Hypertree Decompositions: A Survey

Georg Gottlob[1], Nicola Leone[2], and Francesco Scarcello[3]

[1] Information Systems Institute, TU-Wien
Vienna, Austria
gottlob@acm.org
[2] Dept. of Mathematics, Univ. of Calabria
Rende (CS), Italy
leone@unical.it
[3] D.E.I.S., Univ. of Calabria
Rende (CS), Italy
scarcello@acm.org

Abstract. This paper surveys recent results related to the concept of hypertree decomposition and the associated notion of hypertree width. A hypertree decomposition of a hypergraph (similar to a tree decomposition of a graph) is a suitable clustering of its hyperedges yielding a tree or a forest. Important NP hard problems become tractable if restricted to instances whose associated hypergraphs are of bounded hypertree width. We also review a number of complexity results on problems whose structure is described by acyclic or nearly acyclic hypergraphs.

1 Introduction

One way of coping with an NP hard problem is to identify significantly large classes of instances that are both recognizable and solvable in polynomial time. Such instances are often defined via some structural property of a graph $G(I)$ that is associated in a canonical way with the instance I. For example, many problems that are NP complete in general become tractable for instances I whose associated graph has bounded treewidth (cf. Sect. 4). Treewidth is a measure of the degree of cyclicity of a graph. Note that instances of bounded treewidth are also easy to recognize given that deciding whether the treewidth of a graph is at most k is decidable in linear time for each constant k.

The structure of a large number of problems is, however, more faithfully described by a *hypergraph* than by a graph. Again, several NP complete problems become tractable if restricted to instances with acyclic hypergraphs. In order to obtain larger tractable instance-classes of hypergraph-based problems, we thus investigated measures of hypergraph cyclicity that play a similar role for hypergraphs as the concept of treewidth does for graphs. In particular, an appropriate notion of hypergraph width (and an associated method of hypergraph decomposition) should fulfil both of the following conditions:

1. Relevant hypergraph-based problems should be solvable in polynomial time for instances of bounded width.

2. For each constant k, one should be able to check in polynomial time whether a hypergraph is of width k, and, in the positive case, it should be possible to produce an associated decomposition of width k of the given hypergraph.

Existing measures for hypergraph cyclicity we were aware of do either not fulfil one of these two conditions (e.g. recognizing hypergraphs of bounded *query width* is NP complete, cf. Section 4.3), or are not general enough (such methods are mentioned in Section 10). In particular, various notions of hypergraph width can be obtained by first transforming a hypergraph into a graph (there are several ways of doing so, see Section 4.2) and then considering the treewidth of that graph. However, it is not hard to see that such measures of cyclicity are not very significant due to a loss of structural information caused by the transformation of the hypergraph to a graph (cf. Sect. 4.2). In summary, it appeared that a satisfactory way of determining the degree of cyclicity of a hypergraph was missing, on the basis of which large tractable instances of relevant NP-hard problems could be defined.

Consequently, after a careful analysis of the shortcomings of various hypergraph decomposition methods, we introduced the new method of *hypertree decomposition* and the associated notion of *hypertree width.* To our best knowledge, the method of hypertree decomposition is currently the most general known hypergraph decomposing method leading to large tractable classes of important problems such as *constraint satisfaction problems* or *conjunctive queries.* The notion of hypertree decomposition and the associated notion of hypertree width are the main topics of the present survey paper. However, we will also report on a number of other closely related issues, such as the precise (parallel) complexity of acyclic database queries, and the notion of *query decomposition.* Our results surveyed here are mainly from the following sources, where formal proofs, details, and a number of further results can be found:

– Reference [17], where the precise complexity of acyclic Boolean conjunctive queries (ABCQs) is determined, and where highly parallel database algorithms for solving such queries are presented. (In the present paper, we will not discuss parallel database algorithms and refer the interested reader to [17] and [22]).
– Reference [19], where we first study *query width*, a measure for the amount of cyclicity of a query introduced by Chekuri and Rajamaran [6], and where we define and study the new (more general) concept of *hypertree width.*
– Reference [21], where we establish criteria for comparing different CSP decomposition methods and where we compare various methods including the method of hypertree decomposition. The comparison criteria and the results of the comparison are reported in Section 10 of the present paper.
– Reference [24], where hypertree width is compared to Courcelle's notion of *clique width* [7,8].
– Reference [23], where we give a game theoretic and a logical characterization of hypertree width. These results are reported in Sections 8 and 9 of the present paper, respectively.

This paper is organized as follows. In Section 2 we define a number of important hypergraph-based problems. In Section 3 we discuss the complexity of acyclic instances of these problems. In Section 4, discuss graph treewidth and a generalization termed *query width*. In Section 5 we define the concepts of hypertree decomposition and hypertree width. In Section 6, we show how hypergraphs of bounded hypertree width can be recognized in polynomial time. In Section 7, we show how CSPs can be solved in polynomial time for instances of bounded hypertree-width. In Section 8, we describe the *Robber and Marshals* game which characterizes hypergraphs of bounded hypertree width. In Section 9, we briefly describe our logical characterization of queries of bounded hypertree-width. In Section 10, we give a brief account on how the notion of hypertree decomposition compares to other related notions. Finally, in Section 11, we state some relevant open problems.

2 Hypergraph-Based Problems

A *relational vocabulary* (short: *vocabulary*) consists of a finite nonempty set of relation symbols $P, Q, R \ldots$, with associated arities. A *finite relational structure* (short: *finite structure*) C over a vocabulary τ consists of a finite *universe* U_C and for each k-ary relation symbol R in τ a relation $R^C \subseteq U_C^k$. For a tuple $(c_1, \ldots, c_k) \in R^C$ we often write $R^C(c_1, \ldots, c_k)$. We denote the vocabulary of a structure C by $vo(C)$.

Let A and B be two finite structures such that $vo(A) \subseteq vo(B)$. Then a mapping $h : A \longrightarrow B$ is a *homomorphism* from A to B if for each relation symbol $R \in vo(A)$ it holds that whenever $(c_1, \ldots, c_k) \in R^A$ for some elements $c_1, \ldots, c_k \in U_A$, then it also holds that $(h(c_1), \ldots, h(c_k)) \in R^B$. The following is a fundamental computational problem in Algebra:

Definition 1 (The Homomorphism Problem HOM). *Given two finite structures A and B, decide whether there exists a homomorphism from A to B. We denote such an instance of* HOM *by* HOM(A, B)

It is well-known (cf. [12]) that HOM is an NP-complete problem. For example, checking whether a graph (V, E) is three colorable amounts to solve the $HOM(A, B)$ problem for structures A and B over a vocabulary with a unique binary relation symbol R, where $R^A = E$ and $R^B = \{(red, blue), (blue, red), (red, green), (green, red), (blue, green), (green, blue)\}$.

In [12,30] it was observed that HOM is equivalent to (and actually, in essence, the same as) the important *constraint satisfaction problem* (CSP) of Artificial Intelligence [9], which, in turn, is equivalent to the database problem BCQ of evaluating Boolean conjunctive queries:

Definition 2 (The Constraint Satisfaction Problem CSP.). *Given a finite set* Var *of variables, a finite domain U of values, a set of constraints $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$, where each constraint C_i is a pair (S_i, r_i), and where S_i is* *a list of variables of length m_i, called the constraint scope, and r_i is an m_i-ary relation over U, called a constraint relation, decide whether there is a substitution $\vartheta : \mathrm{Var} \longrightarrow U$, such that, for each $1 \leq i \leq q$, $S_i\vartheta \in r_i$.*

Definition 3 (The Boolean Conjunctive Query Problem BCQ.). *A relational database is formalized as a finite relational structure D. A Boolean conjunctive query (BCQ) on D is a sentence of first-order logic of the form: $\exists X_1, \ldots, X_r\ R_1(t_1^1, t_2^1 \ldots, t_{\alpha(1)}^1) \ \wedge \ \ldots \ \wedge\ R_k(t_1^k, t_2^k \ldots, t_{\alpha(k)}^k)$, where, for $1 \leq i \leq k$, R_i is a relation symbol from $vo(D)$ of associated arity $\alpha(i)$, and for $1 \leq i \leq k$ and $1 \leq j \ \leq \alpha(i)$, each t_j^i is a term, i.e., either a variable from the list $X_1, \ldots, X_r$, or a constant element from U_D. The decision problem* BCQ *is the problem of deciding for a pair $\langle D, Q\rangle$, where D is a database and Q is a Boolean conjunctive query, whether Q evaluates to true over D, denoted by $D \models Q$.*

Given that all variables occuring in a BCQ are existentialy quantified, we usually omit the quantifier prefix and write a BCQ as a conjunction of *query atoms*. For example, let `emp` denote a relation containing (employee#,project#) pairs, and let `rel` be a relation containing a pair (n_1, n_2) if n_1 and n_2 are numbers of distinct employees who are relatives, then the BCQ $\texttt{emp}(X, Z) \wedge \texttt{emp}(Y, Z) \wedge \texttt{rel}(X, Y)$ expresses that there are two employees who are relatives and work on the same project.

Note that by simple logspace operations (selections and projections on the corresponding relations) one can always eliminate constants occurring in BCQ atoms. We thus assume w.l.o.g. that query atoms contain only variables as arguments. By this assumption, CSP and BCQ are exactly the same problem, where each constraint scope corresponds to a query atom, each constraint relation corresponds to a database relation, and vice-versa. Now each CSP (or BCQ) instance I can in turn be identified with HOM(A, B), where A is the structure whose universe U_A consists of the set Var of all variables of I and whose relations contain constraint scopes (or query atoms) as tuples, and where B is the structure whose universe U_B is the finite domain U of I and whose relations are just the constraint relations (or the database relations). In this sense, we can speak about instances CSP(A, B) and BCQ(A, B), where A is a structure representing the constraint scopes or the query, and B denotes the set of constraint relations, or the database, respectively. On the other hand, each instance $I = \mathrm{HOM}(A, B)$ of HOM can be identified in the obvious way with a CSP instance (or a BCQ instance) by interpreting the elements of U_A as variables and those of U_B as domain elements of the constraint (or database) relations.

Thus all three problems HOM, CSP, and BCQ are the same and are NP-complete (for BCQ this was first shown in [5]). Therefore, it is important to find large classes of instances that can be evaluated in polynomial time. Such classes can be defined by imposing structural restrictions on the problem instances. In particular, for HOM(A, B), CSP(A, B), or, equivalently, BCQ(A, B), one could impose restrictions on the structure A, or on the structure B, or on both (cf. [35,33,30]). In this paper we are interested in restrictions on the structure A. In database terms, we can recast this by saying that we are interested in the structure of the query, rather than on the properties of the database content.

If $\mathcal{A}$ denotes a set of structures, then HOM($\mathcal{A}$), CSP($\mathcal{A}$), and BCQ($\mathcal{A}$) denote the restrictions of HOM, CSP, and BCQ to instances HOM(A, B), CSP(A, B), and BCQ(A, B) respectively, where $A \in \mathcal{A}$.

Each finite structure C over universe U_C defines a hypergraph $\mathcal{H}(C) = (V, H)$ as follows: The set V of vertices V of $\mathcal{H}(C)$ coincides with U_C; the set of hyperedges H of $\mathcal{H}(C)$ consists of all sets $\{c_1, \ldots, c_k\}$ such that there exists a relation R in $voc(C)$ and $(c_1, \ldots, c_k) \in R^C$.

To each problem instance I = HOM(A, B) or I = CSP(A, B), or I = BCQ(A, B), we define the associated hypergraph $\mathcal{H}_I$ by $\mathcal{H}_I = \mathcal{H}(A)$. In particular, this means, that for an instance I of CSP, $\mathcal{H}_I$ denotes the hypergraph whose vertices are the variables of I and whose hyperedges are all sets $\{X_1, \ldots, X_k\}$ such that there exists a constraint scope $S = (X_1, \ldots, X_k)$ belonging to I.
For an instance $I = (D, Q)$ of BCQ, $\mathcal{H}_I$ denotes the hypergraph whose vertices are all the variables occurring in Q and whose hyperedges are the sets $var(\alpha)$ of variables occuring in α, for each query atom α.

Example 4. Figure 1(a) shows $\mathcal{H}_{I_1}$ for an instance I_1 of BCQ having query Q_1 : $a(S, X, T, R) \wedge b(S, Y, U, P) \wedge f(R, P, V) \wedge g(X, Y) \wedge c(T, U, Z) \wedge \ d(W, X, Z) \wedge e(Y, Z)$

It is furthermore easy to see that HOM, BCQ, and CSP are all equivalent (via logspace transformations) to the following fundamental problems in database theory and artificial intelligence [17]: The *Query Output Tuple Problem:* Given a conjunctive query Q, a database **db**, and a tuple t, determine whether t belongs to the answer Q(**db**) of Q over **db**. The *Conjunctive Query Containment:* Decide whether a conjunctive query Q_1 is contained in a conjunctive query Q_2. Query Q_1 is contained in query Q_2 if, for each database instance **db**, the answer Q_1(**db**) is a subset of Q_2(**db**). The *Clause Subsumption Problem*: Check whether a (general) clause C subsumes a clause D, i.e., whether there exists a substitution ϑ such that $C\vartheta \subseteq D$. A (general) clause is a disjunction of (positive or negative) literals,

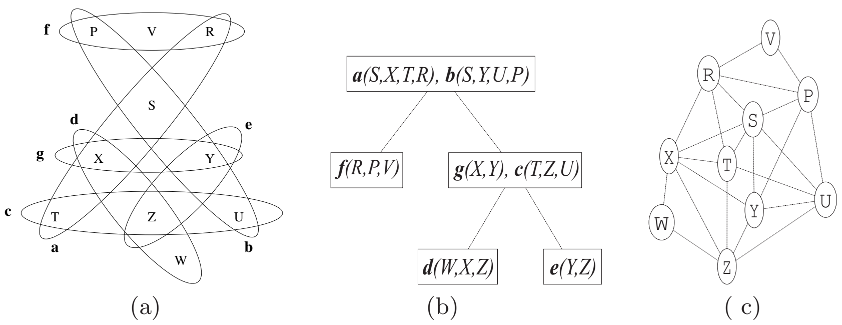


Fig. 1. (a) Hypergraph $\mathcal{H}_{I_1}$; (b) a width 2 hypertree decomposition of $\mathcal{H}_{I_1}$; and (c) the primal graph of $\mathcal{H}_{I_1}$

possibly containing function symbols. Note that subsumption is an extremely important technique used in clause-based theorem proving [1].

Just for the sake of presentation, we will focus in the rest of this paper on the constraint statisfaction problem (CSP).

While, as we will see, many interesting structural properties of a CSP instance $I = \mathrm{CSP}(A, B)$ can be identified by looking at the associated hypergraph $\mathcal{H}_I = \mathcal{H}(A)$, which in AI is called the *constraint hypergraph*, some structural properties of I may be also detected using its *primal graph*, i.e., the primal graph of the hypergraph associated to A, which coincides with the Gaifman graph of A [15]. Let $\mathcal{H}_I = (V, H)$ be the constraint hypergraph of a CSP instance I. The *primal graph* of I is a graph $G = (V, E)$, having the same set of variables (vertices) as $\mathcal{H}_I$ and an edge connecting any pair of variables $X, Y \in V$ such that $\{X, Y\} \subseteq h$ for some $h \in H$. Note that, if the vocabulary of A contains only binary predicates, then all constraints of I are binary and its associated hypergraph is identical to its primal graph. The primal graph of the hypergraph of query Q_1 (and of the equivalent CSP instance) is depicted in Fig. 1(c).

Since in this paper we always deal with hypergraphs corresponding to CSP or BCQ instances, the vertices of any hypergraph $\mathcal{H} = (V, H)$ can be viewed as the variables of some constraint satisfaction problem or of some conjunctive query. Thus, we will often use the term *variable* as a synonym for vertex, when referring to elements of V. For the hypergraph $\mathcal{H} = (V, H)$, $var(\mathcal{H})$ and $edges(\mathcal{H})$ denote the sets V and H, respectively. When illustrating a decomposition, we will usually represent hyperedges of the hypergraph $\mathcal{H}_I$ of a BCQ or CSP instance I by their corresponding query atoms or constraint scopes.

3 Acyclic Instances

The most basic and most fundamental structural property considered in the context of CSPs and conjunctive queries is *acyclicity*. It was recognized in AI and database theory that *acyclic* CSPs or conjunctive queries are polynomially solvable. A CSP instance I is *acyclic* if its associated hypergraph $\mathcal{H}_I$ is acyclic.

A hypergraph $\mathcal{H}$ is acyclic if and only if its primal graph G is chordal (i.e., any cycle of length greater than 3 has a chord) and the set of its maximal cliques coincide with $edges(\mathcal{H})$ [2].

A *join tree* $\mathcal{JT}(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose nodes are the edges of $\mathcal{H}$ such that whenever the same vertex $X \in V$ occurs in two edges A_1 and A_2 of $\mathcal{H}$, then A_1 and A_2 are connected in $\mathcal{JT}(\mathcal{H})$, and X occurs in each node on the unique path linking A_1 and A_2 in $\mathcal{JT}(\mathcal{H})$. In other words, the set of nodes in which X occurs induces a (connected) subtree of $\mathcal{JT}(\mathcal{H})$. We will refer to this condition as the *Connectedness Condition* of join trees.

Acyclic hypergraphs can be characterized in terms of join trees: A hypergraph $\mathcal{H}$ is *acyclic* iff it has a join tree [3,2,32].

Note that acyclicity as defined here is the usual concept of acyclicity in the context of database theory and AI. It is referred to as α-acyclicity in [11]. This is the least restrictive concept of hypergraph acyclicity among all those defined in the literature.

Acyclic CSPs and conjunctive queries have highly desirable computational properties:

1. Acyclic instances can be efficiently solved. Yannakakis provided a (sequential) polynomial time algorithm solving BCQ on acyclic queries[1] [41].
2. Acyclicity is efficiently recognizable, and a join tree of an acyclic hypergraph is efficiently computable. A linear-time algorithm for computing a join tree is shown in [37]; an L^{SL} method has been provided in [17] (L^{SL} denotes logspace relativized by an oracle in symmetric logspace; this class could also be termed "functional SL").
3. The result of a (non-Boolean) acyclic conjunctive query Q can be *computed* in time polynomial in the combined size of the input instance and of the output relation [41].
4. *Arc-consistency* for acyclic CSP instances can be enforced in polynomial time [9,10].

Intuitively, the efficient behavior of acyclic instances is due to the fact that they can be evaluated by processing any of their join trees bottom-up by performing upward semijoins, thus keeping small the size of the intermediate relations (which could become exponential if regular join were performed).

We have recently determined the precise computational complexity of BCQ, and hence of HOM, CSP, and all their equivalent problems. It turned out all these problems are highly parallelizable on acyclic structures, as they are complete for the low complexity class LOGCFL [17]. This is the class of all decision problems that are logspace-reducible to a context-free language. Note that $\mathrm{NL} \subseteq \mathrm{LOGCFL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \mathrm{P}$ where NL denotes nondeterministic logspace and AC^1 and NC^2 are logspace-uniform classes based on the corresponding types of Boolean circuits (for precise definitions of all these complexity classes, cf. [29]). Let $\mathcal{AH}$ be the set of all finite acyclic relational structures.

Theorem 5 ([17]). CSP($\mathcal{AH}$) *is* LOGCFL-*complete.*

Moreover, the functional version of these problems belongs to the functional version of LOGCFL, i.e., a solution for a CSP instance can be computed in L^{LOGCFL}, i.e., functional logspace with an oracle in LOGCFL. Efficient parallel algorithms – even for non-Boolean queries – have been proposed in [22]. They run on parallel database machines that exploit the *inter-operation parallelism* [40], i.e., machines that execute different relational operations in parallel.

The important speed-up obtainable on acyclic instances stimulated several research efforts towards the identification of wider classes of queries and constraints having the same desirable properties as acyclic CQs and CSPs.

[1] Note that, since both the database **db** and the query Q are part of an input-instance of BCQ, what we are considering is the *combined complexity* of the query [38].

4 Treewidth, Query Width, and Hypertree Width

4.1 Tree Decompositions and Treewidth of Graphs

The treewidth of a graph is a well-known measure of its tree-likeness introduced by Robertson and Seymour in their work on graph minors [34]. This notion plays a central role in algorithmic graph theory as well as in many subdisciplines of Computer Science.

Definition 6. A *tree decomposition* of a graph $G = (V, E)$ is a pair $\langle T, \chi \rangle$, where $T = (N, F)$ is a tree, and χ is a labeling function associating to each vertex $p \in N$ a set of vertices $\chi(p) \subseteq V$, such that the following conditions are satisfied: (1) for each vertex b of G, there exists $p \in N$ such that $b \in \chi(p)$; (2) for each edge $\{b, d\} \in E$, there exists $p \in N$ such that $\{b, d\} \subseteq \chi(p)$; (3) for each vertex b of G, the set $\{p \in N \mid b \in \chi(p)\}$ induces a (connected) subtree of T.

The *width* of the tree decomposition $\langle T, \chi \rangle$ is $\max_{p \in N} |\chi(p) - 1|$. The *treewidth* of G is the minimum width over all its tree decompositions. The treewidth of a CSP instance is the treewidth of its associated primal graph.

The notion of treewidth is a generalization of graph acyclicity. In particular, a graph is acyclic if and only if its treewidth is one [34].

Checking whether a graph has treewidth at most k for a fixed constant k, and in the positive case computing a k-width tree decomposition, is feasible in linear time [4]. Moreover, this task is also parallelizable. Indeed, Wanke [39] has shown that, for a fixed constant k, checking whether a graph has treewidth k is in LOGCFL. By proving some general complexity-theoretic results and by using Wanke's result, the following was shown in [18]:

Theorem 7 ([18]). *For each constant k, there exists an $\mathrm{L}^{\mathrm{LOGCFL}}$ transducer T_k that behaves as follows on input G. If G is a graph of treewidth $\leq k$, then T_k outputs a tree decomposition of width $\leq k$ of G. Otherwise, T_k halts with empty output.*

Thus, a tree decomposition of width at most k can be also computed in (the functional version of) LOGCFL, and thus by logspace uniform AC^2 and NC^2 circuits.

An important feature of treewidth is that many NP-complete problems are decidable in polynomial-time on structures having bounded treewidth, i.e., having treewidth at most k for some fixed constant $k > 0$. In particular, Courcelle proved that every property expressible in monadic second order logic is decidable in linear time over bounded treewidth graphs [7].

4.2 Treewidth of Hypergraphs

As mentioned in the previous section, many NP-complete problems become tractable on bounded treewidth graphs. In order to exploit this nice feature for CSP, BCQ, and their equivalent problems, many researchers in the AI and the database communities considered the primal graph (of the hypergraph) associated to the relational structure. Let TW$[k]$ be the set of all finite relational structures whose associated primal graph has treewidth at most k. It has been shown that CSP(TW$[k]$) is solvable in polynomial time [14] and has the same properties of CSP($\mathcal{AH}$), including its precise computational complexity.

Theorem 8 ([17]). CSP(TW$[k]$) *is* LOGCFL-*complete.*

Note that considering the primal graph associated to a hypergraph is not the one possible choice. Given a CSP instance I, the *dual graph* [9,10,32] of the hypergraph $\mathcal{H}_I$ is a graph $G_I^d = (V, E)$ defined as follows: the set of vertices V coincides with the set of (hyper)edges of $\mathcal{H}_I$, and the set E contains an edge $\{h, h'\}$ for each pair of vertices $h, h' \in V$ such that $h \cap h' \neq \emptyset$. That is, there is an edge between any pair of vertices corresponding to hyperedges of $\mathcal{H}_I$ sharing some variable.

The dual graph often looks very intricate even for simple CSPs. For instance, in general, acyclic CSPs do not have acyclic dual graphs. However, it is well known that the dual graph G_I^d can be suitably simplified in order to obtain a "better" graph G' which can still be used to solve the given CSP instance I. In particular, if I is an acyclic CSP, G_I^d can be reduced to an acyclic graph that represents a join tree of $\mathcal{H}_I$. In this case, the reduction is feasible in polynomial (actually, linear) time. (See, e.g., [32].) However, in general, it is not known whether there exists an efficient algorithm for obtaining the best simplified graph G' with respect to the treewidth notion, i.e., the simplification of G_I^d having the smallest treewidth over all its possible simplifications (see [30] for a formal statement of this open problem and [21] for a comparison of this notion with some hypergraph-based notions).

Another possibility is considering the so called *incidence graph* [6]. Given a CSP instance I, the incidence graph $G_I^i(\mathcal{H}_I) = (V', E)$ associated to the hypergraph $\mathcal{H}_I = (V, H)$ has a vertex for each variable and for each hyperedge of $\mathcal{H}_I$. There is an edge $\{x, h\} \in E$ between a variable $x \in V$ and and hyperedge $h \in H$ whenever x occurs in h.

The class of all CSP instances whose dual graphs (resp. incidence graphs) have bounded treewidth are solvable in polynomial time and, actually, they are LOGCFL-complete. However, note that none of these classes of CSP instances generalize the class CSP($\mathcal{AH}$). Indeed, there are families of acyclic hypergraphs whose associated primal graphs, dual graphs (without considering simplification), and incidence graphs have unbounded treewidth.

Note that by results of [25] bounded treewidth is most likely the best and most general structural restriction for obtaining tractable CSP and BCQ instances, when the structure of a CSP or BCQ is described via a *graph* (e.g. the primal graph), rather than by a hypergraph. Further interesting material on BCQ and treewidth can be found in [13].

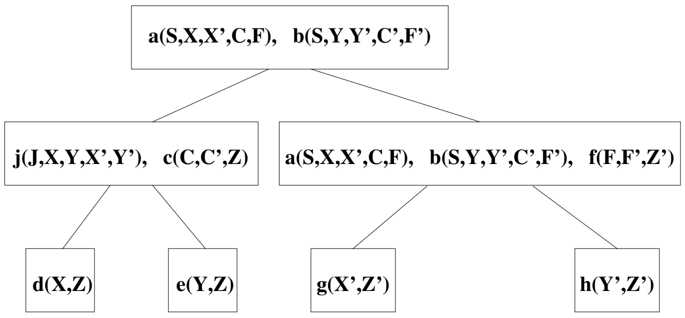
4.3 Query Decompositions and Query Width

A more general notion that generalizes hypergraph acyclicity is *query width* [6]. The notion of bounded query-width is based on the concept of *query decomposition* [6]. We next adapt this notion to the more general setting of hypergraphs, while it was originally defined in terms of queries. Roughly, a query decomposition of a hypergraph $\mathcal{H}$ consists of a tree each vertex of which is labelled by a set of hyperedges and/or variables. Each variable and hyperedge induces a connected subtree (*connectedness condition*). Each hyperedge occurs in at least one label. The width of a query decomposition is the maximum of the cardinalities of its vertices. The *query-width* $qw(\mathcal{H})$ of $\mathcal{H}$ is the minimum width over all its query decompositions.

Example 9. Consider the CSP instance I_2 having the following constraint scopes:

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

The query-width of $\mathcal{H}_{I_2}$ is 3. Figure 2 shows a query decomposition of $\mathcal{H}_{I_2}$ of width 3. W.l.o.g. we represent hyperedges by the corresponding constraint scopes or query atoms in such decompositions.



a(S,X,X',C,F), b(S,Y,Y',C',F')
j(J,X,Y,X',Y'), c(C,C',Z)
a(S,X,X',C,F), b(S,Y,Y',C',F'), f(F,F',Z')
d(X,Z)
e(Y,Z)
g(X',Z')
h(Y',Z')

Fig. 2. A 3-width query decomposition of $\mathcal{H}_{I_2}$

Each hypergraph whose primal graph has treewidth at most k has query width at most k, too. The converse does not hold, in general. Moerover, this notion is a true generalization of the basic concept of acyclicity: A hypergraph is acyclic iff it has query width 1.

Let k be a fixed constant. Chekuri and Rajaraman [6] proved that, given a BCQ instance I and a query decomposition of $\mathcal{H}_I$ having width at most k, I is solvable in polynomial time. In [17] it was shown that this problem is LOGCFL-complete (and thus highly parallelizable).

However, when the notion of query-width was defined and studied in [6], no polynomial algorithm for checking whether a hypergraph has query-width at most k was known, and Chekuri and Rajaraman [6] stated this as an open problem. This problem is solved in [19], where it is shown that is unlikely to find an efficient algorithm for recognizing instances of bounded query-width.

Theorem 10 ([19]). *Determining whether the query-width of a hypergraph is at most 4 is* NP*-complete.*

Fortunately, it turned out that the high complexity of determining bounded query-width is not, as one would usually expect, the price for the generality of the concept. Rather, it is due to some peculiarity in its definition. In the next section, we present a new notion that does not suffer from such problems. Indeed, this notion generalizes query width (and hence acyclicity) and is tractable.

5 Hypertree Decompositions and Hypertree Width

A new class of tractable CSP instances, which generalizes the class CSP($\mathcal{AH}$) of CSP instances having an acyclic hypergraph, has recently been identified [19]. This is the class of CSPs whose hypergraph has a bounded-width hypertree decomposition [19].

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 labeling functions which associate to each vertex $p \in N$ 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 11. 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 $var(h) \subseteq \chi(p)$ (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 $p \in vertices(T)$, $var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p)$.

Note that the inclusion in Condition 4 is actually an equality, because Condition 3 implies the reverse inclusion.

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 $h \in \lambda(p)$. We then say that p strongly covers h.

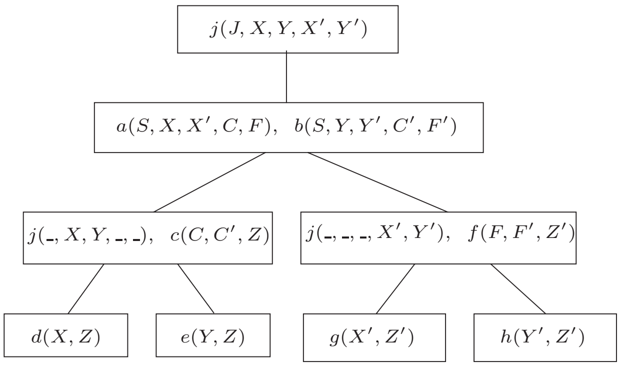
A hypertree decomposition *HD* of hypergraph $\mathcal{H}$ is a *complete decomposition* of $\mathcal{H}$ if every edge of $\mathcal{H}$ is strongly covered in *HD*.

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 $c = hw(\mathcal{H})$.

The acyclic hypergraphs are precisely those hypergraphs having hypertree width one. Indeed, any join tree of an acyclic hypergraph $\mathcal{H}$ trivially corresponds to a hypertree decomposition of $\mathcal{H}$ of width one. Furthermore, if a hypergraph $\mathcal{H}'$ has a hypertree decomposition of width one, then, from this decomposition, we can easily compute a join tree of $\mathcal{H}'$, which is therefore acyclic [19].

It is worthwhile noting that from any hypertree decomposition HD of $\mathcal{H}$, we can easily compute a complete hypertree decomposition of $\mathcal{H}$ having the same width in $O(\|\mathcal{H}\| \cdot \|HD\|)$ time.

Intuitively, if $\mathcal{H}$ is a cyclic hypergraph, the χ labeling selects the set of variables to be fixed in order to split the cycles and achieve acyclicity; $\lambda(p)$ "covers" the variables of $\chi(p)$ by a set of edges.



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

Fig. 3. A Hypertree decomposition of $\mathcal{H}_2$

Example 12. Figure 3 shows an hypertree decomposition HD_2 of the cyclic hypergraph $\mathcal{H}_2$ associated to the CSP instance in Example 9. Each node p in the tree is labeled by a set of hyperedges representing $\lambda(p)$; $\chi(p)$ is the set of all variables, distinct from '_', appearing in these hyperedges. Thus, the anonymous variable '_' replaces the variables in $var(\lambda(p)) - \chi(p)$.

Using this graphical representation, we can easily observe an important feature of hypertree decompositions. Once an hyperedge has been covered by some vertex of the decomposition tree, any subset of its variables can be used freely in order to decompose the remaining cycles in the hypergraph. For instance, the variables in the hyperedge corresponding to constraint j in $\mathcal{H}_2$ are jointly included only in the root of the decomposition. If we were forced to take all the variables in every vertex where j occurs, it would not be possible to find a decomposition of width 2. Indeed, in this case, any choice of two hyperedges per vertex yields a hypertree which violates the connectedness condition for variables (i.e., Condition 2 of Definition 11).

Figure 1(b) shows a complete hypertree decomposition of width 2 of the hypergraph $\mathcal{H}_{I_1}$ in part (a) of the figure. Note that this decomposition also happens to be a query decomposition of width 2.

Let k be a fixed positive integer. We say that a CSP instance I has k-bounded hypertree width if $hw(\mathcal{H}_I) \leq k$, where $\mathcal{H}_I$ is the hypergraph associated to I.

6 Computing Hypertree Decompositions

Let $\mathcal{H}$ be a hypergraph, and let $V \subseteq var(\mathcal{H})$ be a set of variables and $X, Y \in var(\mathcal{H})$. Then 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, for all $X, Y \in W$, 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 $\mathcal{H}$. 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. We introduce a normal form for hypertree decompositions.

```
ALTERNATING ALGORITHM  k-decomp
Input: A non-empty Hypergraph $\mathcal{H}$.
Result: "Accept", if $\mathcal{H}$ has $k$-bounded hypertree-width; "Reject", otherwise.

Procedure k-decomposable($C_R$: SetOfVariables, R: SetOfHyperedges)
begin
1)  Guess a set $S \subseteq edges(\mathcal{H})$ of $k$ elements at most;
2)  Check that all the following conditions hold:
    2.a) $\forall P \in edges(C_R)$, $(var(P) \cap var(R)) \subseteq var(S)$ and
    2.b) $var(S) \cap C_R \neq \emptyset$
3)  If the check above fails Then Halt and Reject; Else
    Let $\mathcal{C} := \{C \subseteq var(\mathcal{H}) \mid C$ is a $[var(S)]$-component and $C \subseteq C_R\}$;
4)  If, for each $C \in \mathcal{C}$, k-decomposable$(C, S)$
        Then Accept
        Else Reject
end;

begin(* MAIN *)
    Accept if k-decomposable$(var(\mathcal{H}), \emptyset)$
end.
```

Fig. 4. A non-deterministic algorithm deciding k-bounded hypertree-width

Definition 13 ([19]). A hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of a hypergraph $\mathcal{H}$ is in *normal form* (*NF*) if, for each vertex $r \in \mathit{vertices}(T)$, 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. $\mathit{var}(\lambda(s)) \cap \chi(r) \subseteq \chi(s)$.

Intuitively, each subtree rooted at a child node s of some node r of a normal form decomposition tree serves to decompose precisely one $[r]$-*component*.

Theorem 14 ([19]). *For each k-width hypertree decomposition of a hypergraph $\mathcal{H}$ there exists a k-width hypertree decomposition of $\mathcal{H}$ in normal form.*

This normal form theorem immediately entails that, for each optimal hypertree decomposition of a hypergraph $\mathcal{H}$, there exists an optimal hypertree decomposition of $\mathcal{H}$ in normal form.

Importantly, NF hypertree decompositions can be efficiently computed. Figure 4 shows the algorithm `k-decomp`, deciding whether a given hypergraph $\mathcal{H}$ has a k-bounded hypertree-width decomposition. k`-decomp` can be implemented on a logspace ATM having polynomially bounded tree-size, and therefore entails LOGCFL membership of deciding k-bounded hypertree-width.

Theorem 15 ([19]). *Deciding whether a hypergraph $\mathcal{H}$ has k-bounded hypertree-width is in* LOGCFL.

From an accepting computation of the algorithm of Figure 4 we can efficiently extract a NF hypertree decomposition. Since an accepting computation tree of a bounded-treesize logspace ATM can be *computed* in (the functional version of) LOGCFL [18], we obtain the following.

Theorem 16 ([19]). *Computing a k-bounded hypertree decomposition (if any) of a hypergraph $\mathcal{H}$ is in* $\mathrm{L}^{\mathrm{LOGCFL}}$*, i.e., in functional* LOGCFL*.*

As for sequential algorithms, a polynomial time algorithm `opt-`k`-decomp` which, for a fixed k, decides whether a hypergraph has k-bounded hypertree width and, in this case, computes an optimal hypertree decomposition in normal form is described in [20]. As for many other decomposition methods, the running time of this algorithm to find the hypergraph decomposition is exponential in the parameter k. More precisely, `opt-`k`-decomp` runs in $O(m^{2k}v^2)$ time, where m and v are the number of edges and the number of vertices of $\mathcal{H}$, respectively.

7 Solving CSP Instances of Bounded Hypertree Width

Figure 5 outlines an efficient method to solve CSP instances of bounded Hypertree Width. The key point is that any CSP instance I having k-bounded hypertree width can be efficiently transformed into an equivalent acyclic CSP instance (Step 4.), which is then evaluated by the well-known techniques defined for acyclic CSPs (see Section 3). Let HW$[k]$ be the set of all finite relational structures whose associated hypergraph has hypertree width at most k.

```
ALGORITHM
Input: A k-bounded hypertree width CSP instance $I$.
Result: A solution to $I$, if $I$ is satisfiable; "No", otherwise.

begin
1)  Build the hypergraph $\mathcal{H}_I$ of $I$.
2)  Compute a $k$-width hypertree decomposition $HD$ of $\mathcal{H}_I$ in normal form.
3)  Compute from $HD$ a complete hypertree decomposition $HD' = (T, \chi, \lambda)$ of $\mathcal{H}_I$.
4)  Compute from $HD'$ and $I$ an acyclic instance $I^*$ equivalent to $I$.
5)  Evaluate $I^*$ employing any efficient technique for solving acyclic CSPs.
6)  If $I^*$ is satisfiable, then return a solution to $I^*$;
        Else Return "No"
end.
```

Fig. 5. An algorithm solving CSP instances of k-bounded hypertree-width

Theorem 17 ([21]). *Given a CSP instance* $I \in \text{CSP}(\text{HW}[k])$ *and a k-width hypertree decomposition of* $\mathcal{H}_I$ *in normal form, I is solvable in* $O(\|I\|^{k+1} \log \|I\|)$ *time.*

We have also determined the precise computational complexity of solving CSP instances having bounded hypertree-width.

Theorem 18 ([19]). CSP(HW[k]) *is* LOGCFL-*complete.*

8 Game Theoretic Characterization of Hypertree Width

In [36], graphs G of treewidth k are characterized by the so called *Robber-and-Cops game* where $k+1$ cops have a winning strategy for capturing a robber on G. Cops can control vertices of a graph and can jump at each move to arbitrary vertices. The robber can move (at infinite speed) along paths of G but cannot go over vertices controlled by a cop. It is, moreover, shown that a winning strategy for the cops exists, iff the cops can capture the robber in a *monotonic* way, i.e., never returning to a vertex that a cop has previously vacated, which implies that the moving area of the robber is monotonically shrinking. For more detailed descriptions of the game, see [36] or [23].

In order to provide a similarily natural characterization for hypertree-width, we defined in [23] a new game, the *Robber and Marshals game (R&Ms game).* A marshal is more powerful than a cop. While a cop can control a single vertex (=variable) only, a marshal controls an entire hyperedge. In the *R&Ms* game, the robber moves on vertices just as in the robber and cops game, but now marshals instead of cops are chasing her. During a move of the marshals from the set of hyperedges E to to the set of hyperedges E', the robber cannot pass through the vertices in $B = (\cup E) \cap (\cup E')$, where, for a set of hyperedges F, $\cup F$ denotes the union of all hyperedges in F. Intuitively, the vertices in B are those not released by the marshals during the move. As in the monotonic robber and cops game, it is required that the marshals capture the robber by monotonically shrinking the moving space of the robber. The game is won by the marshals if they corner the robber somewhere in the hypergraph.

Example 19. Let us play the robber-and-marshals game on the hypergraph $\mathcal{H}_{I_1}$ of query Q_1 of Example 4 (see Fig 1). We can easily recognize that two marshals can always capture the robber and win the game by using the following strategy: Independently of the initial position of the robber, the two marshals initially move on edges $\{a,b\}$, and thus control the vertices (=variables) T, X, S, R, P, Y, U, as shown in Figure 6.A. After this move of marshals, the robber may be in V, in Z or in W. If the robber is on V, then the marshals move on edge f, and capture the robber, as shown in Figure 6.B (note that the robber cannot escape from V during this move, as both P and R – the only possible ways to leave V – are kept under the marshals' control during the move). Otherwise, if the robber is on W or on Z, then the marshals move on $\{g,\ c\}$ (see Figure 6.C). Since they keep the control of X, Y, T, and U during the move, then the robber can escape only to vertex W. Therefore, a further move on edge `d` allows the marshals to eventually capture the robber, as shown in Figure 6.D.

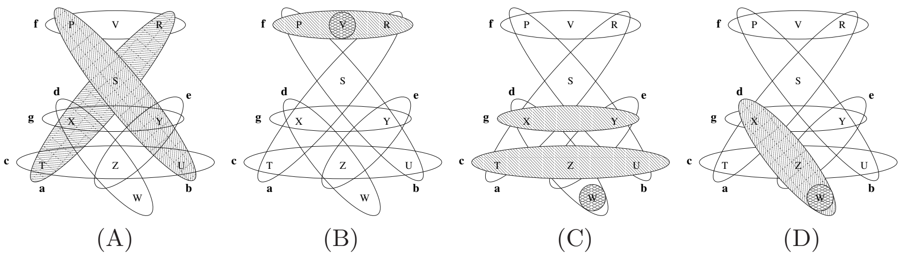


Fig. 6. (A) The first move of the marshals playing the game on $\mathcal{H}_{I_1}$; (B) move of the marshals if the robber stands on V (capture position); (C) move of the marshals if the robber stands on W or on Z; (D) the marshals capture the robber in W

In [23] we prove that there is a one-to-one correspondence between winning strategies for k marshals and hypertree decompositions of width at most k in a certain normal form.

Theorem 20 ([23]). *A hypergraph $\mathcal{H}$ has k-bounded hypertree width if and only k marshals have a winning strategy for the R&Ms game played on $\mathcal{H}$.*

9 Logical Characterization of Hypertree Width

Denote by L the fragment of first-order logic (FO) whose connectives are restricted to existential quantification and conjunction (i.e., $\neg$, $\vee$, and $\forall$ are disallowed). Kolaitis and Vardi [30] proved that the class of all queries having treewidth $< k$ coincides in expressive power with the k-variable fragment L^k of L, i.e., the class of all L formulas that use k variables only. Se also [13].

In [23], we characterize HW[k] in terms of a guarded logic. We show that $\mathrm{HW}[k] = \mathrm{GF}_k(\mathrm{L})$, where $\mathrm{GF}_k(\mathrm{L})$ denotes the k-guarded fragment of L. The 1-guarded fragment coincides with the classical notion of guardedness, where existentially quantified subformulas φ of a formula are always conjoined with a *guard*, i.e., an atom containing all free variables of φ. In the k-guarded fragment, up to k atoms may jointly act as a guard (for precise definitions, see [23]). For the particular case $k = 1$, this gives us a new characterization of the acyclic queries stating that the acyclic queries are precisely those expressible in the guarded fragment of L. In order to prove these results, we played the robber and marshals game on the appropriate query hypergraphs.

10 Comparison of Hypertree Width with Other Methods

We report about results comparing the Hypertree decomposition method with other methods for solving efficiently CSPs and conjunctive queries, which are based only on the structure of the hypergraph associated with the problem (we consider *tractability due to restricted structure*, as discussed in Section 2). We call these methods *decomposition methods (DM)*, because each one provides a decomposition which transforms any hypergraph to an acyclic hypergraph. For each decomposition method D, this transformation depends on a parameter called *D-width*. Let k be a fixed constant. The tractability class $C(D, k)$ is the (possibly infinite) set of hypergraphs having D-width $\leq k$. D ensures that every CQ or CSP instance whose associated hypergraph belongs to this class is polynomial-time solvable.

The main decomposition methods considered in database theory and in artificial intelligence are: *Treewidth* [34] (see also [30,17]), *Cycle Cutset* [9], *Tree Clustering* [10], *Induced Width (w^*)* cf. [9], *Hinge Decomposition* [27,26], *Hinge Decomposition + Tree Clustering* [26], *Cycle Hypercutset* [21], *Hypertree Decomposition*. All methods are briefly explained in [21]. Here, we do not consider the notion of query width, because deciding whether a hypergraph has bounded query width is NP-complete. However, recall that this notion is generalized by hypertree width, in that whenever a hypergraph has query width at most k, it has hypertree width at most k, too. The converse does not hold, in general [19]. For comparing decomposition methods we introduce the relations $\preceq$, $\rhd$, and $\prec\!\!\prec$ defined as follows:

$\underline{D_1 \preceq D_2}$, in words, D_2 *generalizes* D_1, if $\exists \delta \geq 0$ such that, $\forall k > 0$, $C(D_1, k) \subseteq C(D_2, k + \delta)$. Thus $D_1 \preceq D_2$ if every class of CSP instances which is tractable according to D_1 is also tractable according to D_2.

$\underline{D_1 \rhd D_2}$ (D_1 beats D_2) if there exists an integer k such that $\forall m$ $C(D_1, k) \not\subseteq C(D_2, m)$. To prove that $D_1 \rhd D_2$, it is sufficient to exhibit a class of hypergraphs contained in some $C(D_1, k)$ but in no $C(D_2, j)$ for $j \geq 0$. Intuitively, $D_1 \rhd D_2$ means that at least on some class of CSP instances, D_1 outperforms D_2.

$\underline{D_1 \prec\!\!\prec D_2}$ if $D_1 \preceq D_2$ and $D_2 \rhd D_1$. In this case we say that D_2 *strongly generalizes* D_1.

Mathematically, $\preceq$ is a *preorder*, i.e., it is reflexive, transitive but not antisymmetric. We say that D_1 *is* $\preceq$*-equivalent to* D_2, denoted $D_1 \equiv D_2$, if both $D_1 \preceq D_2$ and $D_2 \preceq D_1$ hold.

The decomposition methods D_1 and D_2 are *strongly incomparable* if both $D_1 \rhd D_2$ and $D_2 \rhd D_1$. Note that if D_1 and D_2 are strongly incomparable, then they are also incomparable w.r.t. the relations $\preceq$ and $\prec\!\!\prec$.

Figure 7 shows a representation of the hierarchy of DMs determined by the $\prec\!\!\prec$ relation. Each element of the hierarchy represents a DM, apart ¿from that containing the three $\preceq$-equivalent methods *Tree Clustering*, *Treewidth*, and w^*.

Theorem 21 ([21]). *For each pair D_1 and D_2 of decompositions methods represented in Figure 7, the following holds. There is a directed path from D_1 to D_2 iff $D_1 \prec\!\!\prec D_2$, i.e., iff D_2 strongly generalizes D_1. Moreover, D_1 and D_2 are not linked by any directed path iff they are strongly incomparable. Hence, Fig. 7 completely describes the relationships among the different methods.*

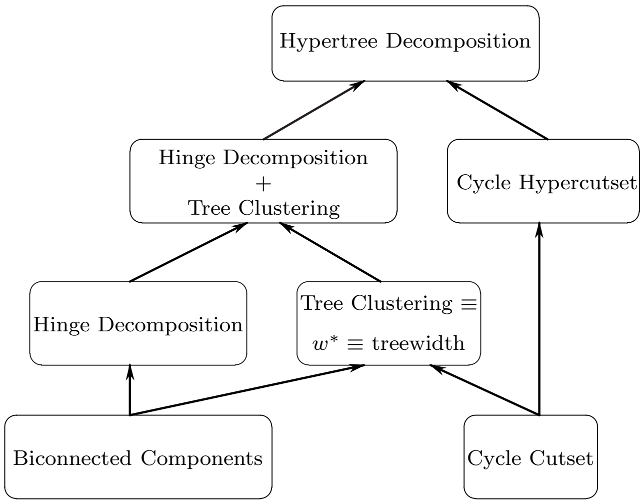


Fig. 7. Constraint tractability hierarchy

Recently, a comparison between hypertree width and Courcelle's concept of *clique-width* [7,8] was made [24]. Given that clique-width is defined for graphs, it had to be suitably adapted to hypergraphs. Defining the clique-width of a hypergraph $\mathcal{H}$ as the cliquewidth of its primal graph makes no sense in the context of CSP-tractability, because then CSPs of bounded clique-width would be intractable. Therefore, in [24], the clique-width $\mathcal{H}$ is defined as the clique-width of its incidence graph $G_I^i(\mathcal{H})$. With this definition it could be shown in [24] that

(a) CSP's whose hypergraphs have bounded clique-width are tractable, and (b) bounded hypertree width strongly generalizes bounded clique-width.

11 Open Problems

Several questions are left for future research. In particular, it would be interesting to know whether the method of hypertree decompositions can be further generalized. For instance, let us define the concept of *generalized hypertree decomposition* by just dropping condition 4 from the definition of hypertree decomposition (Def. 11). Correspondingly, we can introduce the concept of *generalized hypertree width* $ghw(\mathcal{H})$ of a hypergraph $\mathcal{H}$. We know that all classes of Boolean queries having bounded *ghw* can be answered in polynomial time. But we currently do not know whether these classes of queries are polynomially recognizable. This recognition problem is related to the mysterious *hypergraph sandwich problem* [31], which has remained unsolved for a long time. If the latter is polynomially solvable, then also queries of bounded *ghw* are polynomially recognizable. Another question concerns the time complexity of recognizing queries of bounded hypertree width. Is this problem fixed-parameter tractable such as the recognition of graphs of bounded treewidth?

Acknowledgements. Research supported by *FWF (Austrian Science Funds)* under the project Z29-INF (Wittgenstein Award), and by *MURST* under project COFIN-2000 "From Data to Information (D2I)"

References

1. L. Bachmair, Ta Chen, C. R. Ramakrishnan, and I. V. Ramakrishnan. Subsumption Algorithms Based on Search Trees. *Proc. CAAP'96*, Springer LNCS Vol.1059. 42
2. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the Desiderability of Acyclic Database Schemes. *Journal of ACM*, 30(3):479–513, 1983. 42
3. P. A. Bernstein, and N. Goodman. The power of natural semijoins. *SIAM J. Computing*, 10(4):751–771, 1981. 42
4. H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Computing*, 25(6):1305-1317, 1996. 44
5. A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in relational Databases. *Proc. STOC'77*, pp.77–90, 1977. 40
6. Ch. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *Theoretical Computer Science*, 239(2):211–229, 2000. 38, 45, 46, 47
7. B. Courcelle. Graph Rewriting: an algebraic and logic approach. Chapter 5 in *Handbook of Theor. Comp. Sci., vol. B*, J. Van Leeuwen ed., 1990. 38, 44, 54
8. B. Courcelle: Monadic second-order logic of graphs VII: Graphs as relational structures, in Theoretical Computer Science, Vol 101, pp. 3-33 (1992). 38, 54
9. R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence*, second edition, Wiley and Sons, pp. 276-f285, 1992. 39, 43, 45, 53
10. R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pp. 353–366, 1989. 43, 45, 53

11. R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. of the ACM*, 30(3):514–550, 1983. 42
12. T. Feder and M. Y.Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. Comput.*, 28(1):57–104, 1998. 39
13. J. Flum, M. Frick, and M. Grohe. Query Evaluation via Tree-Decomposition. In *Proc. of ICDT'01*, Springer LNCS, Vol. 1973, pp.22–38, 2001. 45, 53
14. E. C. Freuder. Complexity of K-Tree Structured Constraint Satisfaction Problems. *Proc. of AAAI'90*, 1990. 45
15. H. Gaifman. On local and nonlocal properties. In *Logic Colloquium '81*, pp. 105–135, J. Stern ed., North Holland, 1982. 42
16. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of* NP*-completeness*. Freeman and Comp., NY, USA, 1979.
17. G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries. *Journal of the ACM*, 48(3), 2001. Preliminary version in FOCS'98. 38, 41, 43, 45, 46, 53
18. G. Gottlob, N. Leone, and F. Scarcello. Computing LOGCFL Certificates. *Theoretical Computer Sciences*, to appear. Preliminary version in ICALP'99. 44, 50
19. G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable Queries. JCSS. to appear. Preliminary version in PODS'99. 38, 47, 48, 50, 51, 53
20. G. Gottlob, N. Leone, and F. Scarcello. "On Tractable Queries and Constraints," in *Proc. DEXA'99*, Florence, 1999, LNCS 1677, pp. 1-15, Springer. 50
21. G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124(2):243–282, 2000. Preliminary version in *IJCAI'99*. 38, 45, 51, 53, 54
22. G. Gottlob, N. Leone, and F. Scarcello. "Advanced Parallel Algorithms for Processing Acyclic Conjunctive Queries, Rules, and Constraints," *Proc. SEKE00*, pp. 167–176, KSI Ed., Chicago, USA, July 6-8, 2000. 38, 43
23. G. Gottlob, N. Leone, and F. Scarcello. "Robbers, Marshals, and Guards: Game-Theoretic and Logical Characterizations of Hypertree Width," in *Proc. PODS'01*. 38, 51, 52, 53
24. G. Gottlob and R. Pichler. Hypergraphs in Model Checking: Acyclicity and Hypertree-Width Versus Clique-Width. *Proc. ICALP 2001*, to appear. 38, 54
25. M. Grohe, T. Schwentick, and L. Segoufin. When is the Evaluation of Conjunctive Queries Tractable? Proc. ACM STOC 2001. 45
26. M. Gyssens, P. G. Jeavons, and D. A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 1994. 53
27. M. Gyssens, and J. Paredaens. A Decomposition Methodology for Cyclic Databases. In *Advances in Database Theory*, vol.2, 1984. 53
28. P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548.
29. D. S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 2, pp.67–161. Elsevier Science Publishers B. V. (North-Holland), 1990. 43
30. Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences*, 61:302–332, 2000. 39, 40, 45, 52, 53
31. A. Lustig and O. Shmueli. Acyclic Hypergraph Projections. *J. of Algorithms*, 30:400–422, 1999. 55
32. D. Maier. *The Theory of Relational Databases*, Rochville, Md, Computer Science Press, 1986. 42, 45

33. J. Pearson and P. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Halloway University of London, 1997. 40
34. N. Robertson and P. D. Seymour. Graph Minors II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7:309-322, 1986. 44, 53
35. T. J. Schaefer. The Complexity of Satisfiability Problems. In *Proc. STOC'78*. 40
36. P. D. Seymour and R. Thomas. Graph Searching and a Min-Max Theorem for Tree-Width. *J. of Combinatorial Theory, Series B*, 58:22–33, 1993. 51
37. 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 J. Computing*, 13(3):566-579, 1984. 43
38. M. Vardi. Complexity of Relational Query Languages. In *Proc. of 14th ACM STOC*, pp. 137–146, 1982. 43
39. E. Wanke. Bounded Tree-Width and LOGCFL. *Journal of Algorithms*, 16:470–491, 1994. 44
40. A. N. Wilschut, J. Flokstra, and P. M. G. Apers. Parallel evaluation of multi-join queries. In *Proc. of SIGMOD'95*, San Jose, CA USA, pp.115–126, 1995. 43
41. M. Yannakakis. Algorithms for Acyclic Database Schemes. *Proc. VLDB'81*, pp. 82–94, C. Zaniolo and C. Delobel Eds., Cannes, France, 1981. 43