–666, 2001.

[30] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Journal of Algorithms*, 66, pp. 57–89, 1994.

[31] D.S. Johnson, A Catalog of Complexity Classes, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pp. 67-161, 1990.

[32] P. G. Kolaitis and M. Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2), pp. 302–332, 2000.

[33] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1986.

[34] C.H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proc. of PODS'97, Tucson, Arizona*, pp. 12–19, 1997.

[35] J. Pearson and P.G. Jeavons. A Survey of Tractable Constraint Satisfaction Problems, CSD-TR-97-15, Royal Holloway, Univ. of London, 1997.

[36] N. Robertson and P.D. Seymour. Graph minors ii. algorithmic aspects of tree width. *Journal of Algorithms*, 7, pp. 309–322, 1986.

[37] W.L. Ruzzo. Tree-size bounded alternation. *Journal of Cumputer and System Sciences*, 21, pp. 218-235, 1980.

[38] D. Saccà. Closures of database hypergraphs. *Journal of the ACM*, 32(4), pp. 774–803, 1985.

[39] S. Skyum and L.G. Valiant. A complexity theory based on Boolean algebra. *Journal of the ACM*, 32:484–502, 1985.

[40] I.H. Sudborough. Time and Tape Bounded Auxiliary Pushdown Automata. In *Mathematical Foundations of Computer Science (MFCS'77)*, LNCS 53, Springer-Verlag, pp.493–503, 1977.

[41] Francesco Scarcello and Alfredo Mazzitelli. The hypertree decompositions homepage, since 2002. `http://wwwinfo.deis.unical.it/~frank/Hypertrees/`

[42] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.

[43] M. Vardi. Complexity of relational query languages. In *Proc. of STOC'82, San Francisco, California, United States*, pp. 137–146, 1982.

[44] M. Vardi. Constraint Satisfaction and Database Theory. *Tutorial at the 19th ACM Symposium on Principles of Database Systems (PODS'00)*. Currently available at: `http://www.cs.rice.edu/~vardi/papers/pods00t.ps.gz`.

[45] M. Yannakakis. Algorithms for acyclic database schemes. In *Proc. of VLDB'81, Cannes, France*, pp. 82–94, 1981.

[46] C.T. Yu and M.Z. Özsoyoğlu. On determining tree-query membership of a distributed query. *Infor*, 22(3), pp. 261–282, 1984.

[47] C.T. Yu, M.Z. Özsoyoğlu, and k. Lam. Optimization of Distributed Tree Queries. *Journal of Computer and System Sciences*, 29(3), pp. 409-445, 1984.

Appendix A: Query Decompositions

The following definition of query decomposition is a slight modification of the original definition given by Chekuri and Rajaraman [9]. Our definition is a bit more liberal because, for any conjunctive query Q, we do not care about the atom $head(Q)$, as well as of the constants possibly occurring in Q.

Definition 7.1 A *query decomposition* of a conjunctive query Q is a pair $\langle T, \lambda \rangle$, where $T = (N, E)$ is a tree, and λ is a labeling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq (atoms(Q) \cup var(Q))$, such that the following conditions are satisfied:

1. for each atom A of Q, there exists $p \in N$ such that $A \in \lambda(p)$;
2. for each atom A of Q, the set $\{p \in N \mid A \in \lambda(p)\}$ induces a (connected) subtree of T;
3. for each $Y \in var(Q)$, the set $\{p \in N \mid Y \in \lambda(p)\} \cup \{p \in N \mid Y$ occurs in some atom $A \in \lambda(p)\}$ induces a (connected) subtree of T.

The *width* of the query decomposition $\langle T, \lambda \rangle$ is $max_{p \in N} |\lambda(p)|$. The *query-width* $qw(Q)$ of Q is the minimum width over all its query decompositions. A query decomposition for Q is *pure* if, for each vertex $p \in N$, $\lambda(p) \subseteq atoms(Q)$. □

Then, *k-bounded-width queries* are queries whose query-width is bounded by a fixed constant $k > 0$. The notion of bounded query-width generalizes the notion of acyclicity [9]. Indeed, acyclic queries are exactly the conjunctive queries of query-width 1, because any join tree is a query decomposition of width 1.

It has been shown in [22] that deciding whether a conjunctive query has a bounded-width query decomposition is NP-complete.

Appendix B: LOGCFL– A Class of Parallelizable Problems

We define and characterize the class LOGCFL which consists of all decision problems that are logspace reducible to a context-free language. An obvious example of a problem complete for LOGCFL is Greibach's hardest context-free language [27]. There are a number of very interesting natural problems known to be LOGCFL-complete (see, e.g. [24, 40, 39]). The relationship between LOGCFL and other well-known complexity classes is summarized in the following chain of inclusions:

$$\mathrm{AC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{SL} \subseteq \mathrm{NL} \subseteq \mathrm{LOGCFL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \mathrm{P}$$

Here L denotes logspace, AC^i and NC^i are *logspace-uniform* classes based on the corresponding types of Boolean circuits, SL denotes symmetric logspace, NL denotes nondeterministic logspace, and P is polynomial time. For the definitions of all these classes, and for references concerning their mutual relationships, see [31].

Since LOGCFL $\subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2$, the problems in LOGCFL are all highly parallelizable. In fact, they are solvable in logarithmic time by a concurrent-read-concurrent-write (CRCW) parallel random-access-machine (PRAM) with a polynomial number of processors, or in $\log^2$-time by an exclusive-read-exclusive-write (EREW) PRAM with a polynomial number of processors.

In this paper, we use an important characterization of LOGCFL by Alternating Turing Machines. We assume that the reader is familiar with the *alternating Turing machine (ATM)* computational model introduced by Chandra et al. [7]. Here we assume without loss of generality that the states of an ATM are partitioned into existential and universal states.

As in [37], we define a *computation tree* of an ATM M on an input string w as a tree whose nodes are labeled with configurations of M on w, such that the descendants of any non-leaf labeled by a universal (existential) configuration include all (resp. one) of the successors of that configuration. A computation tree is *accepting* if the root is labeled with the initial configuration, and all the leaves are accepting configurations.

Thus, an accepting tree yields a certificate that the input is accepted. A complexity measure considered by Ruzzo [37] for the alternating Turing machine is the tree-size, i.e. the minimal size of an accepting computation tree.

Definition 7.2 ([37]) A decision problem P is solved by an alternating Turing machine M within *simultaneous* tree-size and space bounds $Z(n)$ and $S(n)$ if, for every "yes" instance w of P, there is at least one accepting computation tree for M on w of size (number of nodes) $\leq Z(n)$, each node of which represents a configuration using space $\leq S(n)$, where n is the size of w. (Further, for any "no" instance w of P there is no accepting computation tree for M.) □

Ruzzo [37] proved the following important characterization of LOGCFL :

Proposition 7.3 ([37]) *LOGCFL coincides with the class of all decision problems recognized by ATMs operating simultaneously in tree-size $O(n^{O(1)})$ and space $O(\log n)$.*