A Comparison of Structural CSP Decomposition Methods ⋆

Georg Gottlob

Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria.
E-mail: gottlob@dbai.tuwien.ac.at

Nicola Leone

Department of Mathematics, University of Calabria, I-87036 Rende (CS), Italy.
E-mail: leone@unical.at

Francesco Scarcello

Dipartimento di Elettronica, Informatica e Sistemistica, University of Calabria, I-87036 Rende (CS), Italy. E-mail: scarcello@deis.unical.it

Abstract

We compare tractable classes of constraint satisfaction problems (CSPs). We first give a uniform presentation of the major structural CSP decomposition methods. We then introduce a new class of tractable CSPs based on the concept of *hypertree decomposition* recently developed in Database Theory, and analyze the cost of solving CSPs having bounded hypertree-width. We provide a framework for comparing parametric decomposition-based methods according to tractability criteria and compare the most relevant methods. We show that the method of hypertree decomposition dominates the others in the case of general CSPs (i.e., CSPs of unbounded arity). We also make comparisons for the restricted case of binary CSPs. Finally, we consider the application of decomposition methods to the dual graph of a hypergraph. In fact, this technique is often used to exploit binary decomposition methods for nonbinary CSPs. However, even in this case, the hypertree-decomposition method turns out to be the most general method.

Key words: Constraint satisfaction, decomposition methods, hypergraphs, tractable cases, degree of cyclicity, treewidth, hypertree width, tree-clustering, cycle cutsets, biconnected components.

⋆ Part of this work has been published in preliminary form in the proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99), Stockholm, 1999.
Please, address correspondence to Francesco Scarcello.

1 Introduction and Summary of Results

The efficient solution of *Constraint Satisfaction Problems (CSPs)* has been for many years an important goal of AI research. Constraint satisfaction is a central issue of *problem solving* and has an impressive spectrum of applications [23]. A constraint (S_i, R_i) consists of a *constraint scope* S_i, i.e., a list of variables and an associated *constraint relation* r_i containing the legal combinations of values. A CSP consists of a set $\{(S_1, r_1), (S_2, r_2), \ldots, (S_q, r_q)\}$ of constraints whose variables may overlap (for a precise definition, see Section 2). A solution to a CSP consists a of an assignment of values to all variables such that all constraints are simultaneously satisfied. By *solving* a CSP we mean detemining whether the problem has a solution at all (i.e., checking for *constraint satisfiability*), and, if so, compute one solution.

Constraint satisfiability is equivalent to various database problems [4,18,7,21], e.g., to the problem of conjunctive query containment [21], or to the problem of evaluating *Boolean conjunctive queries* over a relational database [22] (for a discussion of this and other equivalent problems, see [15]). Actually, evaluating Boolean conjunctive queries, and deciding constraint satisfaction can be also recast as the same fundamental algebraic problem of deciding whether, given two finite relational structures A and B, there exists a homomorphism $f : A \to B$ [21].

Constraint satisfiability in its general form is well-known to be NP-hard. Much effort has been spent by both the AI and database communities to identify *tractable classes* of CSPs. Both communities have obtained deep and useful results in this direction. The various successful approaches to obtain tractable CSP classes can be divided into two main groups [23]:

Tractability due to restricted structure. This includes all tractable classes of CSPs that are identified solely on the base of the structure of the constraint scopes $\{S_1, \ldots S_q\}$, independently of the actual constraint relations $r_1, \ldots, r_q$.

Tractability due to restricted constraint relations. This includes all classes that are tractable due to particular properties of the constraint relations $r_1, \ldots, r_q$.

This paper deals with tractability due to restricted structure. There are several papers proposing polynomially tractable classes of constraints based on different structural properties of the constraint scopes. Usually, these properties can be formalized as graph-theoretic properties of the *constraint graph* in case of binary constraints, or of the *constraint hypergraph* in the general case. The constraint hypergraph of a CSP is the hypergraph whose vertices are the variables of the CSP and whose hyperedges are the sets of all those variables which occur together in a constraint scope.

It is well known that CSPs with *acyclic* constraint hypergraphs are polynomially solvable [7]. The known structural properties that lead to tractable CSP classes are all (explicitly or implicitly) based on some generalization of acyclicity. In particular, each method defines some concept of *width* which can be interpreted as a measure of cyclicity of the underlying constraint (hyper)graph such that, for each fixed width k, all CSPs of width bounded by k are solvable in polynomial time. There is a plethora of proposed methods based on various different measures of cyclicity, but little was known so far on the relative strength of the different methods. A comparison of the main methods is called for.

In this paper we establish a framework for uniformly defining and comparing structural CSP decomposition methods. Within this framework we compare the main methods that have been published so far. In particular, we deal with the following methods (which are reviewed in detail in Section 4): Cycle Cutset [7], Tree Clustering [9], Treewidth [24], Hinge Decomposition [18,19], Hinge Decomposition with Tree Clustering [18], Cycle Hypercutset, and Hypertree Decomposition [16].

We first point out that every considered CSP-decomposition method D gives rise to an infinite hierarchy of CSP classes:

$$C(D,1) \subset C(D,2) \subset \cdots \subset C(D,i), \cdots\cdots$$

such that the CSPs of each class $C(D,k)$ are solvable in time bounded by a polynomial. In particular, for each CSP C belonging to class $C(D,k)$ there exists a *decomposition* of width $\leq k$, i.e., a data structure witnessing that C can be transformed in polynomial time into an equivalent acyclic CSP.

For each CSP-decomposition method D, the class $C(D,k)$ is a tractable class of CSPs because the following important tasks are tractable:

(1) Checking membership of a CSP C in $C(D,k)$, and computing a corresponding CSP decomposition for C.
(2) Solving the CSP C. In turn, this task usually consists of the following two subtasks:
   - Transforming C in polynomial time into an equivalent acyclic CSP C', and
   - solving C' in polynomial time by using well-known algorithms.

In this paper we compare only those methods that are tractable in the above sense. In fact, there are methods for solving CSPs, reported in the literature, for which only one of the two tasks (1) and (2) above is tractable, while the other one is NP-hard. For instance, task (1) is NP-complete for the method of *bounded query decompositions* defined by Chekuri and Rajaraman [6] (see [16] for an NP-completeness proof), while task (2) is intractable for an early method proposed by Freuder [10,11] (see Section 4 for an NP-completeness proof).

For a pair of decomposition methods D_1 and D_2, we define the following comparison criteria:

Generalization. D_2 *generalizes* D_1 if there exists a constant δ such that, for each level k, $C(D_1, k) \subseteq C(D_2, k + \delta)$ holds. In practical terms, this means that whenever a class $\mathcal{C}$ of constraints is tractable according to method D_1, it is also tractable according to D_2. Moreover, the worst case runtime upper bound guaranteed by method D_2 is polynomially bounded by the worst case upper bound guaranteed by method D_1; more precisely, the overhead of D_2 with respect to D_1 is at most n^δ, where n is the size of the input CSP. Note that for all pairs of methods compared in this paper, δ is at most 1. This means that *there is no significant loss of efficiency* when replacing method D_1 with the more general method D_2.

Beating. D_2 *beats* D_1 if there exists an integer k such that $C(D_2, k)$ is not contained in class $C(D_1, m)$ for any m. Intuitively, this means that some classes of problems are tractable according to D_2 but not according to D_1. For such classes, using D_2 is thus better than using D_1.

Strong generalization. D_2 *strongly generalizes* D_1 if D_2 generalizes D_1 and D_2 beats D_1. This means that D_2 is really the more powerful method, given that, whenever D_1 guarantees polynomial runtime for constraint solving, then also D_2 guarantees tractable constraint solving, but there are classes of constraints that can be solved in polynomial time by using D_2 but are not tractable according to D_1.

Equivalence. D_1 and D_2 are *equivalent* if D_1 generalizes D_2 and D_2 generalizes D_1. Intuitively, this means that the methods are polynomial on the same classes of CSPs and do not differ significantly from each other.

In this paper we completely classify all above-mentioned decomposition methods according to these criteria. The result of the classification is given in Figure 1. This figure, in addition mentions another method (ω^*) which is known to be equivalent to the tree-clustering method [9].

An arrow from a method D_1 to a method D_2 in Figure 1 indicates that D_2 is strongly more general than D_1. Since this relationship is transitive, also a directed path between two methods indicates the same relationship. The picture is *complete* in the sense that there is a directed path from method D_1 to method D_2 *if and only if* D_2 strongly generalizes D_1. On the other hand, whenever two methods are not related by a directed path, then they are *incomparable* with respect to the generalization relation, and, moreover, each of the two methods beats the other.

Figure 1 shows that the method of Hypertree Decompositions dominates all other methods, as it is strongly more general than the other decomposition methods. This method was originally introduced in the database field for identifying a large class of tractable conjunctive queries [16]. In this paper we adapt this notion to the setting of constraints and we show that constraints of bounded hypertree-width are polynomially solvable, providing a precise complexity analysis. In particular, we show that CSPs of hypertree width k can be solved in time $O(n^{k+1} \times \log n)$.

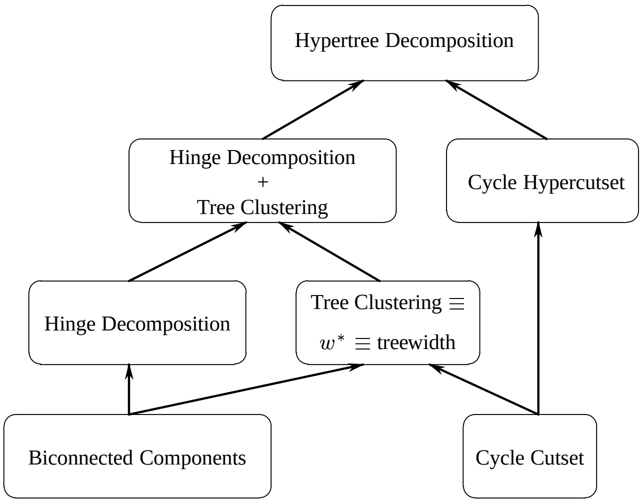


Figure 1. Constraint Tractability Hierarchy

Hypertree width is a measure of cyclicity specifically designed for *hypergraphs*. It is interesting to see how the situation changes in the special case of *graphs*, i.e., of *binary CSPs*. To answer this question, we have compared all considered method in the binary case (in Section 8; see Figure 25). Again, it turns out that the method of Hypertree Decomposition dominates the others, but this time in a slightly weaker sense to be explained in Section 8.

It was recently asked [1] whether the method of Hypertree Decompositions can be explained in terms of simpler and well-known graph cyclicity measures. To every hypergraph $\mathcal{H}$ one defines the *dual graph* of $\mathcal{H}$ by taking as vertices the hyperedges of $\mathcal{H}$ and by connecting two vertices by an edge if their corresponding hyperedges intersect. The question arose whether the hypertree width of a hypergraph coincides with the treewidth or TCLUSTER width of the dual graph of $\mathcal{H}$ (See Section 9 for definitions). We study this interesting question in Section 9 and give a negative answer. More generally, we show that the method of hypertree decompositions *strongly generalizes* all relevant binary methods based on the dual graph of a given hypergraph.

This paper is organized as follows. Section 2 contains preliminaries on CSPs. In Section 3 we discuss tractability of CSPs due to restricted structure. In Section 4 we review well-known CSP decomposition methods. In Section 5 we describe the new method of *hypertree decompositions* and analyze the cost of solving CSPs having bounded hypertree-width. In Section 6 we explain our comparison criteria and in Section 7 we present the comparison results for general CSPs. The case of binary CSPs is briefly discussed in Section 8. In Section 9 we consider the application of "binary" methods to the dual graph of a hypergraph. Finally, in Section 10, we draw our conclusions.

[1] Rina Dechter, personal communication at IJCAI'99.

2 Constraint Satisfaction Problems

An instance of a *constraint satisfaction problem (CSP)* (also *constraint network*) is a triple $I = (Var, U, \mathcal{C})$, where Var is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), 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 the *constraint relation*. (The tuples of r_i indicate the allowed combinations of simultaneous values for the variables S_i). A *solution* to a CSP instance is a substitution $\vartheta : Var \longrightarrow U$, such that for each $1 \leq i \leq q$, $S_i\vartheta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability (CS)*. (This definition is taken almost verbatim from [20].)

Many well-known problems in Computer Science and Mathematics can be formulated as CSPs.

Example 1 The famous *graph three-colorability (3COL)* problem, i.e., deciding whether the vertices of a graph $G = (Vertices, Edges)$ can be colored by three colors (say: red, green, blue) such that no edge links two vertices having the same color, is formulated as follows as a CSP. The set Var contains a variable X_v for each vertex $v \in Vertices$. For each edge $e = \{v, w\} \in Edges$, where $v < w$ according to some ordering on *Vertices*, the set $\mathcal{C}$ contains a constraint $C_e = (S_e, r_e)$, where $S_e = (X_v, X_w)$ and r_e is the relation $r_{\neq}$ consisting of all pairs of different colors, i.e., $r_{\neq} = \{\langle red, green\rangle, \langle red, blue\rangle, \langle green, red\rangle, \langle green, blue\rangle, \langle blue, red\rangle, \langle blue, green\rangle\}$.

For instance, the set of constraints for the graph G_1 in Figure 2 is the following $\mathcal{C} = \{((A, B), r_{\neq}), ((A, D), r_{\neq}), ((A, G), r_{\neq}), ((B, C), r_{\neq}), \ldots, ((G, H), r_{\neq})\}$.

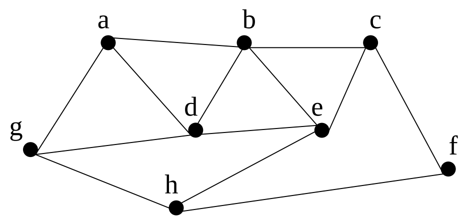


Figure 2. The graph G_1

Example 2 Figure 3 shows a combinatorial crossword puzzle, which is a typical CSP [7,23]. A set of legal words is associated to each horizontal or vertical array of white boxes delimited by black boxes. A solution to the puzzle is an assignment of a letter to each white box such that to each white array is assigned a word from its set of legal words.

This problem is represented as follows. There is a variable X_i for each white box, and a constraint C for each array D of white boxes. (For simplicity, we just write the index i for variable X_i.) The scope of C is the list of variables corresponding to the white boxes of the sequence D; the relation of C contains the legal words for D. For the example in Figure 3, we have $C_{1H} = ((1,2,3,4,5), r_{1H})$, $C_{8H} = ((8,9,10), r_{8H})$, $C_{11H} = ((11,12,13), r_{11H})$, $C_{20H} = ((20,21,22,23,24,25,26), r_{20H})$, $C_{1V} = ((1,7,11,16,20), r_{1V})$, $C_{5V} = ((5,8,14,18,24), r_{5V})$, $C_{6V} = ((6,10,15,19,26), r_{6V})$, $C_{13V} = ((13,17,22), r_{13V})$. Subscripts H and V stand for "Horizontal" and "Vertical," respectively, resembling the usual naming of definitions in the crossword puzzles. A possible instance for the relation r_{1H} is $\{\langle h,o,u,s,e\rangle, \langle c,o,i,n,s\rangle, \langle b,l,o,c,k\rangle\}$.

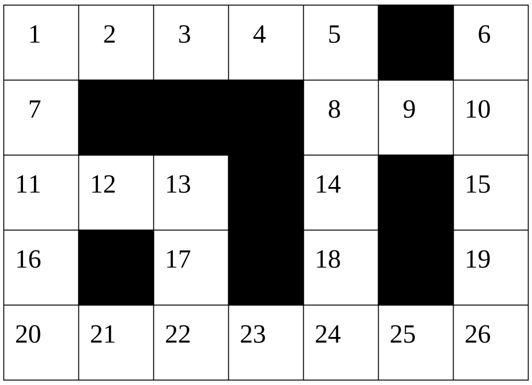


Figure 3. A crossword puzzle

It is well-known and easy to see that Constraint Satisfiability is an NP-complete problem. Membership in NP is obvious. NP-hardness follows, e.g., immediately from the NP hardness of 3COL [13].

3 Tractable classes of CSPs

Much effort has been spent by both the AI and database communities to indentify *tractable classes* of CSPs. Both communities have obtained deep and useful results in this direction. The various successful approaches to obtain tractable CSP classes can be divided into two main groups [23]:

1. **Tractability due to restricted structure.** This includes all tractable classes of CSPs that are identified solely on the base of the structure of the constraint scopes $\{S_1, \ldots S_q\}$, independently of the actual constraint relations $r_1, \ldots, r_q$.

2. **Tractability due to restricted constraints.** This includes all classes that are tractable due to particular properties of the constraint relations $r_1, \ldots, r_q$.

The present paper deals with tractability due to restricted structure.

The *structure* of a CSP is best represented by its associated *hypergraph* and by the corresponding *primal graph*, defined as follows. To any CSP instance $I = (Var, U, \mathcal{C})$, we associate a hypergraph $\mathcal{H}_I = (V, H)$, where $V = Var$, and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, where $var(S)$ denotes the set of variables in the scope S of the constraint C. Figure 4 shows the hypergraph $\mathcal{H}_{cp}$ associated to the crossword puzzle of Example 2.

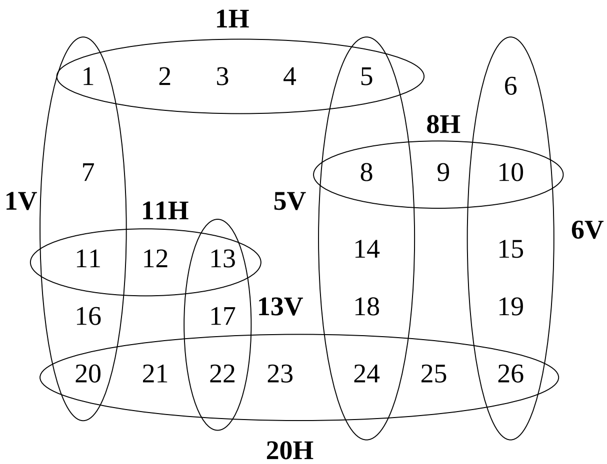


Figure 4. Hypergraph $\mathcal{H}_{cp}$ of the crossword puzzle in Example 2

Since in this paper we always deal with hypergraphs corresponding to CSPs instances, the vertices of any hypergraph $\mathcal{H} = (V, H)$ can be viewed as the variables of some constraint satisfaction problem. Thus, we will often use the term *variable* as a synonym for vertex, when referring to elements of V. Moreover, for the hypergraph $\mathcal{H} = (V, H)$, $var(\mathcal{H})$ and $edges(\mathcal{H})$ denote the sets V and H, respectively.

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 all constraints of a CSP are binary, then its associated hypergraph is identical to its primal graph.

The most basic and most fundamental structural property considered in the context of CSPs (and conjunctive database queries) is *acyclicity*. It was recognized independently in AI and in database theory that *acyclic* CSPs are polynomially solvable. A CSP I is acyclic 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 $edges(\mathcal{H}_I)$ [2].

A *join tree* $\mathcal{JT}(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the edges of $\mathcal{H}$ such that, whenever the same variable $X \in V$ 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 vertex on the unique path linking A_1 and A_2 in $\mathcal{JT}(\mathcal{H})$. In other words, the set of vertices 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 [3,2,22]. There exist various equivalent characterizations of acyclic hypergraphs [2,14,22]. Checking the satisfiability of acyclic CSPs (or, equivalently, evaluating acyclic conjunctive queries) is not only tractable but also highly parallelizable. In fact, as shown in [15], this problem is complete for the complexity class LOGCFL, a very low class contained in the parallel classes AC_1 and NC_2.

Many CSPs arising in practice are not acyclic but are in some sense or another *close* to acyclic CSPs. In fact, the hypergraphs associated with many naturally arising CSPs contain either few cycles or small cycles, or can be transformed to acyclic CSPs by simple operations (such as, e.g., lumping together small groups of vertices). Consequently, CSP research in AI and in database theory concentrated on identifying, defining, and studying suitable classes of *nearly acyclic* CSPs, or, equivalently, decomposition methods, i.e., techniques for *decomposing* cyclic CSPs into acyclic CSPs [23,7].

4 Decomposition Methods

In order to study and compare various decomposition methods, we find it useful to introduce a general formal framework for this notion.

Let $\mathcal{H}$ be a hypergraph. For any set of edges $H' \subseteq edges(\mathcal{H})$, let $var(H') = \bigcup_{h \in H'} h$. Without loss of generality, we assume that $var(H) = var(\mathcal{H})$, i.e., every variable in $var(\mathcal{H})$ occurs in at least one edge of $\mathcal{H}$, and hence, any hypergraph can be simply represented by the set of its edges. Moreover, we assume without loss of generality that all hypergraphs under consideration are both *connected*, i.e., their primal graph consists of a single connected component, and *reduced*, i.e., no hyperedge is contained in any other hyperedge. All our definitions and results easily extend to general hypergraphs.

Let $\mathcal{H}S$ be the set of all (reduced and connected) hypergraphs. A *decomposition method* (short: DM) D associates to any hypergraph $\mathcal{H} \in \mathcal{H}S$ a parameter D-*width($\mathcal{H}$)*, called the D *width* of $\mathcal{H}$.

The decomposition method D ensures that, for fixed k, every CSP instance I whose hypergraph $\mathcal{H}_I$ has D-width $\leq k$ is polynomially solvable, i.e., it is solvable in $p(||I||) = O(||I||^{O(1)})$ time, where $||I||$ denotes the size of I. For any CSP instance I, the size of I is defined in the standard way, i.e., as the number of bits needed for encoding I by listing, for each constraint in I, its constraint scope and all tuples occurring in its constraint relation.

For any $k > 0$, the k-*tractable class* $C(D, k)$ of D is defined by

$$C(D, k) = \{\mathcal{H} \mid D\text{-width}(\mathcal{H}) \leq k\}.$$

Thus, $C(D, k)$ collects the set of CSP instances which, for fixed k, are polynomially solvable by using the strategy D. Typically, the polynomial $p(\| I \|)$ depends on the parameter k. In particular, for each D, there exists a function f such that, for each k, each instance $I \in C(D, k)$ can be transformed in time $O(\|I\|^{O(f(k))})$ into an equivalent *acyclic* CSP instance. (It follows that all problems in $C(D, k)$ are polynomially solvable.)

Every DM D is complete with respect to $\mathcal{H}S$, i.e., $\mathcal{H}S = \bigcup_{k \geq 1} C(D, k)$. Note that, by our definitions, it holds that D-width$(\mathcal{H}) = \min\{k \mid \mathcal{H} \in C(D, k)\}$.

All tractable classes based on restricted structure that we have studied in the literature fit into this framework. We next describe how the notion of width is defined in the decomposition methods we shall compare in this paper. Detailed descriptions of these methods can be found in the corresponding reference (see below) and in many surveys on this subject, e.g., [23,7].

4.1 Biconnected Components (short: `BICOMP`*)* [11]

Let $G = (V, E)$ be a graph. A vertex $p \in V$ is a *separating vertex* for G if, by removing p from G, the number of connected components of G increases. A biconnected component of G is a maximal set of vertices $C \subseteq V$ such that the subgraph of G induced by C is connected and remains connected after any one-vertex removal, i.e., has no separating vertices.

It is well known that, from any graph G, we can compute in linear time a vertex-labeled tree $\langle T, \chi \rangle$, where the labeling function χ is a bijective function that associates to each vertex of the tree T a set of vertices S of G, such that S is either a biconnected component of G, or a singleton containing a separating vertex for G. There is an edge $\{p, q\}$ in the tree T, if $\chi(p)$ is a biconnected component of G and $\chi(q)$ contains a separating vertex for G belonging to the component $\chi(p)$, i.e., $\chi(q) \subseteq \chi(p)$, holds. We say that $\langle T, \chi \rangle$ is the `BICOMP` decomposition of G.

For a hypergraph $\mathcal{H}$, the `BICOMP` decomposition of $\mathcal{H}$ is the `BICOMP` decomposition of its primal graph, and the *biconnected width* of $\mathcal{H}$, denoted by `BICOMP`-width$(\mathcal{H})$, is the maximum number of vertices over the biconnected components of the primal graph of $\mathcal{H}$.

Example 3 Figure 5.a shows a hypergraph $\mathcal{H}_b$ and Figure 5.b its primal graph. The vertices G, C, D, and E are the separating vertices of this primal graph. Note that the maximum number of vertices over its biconnected components is 3, and thus `BICOMP`-width$(\mathcal{H}) = 3$. Figure 6 shows the `BICOMP` decomposition of $\mathcal{H}_b$.

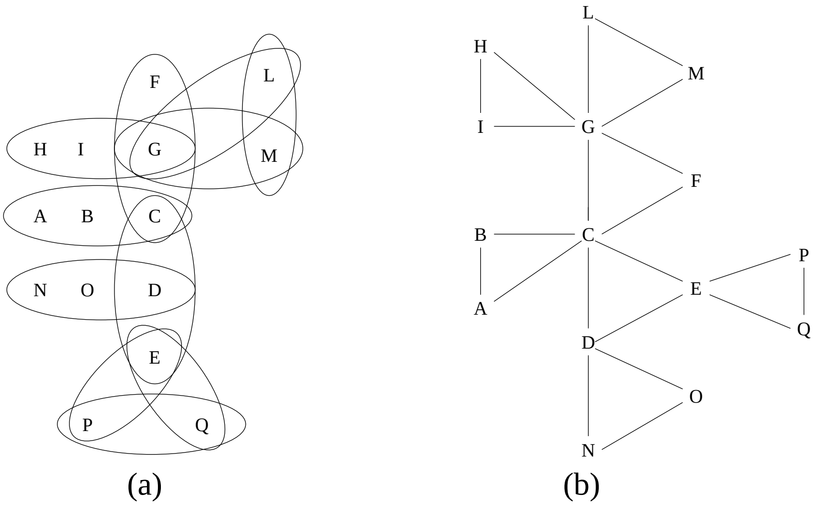


Figure 5. (a) The hypergraph $\mathcal{H}_b$, and (b) its primal graph

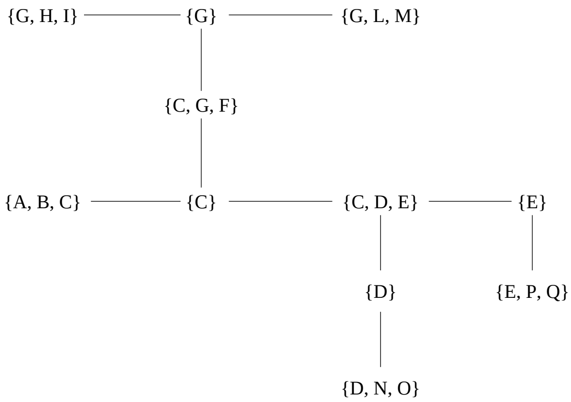


Figure 6. The `BICOMP` decomposition of the hypergraph $\mathcal{H}_b$ in Example 3

4.2 Tree Clustering (short: `TCLUSTER`*)* [9]

The *tree clustering* method is based on a triangulation algorithm which transforms the primal graph $G = (V, E)$ of any CSP instance I into a chordal graph G'. The acyclic hypergraph $\mathcal{H}(G')$ having the same set of vertices as G' and the maximal cliques of G' as its hyperedges is a `TCLUSTER` decomposition of $\mathcal{H}_I$. Intuitively, the hyperedges of $\mathcal{H}(G')$ are used to build the constraints of an acyclic CSP I' equivalent to I. The width of the `TCLUSTER` decomposition $\mathcal{H}(G')$ is the maximum cardinality of its hyperedges. The *tree-clustering width* (short: `TCLUSTER` *width*) of $\mathcal{H}_I$ is 1 if $\mathcal{H}_I$ is an acyclic hypergraph; otherwise, it is equal to the minimum width over the `TCLUSTER` decompositions of $\mathcal{H}_I$.

Example 4 Consider the hypergraph $\mathcal{H}_{tc}$ shown in Figure 7.a. Figure 7.b shows its primal graph.

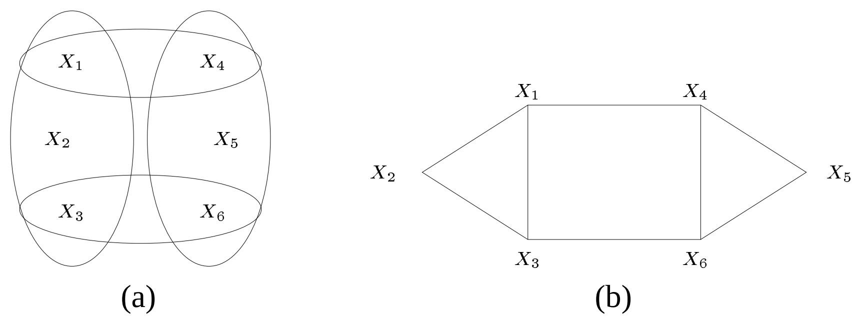


Figure 7. (a) The hypergraph $\mathcal{H}_{tc}$, and (b) its primal graph

This graph can be triangulated as shown in Figure 8.a. If we associate a hyperedge to each maximal clique of this triangulated graph, we get the acyclic hypergraph shown in Figure 8.b. This acyclic hypergraph is a `TCLUSTER` decomposition of $\mathcal{H}_{tc}$ of width 3. Moreover, it is easy to see that there is no `TCLUSTER` decomposition for $\mathcal{H}_{tc}$ having a smaller width, and hence the `TCLUSTER` width of $\mathcal{H}_{tc}$ is 3.

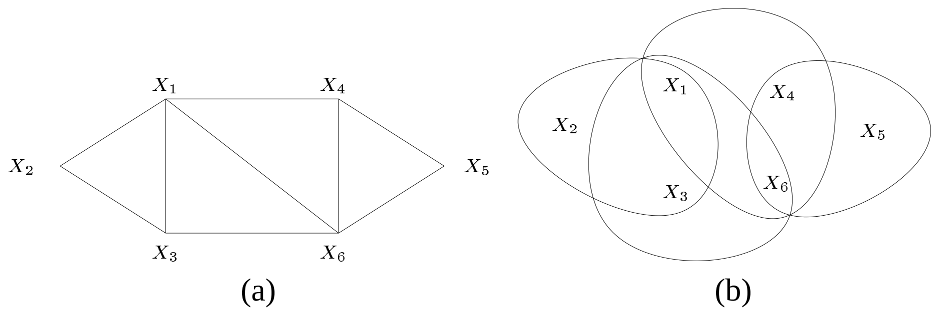


Figure 8. (a) A triangulation of the primal graph of $\mathcal{H}_{tc}$, and (b) a `TCLUSTER` decomposition of $\mathcal{H}_{tc}$

4.3 Treewidth (`TREEWIDTH`*)* [24]

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 hypergraph $\mathcal{H}$ is 1 if $\mathcal{H}$ is an acyclic hypergraph; otherwise, it is equal to the treewidth of its primal graph. As pointed out below, `TREEWIDTH` and `TCLUSTER` are two equivalent methods.

Example 5 Consider again the hypergraph $\mathcal{H}_{tc}$ in Example 4. Figure 9 show a tree decomposition of $\mathcal{H}_{tc}$ having width 2. It follows that the treewidth of $\mathcal{H}_{tc}$ is 2 as only hypergraphs having acyclic primal graphs have treewidth 1.

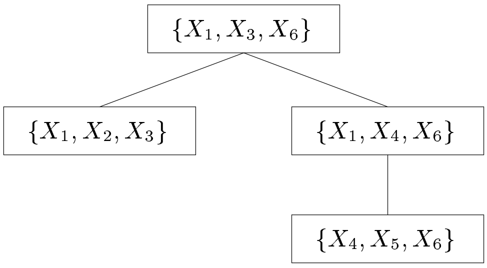


Figure 9. A tree decomposition of hypergraph $\mathcal{H}_{tc}$ in Example 4

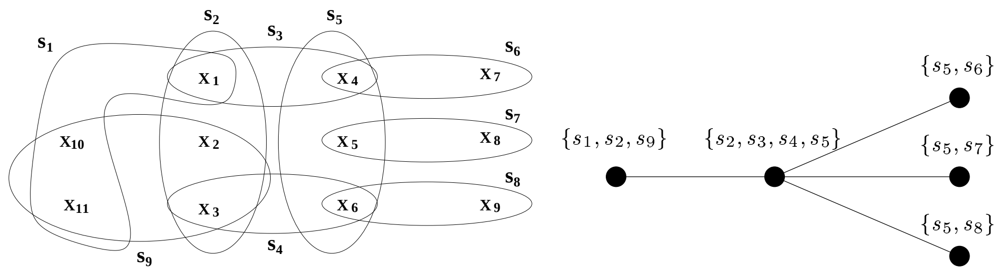
4.4 Hinge Decompositions (short: `HINGE`*)* [18,19]

Let $\mathcal{H}$ be a hypergraph, $H \subseteq edges(\mathcal{H})$, and $F \subseteq edges(\mathcal{H}) - H$. Then F is called *connected with respect to H* if, for any two edges $e, f \in F$, there exists a sequence $e_1, ..., e_n$ of edges in F such that (i) $e_1 = e$; (ii) for $i = 1, ..., n-1$, $e_i \cap e_{i+1}$ is not contained in $\bigcup_{h \in H} h$; and (iii) $e_n = f$. The maximal connected subsets of $edges(\mathcal{H}) - H$ with respect to H are called the *connected components of $\mathcal{H}$ with respect to H*. It is easy to see that the connected components of $\mathcal{H}$ with respect to H form a partition of $edges(\mathcal{H}) - H$.

Let $\mathcal{H} \in \mathcal{HS}$ and let H be either $edges(\mathcal{H})$ or a proper subset of $edges(\mathcal{H})$ containing at least two edges. Let $C_1, ..., C_m$ be the connected components of $\mathcal{H}$ with respect to H. Then, H is a *hinge* if, for $i = 1, ..., m$, there exists an edge $h_i \in H$ such that $var(edges(C_i)) \cap var(H)) \subseteq h_i$. A hinge is *minimal* if it does not contain any other hinge.

A *hinge decomposition* of $\mathcal{H}$ is a tree T such that all the following conditions hold: (1) the vertices of T are minimal hinges of $\mathcal{H}$; (2) each edge in $edges(\mathcal{H})$ is contained in at least one vertex of T; (3) two adjacent vertices A and B of T share precisely one edge $L \in edges(\mathcal{H})$; moreover, L consists exactly of the variables shared by A and B (i.e., $L = var(A) \cap var(B)$); (4) the variables of $\mathcal{H}$ shared by two vertices of T are entirely contained within each vertex on their connecting path in T.

It was shown in [19] that, for any CSP instance I, the cardinality of the largest vertex of any hinge decomposition of $\mathcal{H}_I$ is an invariant of $\mathcal{H}_I$, and is equal to the cardinality of the largest minimal hinge of $\mathcal{H}_I$. This number is called the *degree of cyclicity* of $\mathcal{H}_I$. We will also refer to it as the `HINGE` *width* of $\mathcal{H}_I$.



(a) (b)

Figure 10. (a) Hypergraph $\mathcal{H}_{hg}$, and (b) a hinge-tree decomposition of $\mathcal{H}_{hg}$

Example 6 Consider a CSP instance I_{hg} having the following constraint scopes:

$$s_1(X_1, X_{10}, X_{11}); s_2(X_1, X_2, X_3); s_3(X_1, X_4); s_4(X_3, X_6); s_5(X_4, X_5, X_6);$$
$$s_6(X_4, X_7); s_7(X_5, X_8); s_8(X_6, X_9); s_9(X_2, X_3, X_{10}, X_{11}).$$

Figure 10 shows the corresponding hypergraph $\mathcal{H}_{hg}$, which is clearly cyclic. The minimal hinges of $\mathcal{H}_{hg}$ are $H_1 = \{s_1, s_2, s_9\}$, $H_2 = \{s_2, s_3, s_4, s_5\}$, $H_3 = \{s_5, s_6\}$, $H_4 = \{s_5, s_7\}$, $H_5 = \{s_5, s_8\}$, $H_6 = \{s_3, s_6\}$, and $H_7 = \{s_4, s_8\}$, where s_i denotes the set of variables occurring in the scope s_i, for $1 \leq i \leq 9$.

Since the cardinality of the largest minimal hinge of $\mathcal{H}_{hg}$ (hinge H_2) is 4, it follows that the HINGE width of $\mathcal{H}_{hg}$ is 4. Figure 10.b shows a HINGE decomposition of $\mathcal{H}_{hg}$.

4.5 *Hinge Decomposition + Tree Clustering (short:* HINGE$^{\text{TCLUSTER}}$*)* [18]

It has been observed [18] that the minimal hinges of a hypergraph can be further decomposed by means of the triangulation technique of the above-described tree-clustering method. This leads to a new decomposition method, that we call HINGE$^{\text{TCLUSTER}}$, which combines HINGE and TCLUSTER and can be formally defined as follows. Let $T = (N, E)$ be a hinge tree of a hypergraph $\mathcal{H}$. For any hinge $H \in N$, let $w(H)$ be the minimum of the cardinality of H and the TCLUSTER width of the hypergraph $(var(H), H)$. The HINGE$^{\text{TCLUSTER}}$ width of $\mathcal{H}$ with respect to T is $\max_{H \in N}\{w(H)\}$. A HINGE$^{\text{TCLUSTER}}$ decomposition of $\mathcal{H}$ with respect to T is an acyclic hypergraph $\mathcal{H}'$ having the same set of vertices as $\mathcal{H}$, and whose set of edges is obtained from T and $\mathcal{H}$ as follows. For each hinge $H \in N$, if $w(H) = |H|$, then $\mathcal{H}'$ contains an edge $var(H)$; otherwise, $\mathcal{H}'$ contains the edges of any TCLUSTER decomposition of the (sub)hypergraph $(var(H), H)$ having width $w(H)$.

The HINGE$^{\text{TCLUSTER}}$ *width* of $\mathcal{H}$ is the minimum HINGE$^{\text{TCLUSTER}}$ width over all its HINGE$^{\text{TCLUSTER}}$ decompositions.

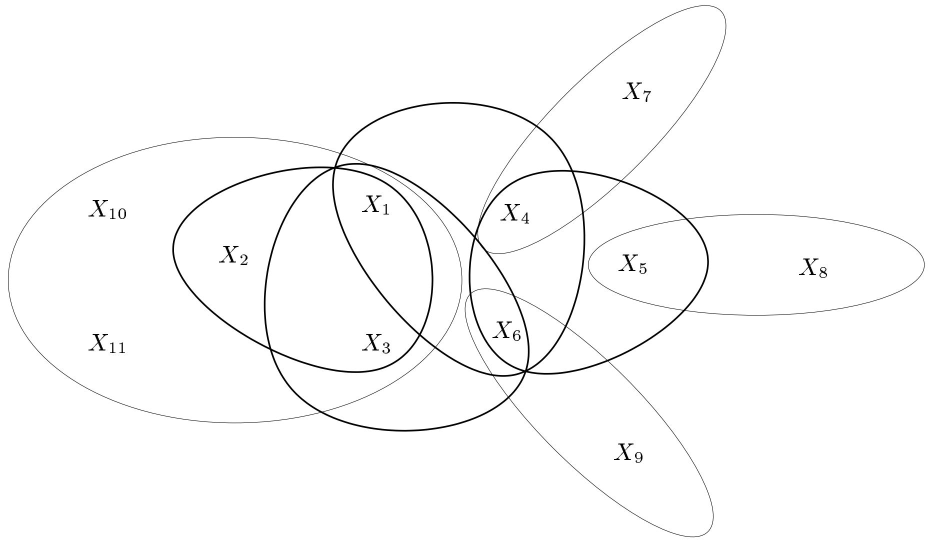


Figure 11. A $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition of hypergraph $\mathcal{H}_{hg}$ in Example 6

Example 7 Consider again the constraint scopes of Example 6 and the hinge-tree decomposition for the hypergraph $\mathcal{H}_{hg}$ shown in Figure 10.b. From this hinge-tree decomposition, we construct a $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition $\mathcal{H}'_{hg}$ of $\mathcal{H}_{hg}$.

Consider the sub-hypergraph $(var(H_1), H_1)$ corresponding to the minimal hinge H_1 occurring in this hinge-tree decomposition. The primal graph of the hypergraph $(var(H_1), H_1)$ is a clique containing the vertices X_1, X_2, X_3, X_{10}, and X_{11}, thus it is easy to see that the $\texttt{TCLUSTER}$ width of this hypergraph is 5. However, the hinge H_1 contains three edges, hence we get $w(H_1) = 3$, and the $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition $\mathcal{H}'_{hg}$ contains the edge $\{X_1, X_2, X_3, X_{10}, X_{11}\}$ with all the variables occurring in H_1.

A different situation concerns the sub-hypergraph $(var(H_2), H_2)$ corresponding to the minimal hinge H_2. This hypergraph is identical to hypergraph $\mathcal{H}_{tc}$ in Example 4. We observed that $\mathcal{H}_{tc}$ has $\texttt{TCLUSTER}$ width 3, which is smaller than $|H_2| = 4$, and hence $w(H_2) = 3$ holds. This means that, in this case, it is convenient to further decompose $(var(H_2), H_2)$ using the $\texttt{TCLUSTER}$ decomposition method, and the $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition $\mathcal{H}'_{hg}$ contains all the edges belonging to the $\texttt{TCLUSTER}$ decomposition of $\mathcal{H}_{tc} = (var(H_2), H_2)$ shown in Figure 7.

Similarly, for $i \in \{4, 5, 6\}$, the sub-hypergraphs $(var(H_i), H_i)$ corresponding to the other hinges occurring in the hinge-tree decomposition at hand are acyclic hypergraphs. Therefore, $w(H_i) = 1$ holds, because the $\texttt{TCLUSTER}$ width of acyclic hypergraphs is 1.

The resulting $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition $\mathcal{H}'_{hg}$ of $\mathcal{H}_{hg}$ is the acyclic hypergraph shown in Figure 11. The thickest edges in this figure come from the $\texttt{TCLUSTER}$ decomposition of $(var(H_2), H_2)$. Recall that both $w(H_1)$ and $w(H_2)$ are 3, which is the maximum value over the hinges occurring in the given $\texttt{HINGE}$ decomposition of $\mathcal{H}_{hg}$. Thus, the width of $\mathcal{H}'_{hg}$ is 3, and it is easy to verify that there is no other $\texttt{HINGE}^{\texttt{TCLUSTER}}$ decomposition having smaller width. It follows that the $\texttt{HINGE}^{\texttt{TCLUSTER}}$ width of $\mathcal{H}_{hg}$ is 3.

4.6 Cycle Cutset (short: `CUTSET`*)* [7]

A *cycle cutset* of a hypergraph $\mathcal{H}$ is a set $S \subseteq var(\mathcal{H})$ such that the subgraph of the primal graph of $\mathcal{H}$ (vertex-)induced by $var(\mathcal{H}) - S$ is acyclic. That is, after deleting the vertices in S, the primal graph of $\mathcal{H}$ becomes acyclic. The `CUTSET` width of $\mathcal{H}$ is 1 if $\mathcal{H}$ is acyclic; otherwise, it is the minimum cardinality over all its possible cycle cutsets.

Example 8 The hypergraph $\mathcal{H}_b$ shown in Figure 5.a has `CUTSET` width 4. Indeed, $\{G, C, D, E\}$ is a cycle cutset of this hypergraph, and any smaller set of vertices does not allow to break all the cycles in its primal graph (see Figure 5.b). As another example, consider the hypergraph $\mathcal{H}_{tc}$ shown in Figure 7. The `CUTSET` width of $\mathcal{H}_{tc}$ is 2, because there is no cycle cutset of cardinality 1, while there are cycle cutsets of cardinality 2, e.g., the set $\{X_1, X_4\}$.

4.7 Cycle Hypercutset (short: `HYPERCUTSET`*)*

This is a simple modification of the `CUTSET` method where the cutset is composed of (hyper)edges rather than vertices of the given hypergraph. A *cycle hypercutset* of a hypergraph $\mathcal{H}$ is a set $\hat{H} \subseteq edges(\mathcal{H})$ such that the subhypergraph of $\mathcal{H}$ induced by $var(\mathcal{H}) - var(\hat{H})$ is acyclic. The `HYPERCUTSET` width of $\mathcal{H}$ is 1 if $\mathcal{H}$ is acyclic; otherwise, it is the minimum cardinality over all its possible cycle hypercutsets.

Example 9 The hypergraph $\mathcal{H}_b$ shown in Figure 5.a has `HYPERCUTSET` width 2. Indeed, the set containing the two edges $\{F, G, C\}$ and $\{C, D, E\}$ is a hypercutset of this hypergraph, as deleting these edges it becomes acyclic. Moreover, by deleting any single edge, we cannot achieve acyclicity. Instead, the hypergraph $\mathcal{H}_{hg}$ shown in Figure 10 has `HYPERCUTSET` width 1. Indeed, e.g., by just deleting from $\mathcal{H}_{hg}$ the edge $\{X_4, X_5, X_6\}$ we get an acyclic hypergraph.

4.8 Solving CSPs using decomposition methods

For each of the above decomposition methods D, it was shown (or it is easy to see) that, for any fixed k, given a CSP instance I, deciding whether a hypergraph $\mathcal{H}_I$ has D-width$(\mathcal{H}_I)$ at most k is feasible in polynomial time and that solving CSPs whose associated hypergraph is of width at most k can be done in polynomial time. In particular, D consists of two phases. Given a CSP instance I,

(1) the (k-bounded) D width w of $\mathcal{H}_I$ along with a corresponding decomposition is computed;
(2) exploiting this decomposition, I is then solved in time $O(n^{w+1} \log n)$, where n is the size of I plus the size of the given decomposition (for most methods this phase consists of the solution of an acyclic CSP instance equivalent to I).

Actually, for these methods it is always possible to give the decompositions in suitable forms without redundancies. Thus, the cost above reduces to $O(\| I \|^{w+1} \log \|I\|)$, i.e., it depends only on the CSP instance, and does not depend on the size of the decomposition. For a detailed analysis, see Section 5, where we study the complexity of evaluating bounded-width CSPs according to a new decomposition method, based on hypertree decompositions [16].

The cost of the first phase is independent on the constraint relations of I; in fact, it is $O(\|\mathcal{H}_I\|^{c_1 k + c_2})$, where $\|\mathcal{H}_I\|$ is the size of the hypergraph $\mathcal{H}_I$, and c_1, c_2 are two constants relative to the method D ($0 \leq c_1, c_2 \leq 3$ for the methods above). As usual, the size of hypergraph $\mathcal{H}_I$ is defined as the number of bits needed for encoding all the edges of $\mathcal{H}_I$ as lists of variables. Clearly, the size of $\mathcal{H}_I$ is always smaller than than $\|I\|$, because the encoding of I includes the encoding of its constraint relations, too. Observe also that computing the D-width w of a hypergraph in general (i.e., without the constant bound $w \leq k$) is NP-hard for most methods, while it is feasible in polynomial time for `HINGE`, and even in linear time for `BICOMP`.

Remark 10 The above complexity bounds, given as functions of the total size of the CSP instance, are appropriate for all considered decomposition methods for *general* CSP instances. Of course, if one considers some restricted cases, e.g., CSP instances with a fixed constant domain size, some finer analysis may be useful. In fact, by exploiting additional information, more accurate complexity bounds may be found in order to choose a method that is better tailored for such a special case.

4.9 Freuder width and adaptive width

Further interesting methods, that do not explicitly generalize acyclic hypergraphs, are based on a different notion of width, that we call *Freuder width* [10,11]. If $\sqsubset$ is a total ordering of the vertices of a graph $G = (V, E)$, then the $\sqsubset$-width of G is defined by $w_{\sqsubset}(G) = \max_{v \in V} |\{ \{v, w\} \in E \;\text{ s.t. }\; w \sqsubset v\}|$. The Freuder width of G is the minimum of all $\sqsubset$-widths over all possible total orderings $\sqsubset$ of V. For each fixed constant k, it can be determined in polynomial time whether a graph is of Freuder width k. The graph G_1 shown in Figure 2 has Freuder width 3. This width can be obtained taking the ordering $b \sqsubset d \sqsubset e \sqsubset a \sqsubset g \sqsubset h \sqsubset c \sqsubset f$. Freuder observed that many naturally arising CSPs have a very low width [10]. He showed that a CSP of width k whose relations enjoy the property of *k'-consistency*, where $k' > k$, can be solved in a backtrack-free manner, and thus in polynomial time [10,11]. Clearly, since the consistency condition on the constraint relations must be satisfied, we cannot define a purely structural decomposition method based on Freuder width. In fact, the following theorem pinpoints that the structural property of bounded Freuder width does not make the CSP problem any easier.

Theorem 11 *Constraint solvability remains NP-complete even if restricted to CSPs whose primal graph has Freuder width bounded by 4.*

PROOF. 3COL remains NP-complete even for graphs of degree 4 (cf. [13]). Such graphs, however, have width at most 4. By the encoding of 3COL as a CSP, as given in Section 2, the theorem follows. $\square$

One can try to enforce a suitable level of consistency on the constraint relations of a given CSP instance. However, the algorithms used to increase the level of consistency in the data also increase the Freuder width of the instance [25,8]. Of course, one can think of devising a more powerful procedure to find an equivalent CSP instance whose Freuder width stays below a fixed bound. However, from the above theorem, if P $\neq$ NP, such a procedure cannot run in polynomial time.

Dechter and Pearl subsequently introduced the notion of *induced width* w^* [8], which is – roughly – the smallest Freuder width k of any graph G' obtained by triangulation methods from the primal graph G of a CSP such that G' ensures $k+1$-consistency. Graphs having induced width at most k can be also characterized as *partial k-trees* [12] or, equivalently, as graphs having treewidth at most k [1]. It follows that, for fixed k, checking whether $w^* \leq k$ is feasible in linear time [5]. If w^* is bounded by a constant, a CSP is solvable in polynomial time. The approach to CSPs based on w^* is referred to as the w^*-Tractability method [7]. Note that this method is implicitly based on hypergraph acyclicity, given that the used triangulation methods enforce chordality of the resulting graph G' and thus acyclicity of the corresponding hypergraph. It was noted [9,7] that, for any cyclic CSP instance I, $\mathsf{TCLUSTER}$ width$(\mathcal{H}_I) = w^*(\mathcal{H}_I) + 1$.

5 Hypertree Decompositions of CSPs

A new class of tractable conjunctive database queries, which generalizes the class of acyclic queries, has recently been identified [16]. This is the class of queries having a bounded-width hypertree decomposition [16]. Deciding whether a given query has this property is feasible in polynomial time and even highly parallelizable. In this section we first adapt the notion of hypertree decomposition, previously defined in the database context, to the general framework of hypergraphs. Then, we show how to employ this notion in order to define a new CSP decomposition method we will refer to as HYPERTREE.

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 12 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 $h \in \lambda(p)$. 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 $c = hw(\mathcal{H})$.

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 [16].

Remark 13 From any hypertree decomposition *HD* of $\mathcal{H}$, we can easily compute a complete hypertree decomposition of $\mathcal{H}$ having the same width. For any "missing" edge h, choose a vertex q of T such that $var(h) \subseteq \chi(q)$ (such a vertex must exist by Condition 1), and create a new vertex p as a child of q with $\lambda(p) = h$ and $\chi(p) = var(h)$. Assuming the use of suitable data structures, this computation can be done in $O(||\mathcal{H}|| \cdot ||HD||)$ time, where $||HD||$ denotes the size of a hypertree decomposition, i.e., the number of bits needed for encoding HD (that is, for encoding the rooted tree of HD and, for each vertex v of this tree, the labelings χ and λ for v, encoded as a list of variables and a list of edge identifiers, respectively).

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.

Example 14 Figure 12 shows a hypertree decomposition of width 2 of the hypergraph $\mathcal{H}_{cp}$ of the crossword puzzle in Example 2 (see Figure 4). Each box b in this figure represents a vertex v of the hypertree decomposition of $\mathcal{H}_{cp}$. The two sets depicted in the box b are the labelings $\chi(v)$ and $\lambda(v)$. The hypergraph $\mathcal{H}_{cp}$ is clearly cyclic, therefore $hw(\mathcal{H}_{cp}) > 1$ (as only acyclic hypergraphs have hypertree width 1). Thus, it follows that the HYPERTREE width of $\mathcal{H}_{cp}$ is 2.

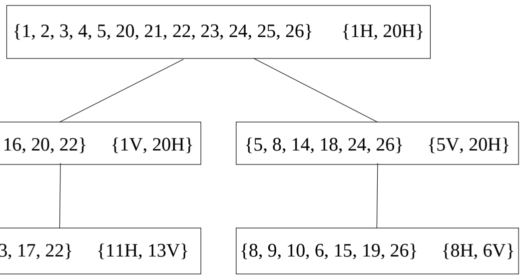


Figure 12. A hypertree decomposition of width 2 of hypergraph $\mathcal{H}_{cp}$ in Example 2

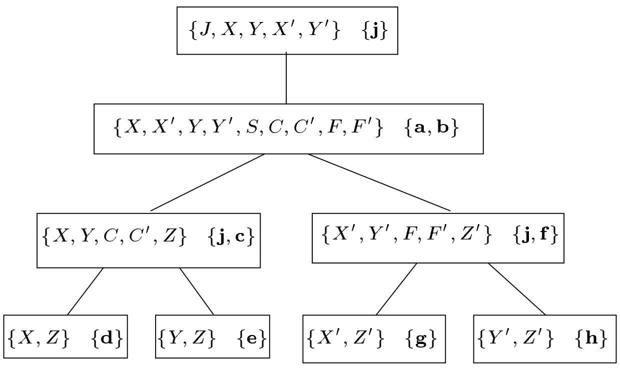


Figure 13. A 2-width hypertree decomposition of $\mathcal{H}_1$

Example 15 Consider the following constraint scopes:

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

Let $\mathcal{H}_1$ be their corresponding hypergraph. Since $\mathcal{H}_1$ is cyclic, $hw(\mathcal{H}_1) > 1$ holds. Figure 13 shows a (complete) hypertree decomposition of $\mathcal{H}_1$ having width 2, hence $hw(\mathcal{H}_1) = 2$.

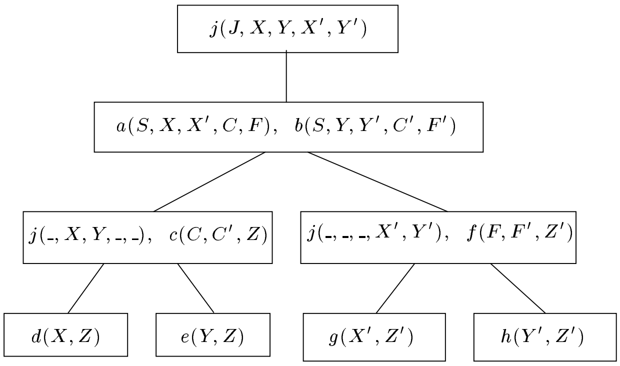


Figure 14. Hyperedge representation of hypertree decomposition HD_5

In order to help the intuition of what a hypertree decomposition is, we also present an alternative representation, called *hyperedge representation*. (Also, "atom representation," in the conjunctive-queries framework.) Figure 14 shows the hyperedge representation of the hypertree decomposition HD_1 of $\mathcal{H}_1$. 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 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 $\mathcal{H}_1$ 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 12).

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. From the results in [16], it follows that k-bounded hypertree width is efficiently decidable, and that a hypertree decomposition of width k can be efficiently computed (if any).

Example 16 Consider again the CSP instance I_{hg} in Example 6. Figure 15 shows the hyperedge representation of a width 2 hypertree decomposition of its hypergraph $\mathcal{H}_{hg}$. It follows that $hw(\mathcal{H}_{hg}) = 2$, because $\mathcal{H}_{hg}$ is cyclic. Thus, I_{hg} has 2-bounded `HYPERTREE` width and, more generally, k-bounded `HYPERTREE` width for any integer $k > 1$.

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

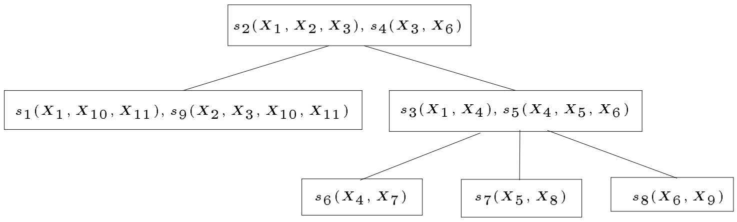


Figure 15. A hypertree decomposition of hypergraph $\mathcal{H}_{hg}$ in Example 6

$W \subseteq var(\mathcal{H})$ 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.

Definition 17 ([16]) 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. $var(\lambda(s)) \cap \chi(r) \;\subseteq\; \chi(s)$.

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

Proposition 18 ([16]) *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.

The fact that no redundancies occur in hypertree decompositions in normal form allows us to give a precise bound on the number of vertices in such hypertree decompositions.

Lemma 19 *Let $HD = (T, \chi, \lambda)$ be a hypertree decomposition in normal form of a hypergraph $\mathcal{H}$. Moreover, let n be the number of vertices of the decomposition tree T, and m the number of strongly covered edges of $\mathcal{H}$ in HD. Then, $n \leq m$ holds.*

PROOF. Let s be some vertex in T. We say that a variable $X \in \chi(s)$ (respectively, an edge $H \subseteq \chi(s)$)) is "first covered" in s if $X \notin \chi(vertices(T) - vertices(T_p))$

(resp., $H \not\subseteq \chi(vertices(T) - vertices(T_p))$); otherwise, X (resp., H) is said to be "previously covered." By Condition 2 of Definition 17 and by Condition 2 of Def. 12, it follows that, for any vertex p of T, there exists at least a variable X in $var(\mathcal{H})$ which is "first covered" in p. Since $X \in \chi(p)$, from Condition 3 of Definition 12, it follows that there is an edge H of $\mathcal{H}$ such that $X \in H$ and $H \in \lambda(p)$. Moreover, from Condition 4 of Definition 12, it follows that every variable belonging to H and not covered in some vertex in $vertices(T) - vertices(T_p)$ must be first covered in p, and belongs to $\chi(p)$.

Moreover, since HD is in normal form, it satisfies Condition 3 of Definition 17. (i.e., $var(\lambda(s)) \cap \chi(r) \subseteq \chi(s)$). It follows that, in fact, any previously-covered variable Y belonging to H must belong to $\chi(p)$. Indeed, since the variable X was not previously covered, the edge H cannot be previously covered, and thus there exists some vertex p' in the subtree T_p such that $H \subseteq \chi(p')$, in order to fulfill Condition 1 of Definition 12. Assume that the variable $Y \in H$ does not belong to $\chi(p)$. Since H is strongly covered by p', $Y \in \chi(p')$. Moreover, by the choice of Y, this variable is previously covered with respect to p. It follows that Y violates the connectedness condition, a contradiction.

Thus, all the variables in H belong to $\chi(p)$. Recall that $H \in \lambda(p)$, too. It follows that at least one edge of $\mathcal{H}$ is first covered in vertex p and strongly covered by p, and, in general, that each vertex in T first and strongly covers some edge of $\mathcal{H}$. This entails that the cardinality of the set of vertices in the decomposition tree T of *HD* is less than or equal to the number m of the strongly covered edges in the normal form hypertree decomposition *HD* of $\mathcal{H}$. □

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 [17]. 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.

We next show that any CSP instance I is efficiently solvable, given a k-bounded complete hypertree-decomposition *HD* of $\mathcal{H}_I$. To this end, we define an acyclic CSP instance which is equivalent to I and whose size is polynomially bounded by the size of I.

For each vertex p of the decomposition *HD*, we define a new constraint scope whose associated constraint relation is the projection on $\chi(p)$ of the join of the relations in $\lambda(p)$. This way, we obtain a join-tree *JT* of an acyclic hypergraph $\mathcal{H}^*$. $\mathcal{H}^*$ corresponds to a new CSP instance I^* over a set of constraint relations of size $O(n^k)$, where n is the input size (i.e., $n = \| I \|$) and k is the width of the hypertree decomposition HD. By construction, I^* is an acyclic CSP, and we can easily show that it is equivalent to the input CSP instance I. Thus, all the efficient techniques available for acyclic CSP instances [9,7], or for any problem equivalent to CSP [26,21,15], can be employed for the evaluation of I^*, and hence of I.

Remark 20 According to our definition, any hypertree is a labeled *rooted* tree. The rooting is necessary for technical reasons concerning the notion of hypertree decomposition only, but has no impact on the actual evaluation of the given CSP instance. In fact, the above discussion describes how to compute from a hypertree decomposition and a CSP instance I a join tree JT of an acyclic instance I^* that is equivalent to I. This construction does not use the fact that the hypertree is rooted. Moreover, note that the acyclic instance I^* can be evaluated rooting the join tree JT at any vertex.

The following theorem provides a detailed analysis of the complexity of evaluating a CSP given a hypertree decomposition for it.

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

PROOF. Let I be a CSP instance and $HD' = (T', \chi', \lambda')$ a k-width hypertree decomposition of $\mathcal{H}_I$ in normal form. We proceed as follows.

Step 1 We compute from HD' a complete hypertree decomposition $HD = (T, \chi, \lambda)$ of $\mathcal{H}_I$.
Step 2 We compute from HD and I an acyclic instance I^* equivalent to I, as described above.
Step 3 We evaluate the acyclic instance I^* employing any efficient technique for solving acyclic CSPs.

Let m be the number of edges of $\mathcal{H}_I$. The following statements hold:

Claim 1. *The decomposition tree of the complete hypertree decomposition HD has at most m vertices.* This immediately follows from the construction of HD and from Lemma 19, since in Step 1 above we just add to the decomposition tree T those edges of $\mathcal{H}_I$ that are not strongly covered in HD'.

Claim 2. *Step 1 is feasible in $O(||\mathcal{H}_I||^2)$.* As observed in Remark 13, this computation takes $O(||HD'|| \cdot ||\mathcal{H}_I||)$ time. From Lemma 19, it easily follows that $||HD'||$ is $O(k\ ||\mathcal{H}_I||) = O(||\mathcal{H}_I||)$, because the number of vertices in T' is at most the number of edges of $\mathcal{H}_I$, and the number of edge-labels of each vertex of T' is bounded by the constant k.

Claim 3. $||I^*||= O(||I||^k)$, and computing I^* from I takes time $O(||I||^k)$. Consider a constraint $C = (S_c, r_c)$ in the acyclic instance I^*. As described above, the relation r_c is obtained as the natural join of at most k relations occurring in the input instance I. One of these input relations, say r_i, – in fact, its constraint scope S_i –

is covered by some vertex p in the decomposition tree of *HD* which corresponds to C in the acyclic instance I^*. (In particular, its scope S_i corresponds to some edge $h_i \in edges(\mathcal{H}_I)$ such that $h_i \in \lambda(p)$, and $h_i \subseteq \chi(p)$.) Let r_{max} be the constraint relation having the maximum size $\|r_{max}\|$ over all the constraint relations occurring in the input instance I. Then, $\|r_c\| \leq (\|r_{max}\|^{k-1} \cdot \|r_i\|)$. Recall that the instance I^* has at most m constraints. Considering all the constraints in I, we get the following upper bound for the size of the whole CSP I^*:

$$\|I^*\| \leq (\|r_{max}\|^{k-1} \cdot \|r_1\| + \cdots + \|r_{max}\|^{k-1} \cdot \|r_m\|)$$

and hence

$$\|I^*\| \leq \|r_{max}\|^{k-1} \cdot (\|r_1\| + \cdots + \|r_m\|) \leq \|r_{max}\|^{k-1} \cdot \|I\|$$

It follows that $\| I^* \| \leq \| I \|^k$. Moreover, the effective *computation* of I^* from I takes time $O(\|I\|^k)$. Indeed, computing the natural join of two relations r_1 and r_2 takes time $O(\|r_1\| \cdot \|r_2\|)$, which is exactly the same bound that we have for the size of the result of this join operation. Thus, by applying the same line of reasoning as used for the space bound, we get that the computation of the acyclic instance I^* is feasible in $O(\|I\|^k)$ time.

From claims 1–3 and from the well-known $O(m \cdot \|I^*\| \cdot \log \|I^*\|)$ complexity of evaluating the acyclic CSP I^* (see. e.g., [7,9]), it follows that the overall cost of this evaluation procedure is $O(\|I\| \cdot \|I\|^k \cdot \log \|I\|^k) + O(\|\mathcal{H}_I\|^2) = O(\|I\|^{k+1} \cdot \log \|I\|)$, because k is fixed, $\|\mathcal{H}_I\| \leq \|I\|$, and $k \geq 1$. $\square$

It is worthwhile noting that the crucial difference between the HYPERTREE method and the TCLUSTER method is the objective function to be minimized in order to obtain the most convenient acyclic decomposition of a given CSP instance. The HYPERTREE method minimizes the number of hyperedges of $\mathcal{H}_I$ associated to any vertex of the acyclic equivalent instance, thus exploiting the fact that one hyperedge "covers" many variables at once. The TCLUSTER method minimizes the number of variables occurring in any vertex of the equivalent acyclic instance, as evidenced by the following example.

Example 22 For any $m > 0$, let $T(m)$ be the hypergraph having the $m + 3$ hyperedges $\{h_1, h_2, h_3, e_1, e_2, \ldots, e_m\}$ defined as follows:

- $h_1 = \{X_1, \ldots, X_m, Y_1, \ldots, Y_m, A\}$;
- $h_2 = \{Y_1, \ldots, Y_m, Z_1, \ldots, Z_m, B\}$;
- $h_3 = \{Z_1, \ldots, Z_m, X_1, \ldots, X_m, C\}$;
- $e_i = \{X_i, Y_i, Z_i\}$, $\forall 1 \leq i \leq m$.

The TCLUSTER width of $T(m)$ is $3m$, because its primal graph is chordal and its maximal clique $C = \{X_1, \ldots, X_m, Y_1, \ldots, Y_m, Z_1, \ldots, Z_m\}$ has cardinality $3m$. In fact, according to the TCLUSTER method, we have to solve a subproblem involving every hyperedge e_i ($1 \leq i \leq m$).

On the other hand, for any $m > 0$, the HYPERTREE width of $T(m)$ is 2. It is worthwhile noting that the number of variables occurring in the largest vertex of this decomposition is $3m + 2$. Hence, the equivalent acyclic instance we obtain according to HYPERTREE is not "optimal" according to the TCLUSTER method, because its associated primal graph has a clique of cardinality $3m + 2$. Nevertheless, the constraint relation associated to this vertex is computable very easily as the join of the constraint relations r_1 and r_2 corresponding to h_1 and h_2, respectively.

A simple way to get decomposition methods which in some way exploit the power of hyperedges is using the dual graph associated to a CSP. We give a detailed analysis of these approaches and of their relationships with the HYPERTREE method in Section 9. It turns out that even such methods do not exploit the full power of hyperedges, and are less general then HYPERTREE, according to a strong notion of generalization, formally defined in the next section.

6 Comparison Criteria

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 there exists $\delta \geq 0$ such that, for every $k > 0$, $C(D_1, k) \subseteq C(D_2, k + \delta)$. Thus, $D_1 \preceq D_2$ implies that every class of CSP instances which is tractable according to D_1 is also tractable according to D_2.

Note that the constant δ above allows us to get rid of small differences among tractability classes that should be irrelevant in the comparison. E.g., it is known (see discussion in Section 4.9) that TCLUSTER and TREEWIDTH are equivalent methods and one would expect TCLUSTER to generalize TREEWIDTH (as well as vice versa). However, for any $k > 1$, $C(\mathtt{TREEWIDTH}, k) \not\subseteq C(\mathtt{TCLUSTER}, k)$, because the treewidth is defined through the cardinality of the vertex-labeling minus one. Rather, $C(\mathtt{TREEWIDTH}, k) \subseteq C(\mathtt{TCLUSTER}, k+1)$ holds. Thus, by taking $\delta = 1$, we easily get $\mathtt{TREEWIDTH} \preceq \mathtt{TCLUSTER}$.

$\underline{D_1 \rhd D_2}$ (D_1 *beats* D_2) if there exists an integer k such that, for every $m,\ C(D_1, k) \not\subseteq C(D_2, m)$. 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 every $j \geq 0$.

Intuitively, $D_1 \rhd D_2$ means that, at least on some class of CSP instances, D_1 outperforms D_2 with respect to tractability, because these instances are tractable according to D_1, but not according to D_2. For such classes, using D_1 is thus better than using 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.

This means that D_2 is really the more powerful method, given that, whenever D_1 guarantees polynomial runtime for constraint solving, then also D_2 guarantees tractable constraint solving, but there are classes of constraints that can be solved in polynomial time by using D_2 but are not tractable according to 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. Note that, on the other hand, $\prec\!\!\prec$ is transitive and antisymmetric, but not reflexive.

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 incomparable with respect to the relations $\preceq$ and $\prec\!\!\prec$, too.

7 Comparison Results

In this section we present a complete comparison of the decomposition methods described in Section 4, according to the above criteria. Figure 1 (reproduced here as Figure 16 with the acronyms of decomposition methods for the reader's convenience) shows a representation of the hierarchy of decomposition methods determined by the $\prec\!\!\prec$ relation. Each element of the hierarchy represents one decomposition method, apart from that containing *Tree Clustering*, w^*, and *Treewidth* which are grouped together because they are $\preceq$-equivalent as easily follows from the observations in Section 4.

Theorem 23 *For each pair D_1 and D_2 of decompositions methods represented in Figure 16, the following holds:*

- *There is a directed path from D_1 to D_2 if and only if $D_1 \prec\!\!\prec D_2$, i.e., if and only if D_2 strongly generalizes D_1.*
- *D_1 and D_2 are not linked by any directed path if and only if they are strongly incomparable.*

Hence, Fig. 16 gives a complete picture of the relationships holding among the different methods.

The following lemmas, together with the transitivity of the relations defined in Section 6, prove Theorem 23.

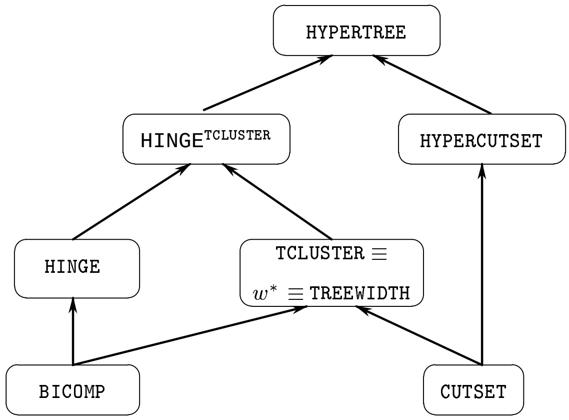


Figure 16. Constraint Tractability Hierarchy

For any $n > 2$ and $m > 0$, let *Circle*(n, m) be the hypergraph having n edges $\{h_1, \ldots, h_n\}$ defined as follows:

- $h_i = \{X_i^1, \ldots, X_i^m, X_{i+1}^1, \ldots, X_{i+1}^m\}\ \forall 1 \le i \le n-1$;
- $h_n = \{X_n^1, \ldots, X_n^m, X_1^1, \ldots, X_1^m\}$.

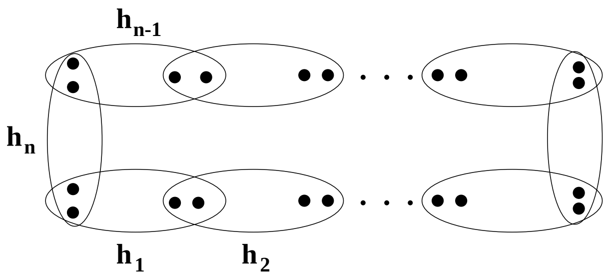


Figure 17. The hypergraph *Circle*$(n, 2)$

Figure 17 shows the hypergraph *Circle*$(n, 2)$, for some $n > 8$. For $m = 1$, *Circle*$(n, 1)$ is a graph consisting of a simple cycle with n edges (like a circle). Note that, for any $n > 2$ and $m > 0$, *Circle*(n, m) has hypertree width 2. A width 2 hypertree decomposition of *Circle*(n, m) is shown in Figure 18. It follows that the (infinite) class of hypergraphs $\bigcup_{n>2,m>0}\{$*Circle*$(n, m)\}$ is included in the tractability class $C(\mathtt{HYPERTREE}, 2)$.

For any $n > 0$, let $triangles(n)$ be the graph (V, E) defined as follows. The set of vertices V contains $2n+1$ vertices $p_1, \ldots, p_{2n+1}$. For each even index i, $2 \le i \le 2n$, $\{p_i, p_{i-1}\}$, $\{p_i, p_{i+1}\}$, and $\{p_{i-1}, p_{i+1}\}$ are edges in E. No other edge belongs to E. Figure 19 shows the graph $triangles(n)$. The HYPERTREE width of $triangles(n)$ is 2. Indeed, a hypertree $\langle T, \chi, \lambda\rangle$, where T is a simple chain of n vertices $v_1, \ldots, v_n$ and, for each v_i ($1 \le i \le n$), $\chi(v_i) = \{p_{2i-1}, p_{2i}, p_{2i+1}\}$ and $\lambda(v_i)$ contains the two edges $\{p_{2i-1}, p_{2i}\}$ and $\{p_{2i}, p_{2i+1}\}$, is a width 2 HYPERTREE decomposition of $triangles(n)$.

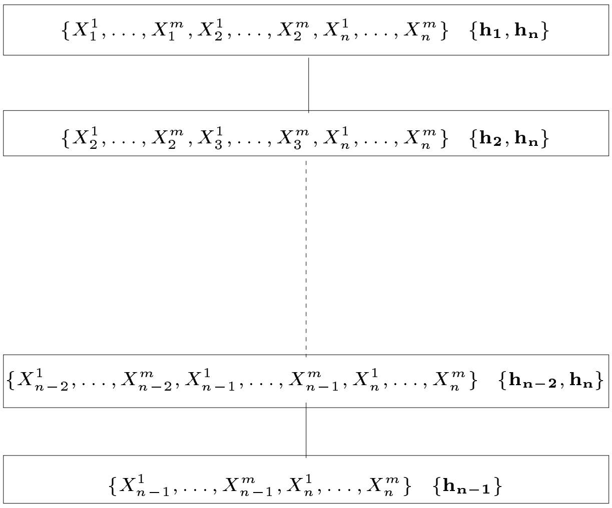


Figure 18. 2-width hypertree decomposition of *Circle*(n, m)

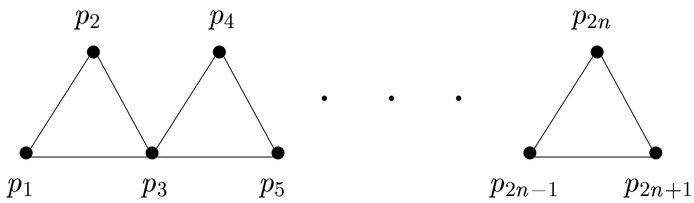


Figure 19. The graph $triangles(n)$

For any $n > 0$, let *book*(n) be a graph having $2n + 2$ vertices and $3n + 1$ edges that form n squares (pages of the book) having exactly one common edge $\{X, Y\}$. It is easy to see that the HYPERTREE width of *book*(n) is 2. Figure 20 shows the graph *book*(4).

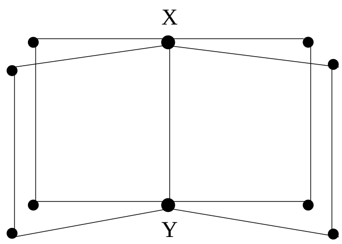


Figure 20. The graph *book*(4)

Lemma 24 $\texttt{CUTSET} \prec\!\!\prec \texttt{HYPERCUTSET}$.

PROOF. HYPERCUTSET clearly generalizes CUTSET. Moreover, $\texttt{HYPERCUTSET} \rhd \texttt{CUTSET}$. Indeed, $\bigcup_{n>2,m>0} \{\textit{Circle}(n, m)\} \not\subseteq C(\texttt{CUTSET}, k)$ holds for any $k > 0$; while, $\bigcup_{n>2,m>0}\{\textit{Circle}(n, m)\} \subseteq C(\texttt{HYPERCUTSET}, 1)$, as deleting any edge of *Circle*(n, m) yields an acyclic hypergraph. $\Box$

Lemma 25 $\texttt{BICOMP} \rhd \texttt{HYPERCUTSET}$.

PROOF. Consider the graph $triangles(n)$ for some $n > 0$. It is easy to see that the HYPERCUTSET width of $triangles(n)$ is $\lceil n/3 \rceil$, while its BICOMP width is 3. Hence, $\bigcup_{n>1}\{triangles(n)\} \subseteq C(\texttt{BICOMP}, 3)$, while, $\bigcup_{n>1}\{triangles(n)\} \not\subseteq C(\texttt{HYPERCUTSET}, k)$ holds for every $k > 0$. □

Lemma 26 BICOMP *and* CUTSET *are strongly incomparable.*

PROOF. (BICOMP $\rhd$ CUTSET.) Follows from Lemma 25 and Lemma 24.

(CUTSET $\rhd$ BICOMP.) Consider the graph *book*(n) for some $n > 0$. The whole graph *book*(n) is biconnected. Thus, its BICOMP width is $2n + 2$. On the other hand, the set $\{X, Y\}$ is a cycle cutset of *book*(n). Thus, $\bigcup_{n>1}\{book(n)\} \subseteq C(\texttt{CUTSET}, 2)$ holds. □

Lemma 27 BICOMP $\prec\!\!\prec$ HINGE.

PROOF. In [18], it was shown that BICOMP$\preceq$ HINGE. Thus, it suffices to prove that HINGE $\rhd$ BICOMP: Consider the graph *book*(n) defined above, for some $n > 0$. As observed above, the BICOMP width of *book*(n) is $2n+2$, while its HINGE width is 4. Indeed, the minimal hinges of *book*(n) correspond to the pages of the book, and each of them has cardinality 4. □

Lemma 28 BICOMP $\prec\!\!\prec$ TCLUSTER.

PROOF. In [7], it was observed that BICOMP$\preceq$ TCLUSTER. (In fact, BICOMP was compared with w^*, which is $\preceq$-equivalent to TCLUSTER.) Furthermore, TCLUSTER$\rhd$ BICOMP follows from CUTSET $\rhd$ BICOMP and from the fact, observed in [7], that TCLUSTER generalizes CUTSET, i.e., CUTSET $\preceq$ TCLUSTER. □

Lemma 29 CUTSET $\prec\!\!\prec$ TCLUSTER.

PROOF. As mentioned above, CUTSET $\preceq$ TCLUSTER [7]. Moreover, TCLUSTER $\rhd$ CUTSET follows from BICOMP $\rhd$ CUTSET and BICOMP $\preceq$ TCLUSTER. □

Lemma 30 CUTSET $\rhd$ HINGE.

PROOF. Every graph in $\bigcup_{n>2}\{Circle(n, 1)\}$ has CUTSET width 1, because deleting any vertex of the graph we get an acyclic graph. However, for any $n > 2$, the degree of cyclicity of *Circle*$(n, 1)$ is n [18]. □

Lemma 31 HINGE *and* TCLUSTER *are strongly incomparable.*

PROOF. ($\mathtt{HINGE} \rhd \mathtt{TCLUSTER}$). Let $S = \{\mathit{Circle}(3, m) \mid m > 1\}$. For any $m > 1$, the primal graph G of *Circle*$(3, m)$ is a clique of $3m$ variables. Thus, G does not need any triangulation, because it is a chordal graph. The TCLUSTER width of *Circle*$(3, m)$ is clearly $3m$; while its HINGE width is 3, because every hypergraph in S has only three (hyper)edges.

($\mathtt{TCLUSTER} \rhd \mathtt{HINGE}$). Follows from $\mathtt{CUTSET} \rhd \mathtt{HINGE}$ and $\mathtt{CUTSET} \preceq \mathtt{TCLUSTER}$. □

Lemma 32 $\mathtt{HINGE} \prec\!\!\prec \mathtt{HINGE}^{\mathtt{TCLUSTER}}$ *and* $\mathtt{TCLUSTER} \prec\!\!\prec \mathtt{HINGE}^{\mathtt{TCLUSTER}}$.

PROOF. It is easy to see that both $\mathtt{HINGE} \preceq \mathtt{HINGE}^{\mathtt{TCLUSTER}}$ and $\mathtt{TCLUSTER} \preceq \mathtt{HINGE}^{\mathtt{TCLUSTER}}$ hold. Furthermore, $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \rhd \mathtt{HINGE}$ follows from $\mathtt{TCLUSTER} \preceq \mathtt{HINGE}^{\mathtt{TCLUSTER}}$ and $\mathtt{TCLUSTER} \rhd \mathtt{HINGE}$; and $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \rhd \mathtt{TCLUSTER}$ follows from $\mathtt{HINGE} \preceq \mathtt{HINGE}^{\mathtt{TCLUSTER}}$ and $\mathtt{HINGE} \rhd \mathtt{TCLUSTER}$. □

Lemma 33 $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \preceq \mathtt{HYPERTREE}$.

PROOF. Let $\mathcal{H}$ be a hypergraph, and $\mathcal{H}'$ be a $\mathtt{HINGE}^{\mathtt{TCLUSTER}}$ decomposition of $\mathcal{H}$ of width k. We show that there exists a hypertree decomposition for $\mathcal{H}$ of width k. We will use as a running example the hypergraph $\mathcal{H}_{hg}$ in Example 6. Figure 11 shows the width 3 $\mathtt{HINGE}^{\mathtt{TCLUSTER}}$ decomposition $\mathcal{H}'_{hg}$ of $\mathcal{H}_{hg}$, described in Example 7.

Recall that, by construction, the $\mathtt{HINGE}^{\mathtt{TCLUSTER}}$ decomposition $\mathcal{H}'$ is an acyclic hypergraph. Note that, in general, $\mathcal{H}'$ is not a reduced hypergraph. For instance, $\mathcal{H}'_{hg}$ is not reduced, as the edge $\{X_1, X_2, X_3\}$, coming from the TCLUSTER decomposition of the hinge H_2, is a subset of $\{X_1, X_2, X_3, X_{10}, X_{11}\}$, which comes from the hinge H_1.

Let $\mathcal{H}''$ be the reduced and acyclic hypergraph obtained from $\mathcal{H}'$ deleting each edge that is a subset of some other edge of the hypergraph. Therefore, e.g., $\mathcal{H}''_{hg}$ contains all the edges of $\mathcal{H}'_{hg}$, but the edge $\{X_1, X_2, X_3\}$.

We partition the edges of $\mathcal{H}''$ into three sets AE, HE, and TE defined as follows.

The set AE contains all edges of $\mathcal{H}''$ that come from the TCLUSTER decomposition of some hinge H_i of $\mathcal{H}$ such that the subgraph $(var(H_i), H_i)$ is acyclic. In the running example, this property holds for hinges H_4, H_5, and H_6. Recall that, in this case, $w(H_i) = 1$ holds, and the decomposition of this hinge is just the acyclic hypergraph $(var(H_i), H_i)$. E.g., for $\mathcal{H}''_{hg}$, AE contains the edges corresponding to the constraint scopes s_5, s_6, s_7, and s_8, i.e., $\{X_4, X_5, X_6\}$, $\{X_4, X_7\}$, $\{X_5, X_8\}$, and $\{X_6, X_9\}$, respectively.

The set TE contains all edges in $edges(\mathcal{H}'') - AE$ that come from the TCLUSTER decomposition of some hinge H_i of $\mathcal{H}$ such that the subgraph $(var(H_i), H_i)$

is cyclic. Since the `TCLUSTER` decomposition of this hypergraph is bounded by k, it follows that each edge in TE contains at most k variables. In our running example, TE contains two edges $\{X_1, X_3, X_6\}$ and $\{X_1, X_4, X_6\}$ that we call te_1 and te_2, respectively.

The set HE contains all those edges in $edges(\mathcal{H}'') - AE - TE$ that come from some hinge of $\mathcal{H}$. Thus, any edge h in HE is the union of at most k edges belonging to some hinge H_i of $\mathcal{H}$. We denote the hinge H_i corresponding to h by $hinge(h)$. In our running example, HE contains one edge $\{X_1, X_2, X_3, X_{10}, X_{11}\}$ that we call he_1 and comes from the hinge $H_1 = \{s_1, s_2, s_9\}$ of $\mathcal{H}_{hg}$. Therefore, $hinge(he_1) = \{s_1, s_2, s_9\}$.

Let JT be a jointree of the acyclic hypergraph $\mathcal{H}''$. Recall that each vertex of the tree JT is an edge of $\mathcal{H}''$ and vice versa, and that the connectedness condition holds, i.e., the subgraph of JT induced by any variable of $\mathcal{H}'$ is connected. Figure 21 shows a jointree of $\mathcal{H}''_{hg}$.

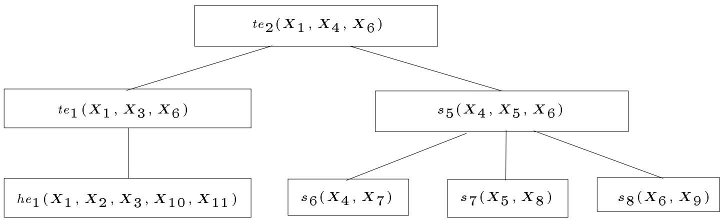


Figure 21. A jointree of the hypergraph $\mathcal{H}''_{hg}$

From JT, we define a hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$, where the tree T has the same shape as JT, and the labelings χ and λ are defined through the following procedure. For each vertex h of JT, denote by p_h the corresponding vertex in the tree T of $\mathcal{H}$.

(1) for each edge h of AE, label the corresponding vertex p_h as follows: $\chi(p_h) = h$ and $\lambda(p_h) = \{h\}$.
(2) for each edge h of HE, label the corresponding vertex p_h as follows: $\chi(p_h) = h$ and $\lambda(p_h) = hinge(h)$.
(3) for each edge h of TE, label the corresponding vertex p_h as follows: $\chi(p_h) = h$ and $\lambda(p_h) = \emptyset$. For the running example, Figure 22 shows the hypertree obtained after these three steps.

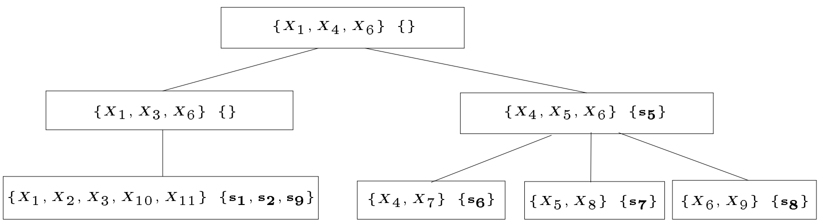


Figure 22. The hypertree for the running example in the proof of Lemma 33 after steps 1, 2, and 3

(4) for each edge $\bar{h}$ of the hypergraph $\mathcal{H}$ such that there is no vertex q in T with $\bar{h} \in \lambda(q)$, choose a vertex h of JT such that $\bar{h} \subseteq h$ and $h \in \mathit{TE}$, and add $\bar{h}$ to the λ labeling of the corresponding vertex p_h in T (i.e., $\lambda(p_h) := \lambda(p_h) \cup \{\bar{h}\}$). In our running example, we add the edge s_3, whose variables are X_1 and X_4, to the λ labeling of the hypertree's root, and the edge s_4, whose variables are X_4 and X_6, to the λ labeling of the left child of the root, as shown in Figure 23.

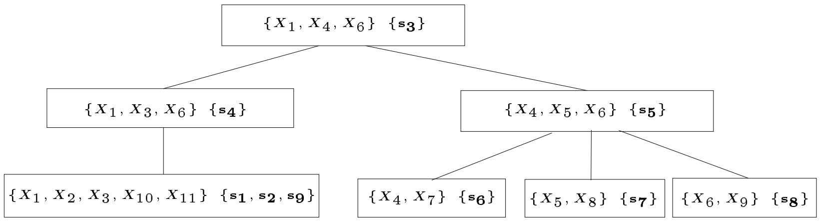


Figure 23. The hypertree for the running example in the proof of Lemma 33 after Step 4

(5) While there is a vertex p in T such that $\chi(p)$ contains a variable X not covered by $\lambda(p)$ (i.e., $X \in \chi(p) - var(\lambda(p))$), proceed as follows.
   (A) Find a path π in T linking p to a vertex q such that (i) $X \in var(\lambda(q))$ and, (ii) $X \notin var(\lambda(s))$ for every vertex s in $\pi - \{q\}$.
   (B) Choose an edge $h \in \lambda(q)$ such that $X \in h$.
   (C) Add h to both $\lambda(s)$ and $\chi(s)$, for every vertex $s \in \pi - \{q\}$ (i.e., $\chi(s) := \chi(s) \cup h$, and $\lambda(s) := \lambda(s) \cup \{h\}$).

   In the running example, the root contains the variable X_6 that is not covered by the edge s_3 (see Figure 23). Then, we choose the path connecting the root and its right child, because X_6 occurs in some edge belonging to its λ labeling, namely in the edge s_5. Thus, we add s_5 to the λ labeling of the root, and the covering of X_6 is done. Similarly, the variable X_1 occurring in the left child of the root is covered by adding to its λ labeling the edge s_1, which occurs in its child. Figure 24 shows the final hypertree obtained for the running example.

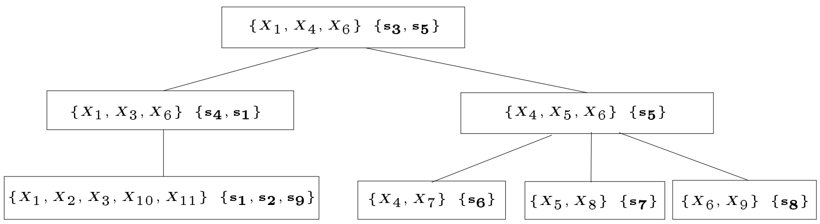


Figure 24. The final hypertree for the running example in the proof of Lemma 33

Note that, after steps 1, 2, and 3, the connectedness condition (i.e., Condition 2 of Definition 12) clearly holds in *HD* because it holds in the jointree JT. However, for any vertex p_h of T corresponding to a vertex $h \in \mathit{TE}$ of JT, Step 3 only provides the χ labeling for p_h. Thus, in Step 4, we select the edges of $\mathcal{H}$ that cover these variables in the vertex p_h of the decomposition *HD*, i.e., we define the λ labeling for p_h.

Since the connectedness condition is preserved in Step 3 above, it is easy to verify that, at the end of the procedure, *HD* is a hypertree decomposition of $\mathcal{H}$. Moreover, its $\mathtt{HYPERTREE}$ width is at most k. Indeed, by the above construction, it follows that for each vertex $h \in HE$, $|\lambda(p_h)| = |hinge(h)| \leq k$, and, for each vertex $h' \in TE$, $|\lambda(p_h)| \leq |h| \leq k$. $\square$

Lemma 34 $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \prec\!\!\prec \mathtt{HYPERTREE}$.

PROOF. From Lemma 33, $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \preceq \mathtt{HYPERTREE}$ holds. We next show that $\mathtt{HYPERTREE} \rhd \mathtt{HINGE}^{\mathtt{TCLUSTER}}$. Consider the cyclic hypergraph *Circle*(n,m), for any $n > 2, m > 0$. This hypergraph has a unique hinge containing all its edges, and therefore its $\mathtt{HINGE}$ width is n. Moreover, its primal graph contains maximal cliques of cardinality at least $2m$, and thus its $\mathtt{TCLUSTER}$ width is at least $2m$. It follows that $\bigcup_{n>2,m>0} \{\textit{Circle}(n,m)\} \not\subseteq C(\mathtt{HINGE}^{\mathtt{TCLUSTER}}, k)$ holds for any $k > 0$. However, for $\mathtt{HYPERTREE}$, $\bigcup_{n>2,m>0}\{\textit{Circle}(n,m)\} \subseteq C(\mathtt{HYPERTREE}, 2)$ holds. (See Figure 18 for a hypertree decomposition of *Circle*(n,m) of width 2.) $\square$

Lemma 35 $\mathtt{HINGE}^{\mathtt{TCLUSTER}}$ *and* $\mathtt{HYPERCUTSET}$ *are strongly incomparable.*

PROOF. $\mathtt{HINGE}^{\mathtt{TCLUSTER}} \rhd \mathtt{HYPERCUTSET}$ follows from $\mathtt{BICOMP} \rhd \mathtt{HYPERCUTSET}$ and $\mathtt{BICOMP} \preceq \mathtt{HINGE}^{\mathtt{TCLUSTER}}$.

$\mathtt{HYPERCUTSET} \rhd \mathtt{HINGE}^{\mathtt{TCLUSTER}}$. Indeed, $\bigcup_{n>2,m>0} \{\textit{Circle}(n,m)\} \not\subseteq C(\mathtt{HINGE}^{\mathtt{TCLUSTER}}, k)$ holds for any $k > 0$; while, $\bigcup_{n>2,m>0}\{\textit{Circle}(n,m)\} \subseteq C(\mathtt{HYPERCUTSET}, 1)$. $\square$

Lemma 36 $\mathtt{HYPERCUTSET} \prec\!\!\prec \mathtt{HYPERTREE}$.

PROOF. We have that $\mathtt{HYPERTREE} \rhd \mathtt{HYPERCUTSET}$ because, from Lemma 25, $\mathtt{BICOMP} \rhd \mathtt{HYPERCUTSET}$, and $\mathtt{BICOMP} \preceq \mathtt{HYPERTREE}$.

We next prove that $\mathtt{HYPERCUTSET} \preceq \mathtt{HYPERTREE}$. Let $\mathcal{H}$ be a hypergraph and $H \subseteq edges(\mathcal{H})$ a cycle hypercutset of $\mathcal{H}$. Let k be the cardinality of H. Let $\mathcal{H}'$ be the subhypergraph of $\mathcal{H}$ induced by $var(\mathcal{H}) - var(H)$, i.e., the hypergraph having an edge $h'(h) = h - var(H)$ for each edge $h \in edges(\mathcal{H})$ such that $h - var(H) \neq \emptyset$. Note that, in general, $\mathcal{H}'$ is not connected. By definition of cycle hypercutset, $\mathcal{H}'$ is acyclic. Thus, there exists a join forest for $\mathcal{H}'$, i.e., a set of jointrees $\mathcal{JT}_1, \ldots, \mathcal{JT}_\ell$ corresponding to the s connected components of $\mathcal{H}'$.

We show that there exists a hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of $\mathcal{H}$ having width $k+1$. The root r of T is labeled by the cycle hypercutset H, i.e., $\lambda(r) = H$, and $\chi(r) = var(H)$. The root r has ℓ children $\{p_1, \ldots, p_\ell\}$ corresponding to the ℓ jointrees $\mathcal{JT}_1, \ldots, \mathcal{JT}_\ell$. In particular, each subtree T_{p_i} rooted at a child p_i ($1 \leq i \leq \ell$) has the same tree shape as the jointree $\mathcal{JT}_i$. Moreover, let q be a vertex of the jointree $\mathcal{JT}_i$, and h be an edge of $\mathcal{H}$ such that $h'(h)$ is the edge of $\mathcal{H}'$ associated to the vertex

q of $J\!T_i$. We label the corresponding vertex $\bar{q}$ in T_{p_i} as follows: $\lambda(\bar{q}) = \{h\} \cup H$, and $\chi(\bar{q}) = h \cup var(H)$.

It is easy to see that the hypertree HD is a hypertree decomposition of $\mathcal{H}$, and its width is $k+1$. It follows that HYPERCUTSET $\preceq$ HYPERTREE. $\square$

8 Binary CSPs

In this section, we focus on binary constraints satisfaction problems, i.e., on CSPs where the constraints relations have arity at most two.

On binary constraint networks, the differences among the decomposition strategies, highlighted in Section 7, become less evident. Indeed, bounding the arities of the constraint relations, the k-tractable classes of some decomposition strategies collapse, while some generalizations are no longer strong generalizations.

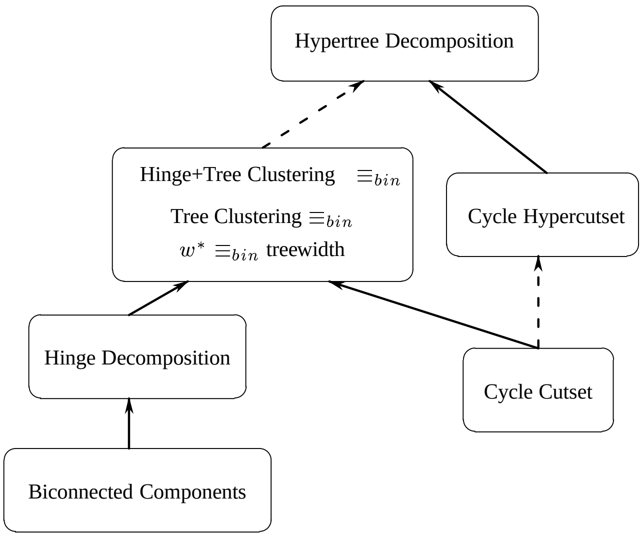


Figure 25. Tractability Hierarchy for Binary CSPs

Let $\lll_{bin}$, $\preceq_{bin}$, $\rhd_{bin}$, and $\equiv_{bin}$ the relations on the decompositions strategies induced by $\lll$, $\preceq$, $\rhd$, and $\equiv$, respectively, when only binary CSPs are considered.

In Figure 25, full arcs (and paths containing full arcs) represent $\lll_{bin}$ relationships, while a dashed arc from a method D_1 to a method D_2 means that $D_1 \preceq_{bin} D_2$ and $D_2 \not\preceq_{bin} D_1$, but at the same time $D_1 \not\lll_{bin} D_2$. From the latter relationship, it follows that every class C that is tractable according to D_1 is also tractable according to D_2, i.e., the D_2 width of every graph belonging to the class C is bounded by some constant $k > 0$. However, $D_2 \not\preceq_{bin} D_1$ entails that D_2 decompositions are more "efficient," in the sense that solving a D_1-tractable class by D_2-solution methods is feasible by augmenting the worst-case complexity by at most an additive constant in the exponent, while this is not possible in the other direction.

Theorem 37 *For each pair D_1 and D_2 of decompositions methods represented in Figure 25, the following holds:*

- *There is a directed path from D_1 to D_2 if and only if $D_1 \preceq_{bin} D_2$.*
- *There is a directed path containing at least one full arrow from D_1 to D_2 if and only if $D_1 \prec\!\!\prec_{bin} D_2$.*
- *D_1 and D_2 are not linked by any directed path if and only if they are incomparable with respect to the $\preceq_{bin}$ relationship, i.e., if both $D_1 \not\preceq_{bin} D_2$ and $D_2 \not\preceq_{bin} D_1$ hold.*

The following lemmas provide the proof of this theorem.

Lemma 38 $\mathtt{HINGE} \prec\!\!\prec_{bin} \mathtt{TCLUSTER}$.

PROOF. First note that $\mathtt{TCLUSTER} \rhd_{bin} \mathtt{HINGE}$ follows from the proof showing that $\mathtt{TCLUSTER} \rhd \mathtt{HINGE}$. Indeed, for any $n > 2$, the graph *Circle*$(n, 1)$ has degree of cyclicity n, while it has TCLUSTER width 3.

To prove that $\mathtt{HINGE} \preceq_{bin} \mathtt{TCLUSTER}$, we show that for any graph $G = (V, E)$ HINGE-width$(G) \geq$TCLUSTER-width(G). If G is an acyclic graph, then its degree of cyclicity is 2 and its TCLUSTER width is 1, by definition. Now, assume G is a cyclic graph and let T be a hinge decomposition of G. From the definition of hinge decomposition, it follows that T represents a join tree of an acyclic hypergraph.

We recall from [19] that, given a hinge H of G, $H' \subseteq H$ is a hinge of G if and only if H' is a hinge of the graph $(var(H), H)$. It follows that any minimal hinge of G must be a connected set of edges. Moreover, it is easy to see that if H is a minimal hinge and $(var(H), H)$ is acyclic, then $|H| = 2$.

Let T' be a new join tree initially set equal to T. As long as there exists some vertex of T' corresponding to a 2-edges hinge of G, modify T' as follows: (1) select a vertex p of T' containing two edges of G e_1 and e_2; (2) add to T' two vertices p_1 and p_2 containing edges e_1 and e_2, respectively; (3) add an edge connecting p_1 and p' for any vertex p' of T' connected to p and sharing e_1 with p; (4) add an edge connecting p_2 and p' for any vertex p' of T' connected to p and sharing e_2 with p; (5) remove p and all its incident edges from T'. It is easy to verify that the final tree T' obtained when the procedure above terminates satisfies the connectedness condition of join trees. In fact, it represents an acyclic hypergraph, say $\mathcal{H}'$.

Let G' be the primal graph of $\mathcal{H}'$. The graph G' is clearly chordal and $E \subseteq E'$, thus it can be obtained by some suitable triangulation of G. Let k be the number of variables occurring in the largest clique C of G'. Since G is a cyclic graph, $k > 2$. By construction of G', the clique C corresponds to some minimal hinge H of G such that the graph $(var(H), H)$ is both connected and cyclic. This entails that $|H| \geq var(H) = k$.

It follows that $k \leq$HINGE-width(G), because HINGE-width(G) is equal to the cardinality of the largest minimal hinge of G. Thus the lemma holds, because TCLUSTER-width$(G) \leq k$, as G' witnesses that there exists a graph obtained by some triangulation of G whose maximal clique has cardinality k. $\square$

Lemma 39 *The following relationships hold between* HYPERTREE *and* TCLUSTER*:*

- TCLUSTER $\preceq_{bin}$ HYPERTREE*;*
- HYPERTREE $\not\rhd_{bin}$TCLUSTER*; and*
- HYPERTREE $\not\preceq_{bin}$ TCLUSTER.

PROOF. (TCLUSTER $\preceq_{bin}$ HYPERTREE.) Easily follows from the same construction described in Lemma 33 to prove that HINGE$^{\text{TCLUSTER}}$ $\preceq$ HYPERTREE.

(HYPERTREE $\not\rhd_{bin}$ TCLUSTER.) Follows from the fact that, for any graph G, TCLUSTER-width$(G) \leq 2\cdot$HYPERTREE-width(G). Let HD be any k-width hypertree decomposition of a graph G. The hypergraph corresponding to the acyclic instance built according to HD has a primal graph G' whose largest clique contains $2 \cdot k$ variables at most. Indeed, at most k edges can be associated to any vertex p of the hypertree decomposition and hence $var(p) \leq 2 \cdot k$.

(HYPERTREE $\not\preceq_{bin}$ TCLUSTER.) Observe that, for every $n > 3$, the complete graph K_n has HYPERTREE width $\lceil n/2 \rceil$, while it has TCLUSTER width n. Thus, $K_n \in C(\mathtt{HYPERTREE}, n')$, for each $n' \geq \lceil n/2 \rceil$, while $K_n \notin C(\mathtt{TCLUSTER}, n'')$, for each $n'' < n$. It follows that there is no fixed δ such that, for every $k > 0$, $C(\mathtt{HYPERTREE}, k) \subseteq C(\mathtt{TCLUSTER}, k + \delta)$. $\square$

Lemma 40 *The following relationships hold between* HYPERCUTSET *and* CUTSET*:*

- CUTSET $\preceq_{bin}$ HYPERCUTSET*;*
- HYPERCUTSET $\not\rhd_{bin}$CUTSET*; and*
- HYPERCUTSET $\not\preceq_{bin}$ CUTSET.

PROOF. The proofs of the first two points above are straightforward. We next show that HYPERCUTSET $\not\preceq_{bin}$ CUTSET. Consider the graph $triangles(n)$ for some $n > 0$. It is easy to see that the HYPERCUTSET width of $triangles(n)$ is $\lceil n/3 \rceil$, while its CUTSET width is $\lceil n/2 \rceil$. Thus, $triangles(n) \in C(\mathtt{HYPERCUTSET}, n')$, for each $n' \geq \lceil n/3 \rceil$, while $triangles(n) \notin C(\mathtt{CUTSET}, n'')$, for each $n'' < \lceil n/2 \rceil$. It follows that there is no fixed δ such that, for every $k > 0$, $C(\mathtt{HYPERCUTSET}, k) \subseteq C(\mathtt{CUTSET}, k + \delta)$. $\square$

All the other relationships follow from transitivity, or from the corresponding proofs given in the general case of hypergraphs, which carry over to the binary case.

9 Solving nonbinary CSPs by dualization

Many structural decomposition methods have been designed to identify "easy" graph structures, rather than "easy" hypergraph structures. In Section 4, we described binary decomposition methods (i.e., decomposition methods designed for graphs, but not for hypergraphs) acting on the primal graph of the hypergraph associated to the given CSP instance. As we showed in the previous section, for binary CSPs some methods become closer to the hypertree-decomposition method.

An alternative approach to the solution of nonbinary CSPs is to exploit binary methods on the dual graph of a hypergraph. (See, e.g., [7].) Given a CSP instance I, the *dual graph* [7,9,22] 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 jointree of $\mathcal{H}_I$. In this case, the reduction is feasible in polynomial (actually, linear) time. (See, e.g., [22].)

Definition 41 *Let $G = (V, E)$ be the dual graph of some hypergraph $\mathcal{H}$. For any pair of vertices $h, h' \in V$, let $\ell(\{h, h'\}) = h \cap h'$. A* reduct *$G'$ of G is a graph (V', E') satisfying the following conditions:*

(i) $V' = V$;
(ii) $E' \subseteq E$; *and*
(iii) *for each edge $q = \{h, h'\}$ belonging to $(E - E')$, there exists in G' a path P from h to h', such that the variables in $\ell(q)$ are included in $\ell(q')$ for each edge q' occurring in the path P. That is, if all the variables shared by two vertices occur in some other path between these vertices, the edge connecting them can be safely deleted from the dual graph.*

We denote by $red(G)$ the set of all the *minimal reducts* of a graph G, i.e., the set containing every graph G' which is a reduct of G and whose set of edges is minimal (with respect to set inclusion) over all the reducts of G. Clearly, computing a graph belonging to $red(G)$ is feasible in polynomial time, because one can just repeatedly delete an edge as long as possible.

It is thus natural to try to solve a nonbinary CSP I using any decomposition method

DM on its dual graph:

(1) compute from G_I^d a suitable reduct $G \in red(G_I^d)$;
(2) compute a DM decomposition of the graph G;
(3) solve the instance I using this decomposition.

For instance, `BICOMP` can easily be modified to be used on the dual graph of a given hypergraph [11]. Call this dual version $\texttt{BICOMP}^d$. The relationship between $\texttt{BICOMP}^d$ and `HINGE` has already been discussed in [18]: it was proved that `HINGE` is more general than $\texttt{BICOMP}^d$. However, Gyssens et al. observed that a fine comparison between the two methods is quite difficult because the performance of $\texttt{BICOMP}^d$ strongly depends on the simplification applied to G_I^d, i.e., depends on the particular graph in $red(G_I^d)$ selected to solve the given CSP instance I. They also argued that there is no obvious way to find a suitable simplification good enough to keep small the biconnected width of the reduct to be used for solving the problem.

Since `HYPERTREE` strongly generalizes `HINGE`, it follows that `HYPERTREE` strongly generalizes $\texttt{BICOMP}^d$. However, as suggested by Dechter (personal communication), it would be interesting to compare `HYPERTREE` with the dual version of `TCLUSTER` (short: $\texttt{TCLUSTER}^d$), defined as follows. Let $\mathcal{H}$ be a hypergraph, and G its dual graph. An acyclic hypergraph $\mathcal{H}^*$ is a $\texttt{TCLUSTER}^d$ decomposition of $\mathcal{H}$ of width w if $\mathcal{H}^*$ is a `TCLUSTER` decomposition of G' of width w, for some reduct $G' \in red(G)$. The *dual tree-clustering width* (short: $\texttt{TCLUSTER}^d$ *width*) of $\mathcal{H}$ is equal to the minimum width over the $\texttt{TCLUSTER}^d$ decompositions of $\mathcal{H}$.

We next show that `HYPERTREE` strongly generalizes the $\texttt{TCLUSTER}^d$ method, too. To this end, we introduce a new class of hypergraphs. For any $n > 1$ let *D-Clique*(n) be the hypergraph having $n + 2$ edges $\{h_a, h_b, h_1, h_2, \ldots, h_n\}$ defined as follows:

- $h_a = \{X_{ij}^a \mid 1 \leq i < j \leq n\}$;
- $h_b = \{X_{ij}^b \mid 1 \leq i < j \leq n\}$;
- for $1 \leq i \leq n$, $h_i = \{X_{1i}^a, X_{2i}^a, \ldots, X_{i-1i}^a, X_{ii+1}^a, \ldots X_{in}^a\} \cup \{X_{1i}^b, X_{2i}^b, \ldots, X_{i-1i}^b, X_{ii+1}^b, \ldots X_{in}^b\}$.

We denote by $G^d(n)$ the dual graph of *D-Clique*(n).

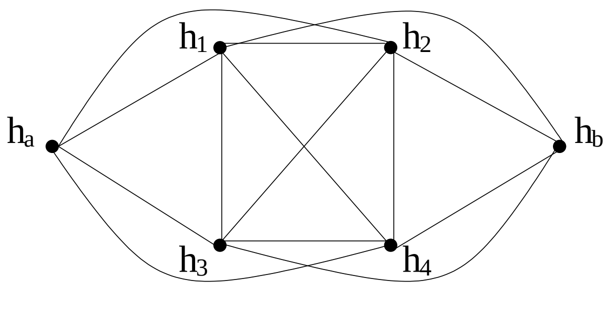


Figure 26. The dual graph of *D-Clique*(4)

Example 42 Consider the hypergraph *D-Clique*(4). Its edges are

$$
\begin{aligned}
h_1 &= \{X_{12}^a, X_{12}^b, X_{13}^a, X_{13}^b, X_{14}^a, X_{14}^b\};\\
h_2 &= \{X_{12}^a, X_{12}^b, X_{23}^a, X_{23}^b, X_{24}^a, X_{24}^b\};\\
h_3 &= \{X_{13}^a, X_{13}^b, X_{23}^a, X_{23}^b, X_{34}^a, X_{34}^b\};\\
h_4 &= \{X_{14}^a, X_{14}^b, X_{24}^a, X_{24}^b, X_{34}^a, X_{34}^b\};\\
h_a &= \{X_{ij}^a \mid 1 \leq i < j \leq 4\};\\
h_b &= \{X_{ij}^b \mid 1 \leq i < j \leq 4\}.
\end{aligned}
$$

Figure 26 shows the dual graph $G^d(4)$. Note that this graph cannot be reduced, and hence $red(G^d(4)) = \{G^d(4)\}$. For instance, consider the vertices h_1 and h_4. Their shared variables are X_{14}^a and X_{14}^b. For any $t \not\in \{1, 4, a, b\}$, $h_t \cap h_1 = \{X_{1t}^a, X_{1t}^b\}$, which clearly does not include $\{X_{14}^a, X_{14}^b\}$. Moreover, $X_{14}^b \not\in h_1 \cap h_a$ and $X_{14}^a \not\in h_1 \cap h_b$. Thus, we cannot delete the edge $\{h_1, h_4\}$, and in fact no edge can be deleted from $G^d(4)$.

Apply `TCLUSTER` to $G^d(4)$. It is already a chordal graph, therefore we can directly identify the maximal cliques, that form the edges of the `TCLUSTER` decomposition of $G^d(4)$. The resulting acyclic hypergraph has the two edges $\{h_a, h_1, h_2, h_3, h_4\}$, and $\{h_b, h_1, h_2, h_3, h_4\}$. Thus, the $\texttt{TCLUSTER}^d$ width of *D-Clique*(4) is 5.

The `HYPERTREE` width of *D-Clique*(4) is 2. Figure 27 shows a complete hypertree decomposition (T, χ, λ) of *D-Clique*(4) having width 2. Observe that, exploiting the two edges h_1 and h_2, even the root of T alone covers all the variables occurring in *D-Clique*(4), and is in fact a hypertree decomposition of this hypergraph. To obtain the complete hypertree decomposition shown in Figure 27, the remaining edges are simply "attached" as singletons to the root.

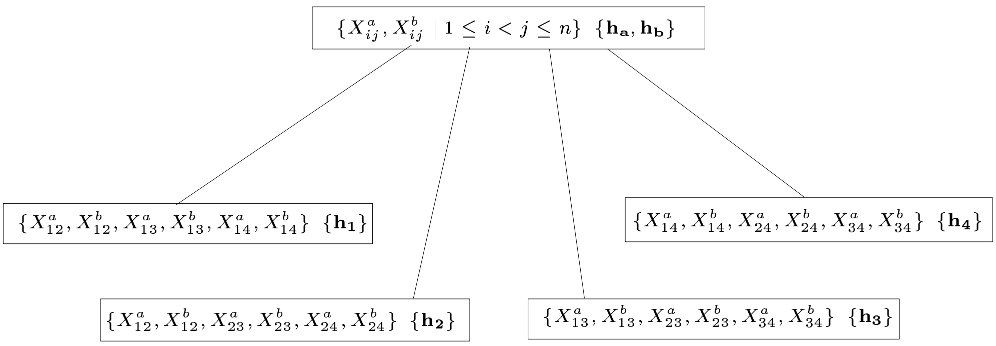


Figure 27. A hypertree decomposition of *D-Clique*(4)

Theorem 43 $\texttt{TCLUSTER}^d \prec\!\!\prec$ HYPERTREE.

PROOF. (HYPERTREE$\rhd$ $\texttt{TCLUSTER}^d$.) Consider the hypergraph class $\{$*D-Clique*$(n) \mid n > 1\}$. Generalizing the above example, it is easily seen that, for any $n \geq 3$, the set $red(G^d(n))$ is a singleton containing only the dual graph $G^d(n)$ of *D-Clique*(n).

This graph is chordal, its maximal cliques are $\{h_a, h_1, \ldots, h_n\}$ and $\{h_b, h_1, \ldots, h_n\}$, and hence the $\mathtt{TCLUSTER}^d$ width of *D-Clique*(n) is $n + 1$. Thus, for any $k > 0$, $\bigcup_{n>0}\{\textit{D-Clique}(n)\} \not\subseteq C(\mathtt{TCLUSTER}^d, k)$, whereas the hypertree width of all these hypergraphs is 2, i.e., $\bigcup_{n>0}\{\textit{D-Clique}(n)\} \subseteq C(hypertree, 2)$. Indeed, a tree with a single vertex r with $\lambda(r) = \{h_a, h_b\}$ and $\chi(r) = h_a \cup h_b$ is a hypertree decomposition of *D-Clique*(n), though not complete. Figure 27 shows what a complete hypertree decomposition for such hypergraphs looks like.

($\mathtt{TCLUSTER}^d \preceq$ HYPERTREE.) Let $\mathcal{H}'$ be a $\mathtt{TCLUSTER}^d$ decomposition of a hypergraph $\mathcal{H}$ of width k. Then, $\mathcal{H}'$ is an acyclic hypergraph whose edges are sets containing at most k edges from $\mathcal{H}$. Any join tree *JT* of $\mathcal{H}'$ can be mapped straightforwardly to a hypertree decomposition (T, χ, λ) of $\mathcal{H}$ with the same tree-shape as *JT*. Every vertex p in T corresponds to a vertex p' in *JT*. The vertex p' of the join tree of $\mathcal{H}'$ corresponds to a maximal clique of (some reduct of) the dual graph of $\mathcal{H}$, and hence contains a set S of edges occurring in $\mathcal{H}$. Then, the vertex p in the hypertree decomposition is labeled by $\lambda(p) = S$ and $\chi(p) = var(S)$. Clearly the hypertree decomposition (T, χ, λ) has the same width as the $\mathtt{TCLUSTER}^d$ decomposition $\mathcal{H}'$. $\square$

Note that the $\mathtt{TCLUSTER}^d$ width of $\mathcal{H}$ does not depend on the choice of the reduct of the dual graph. The width is in fact computed using an optimal reduct of G, i.e., a reduct leading to a lowest-width TCLUSTER decomposition of $\mathcal{H}$. However, as observed in [18], it is not clear how to choose the right reduct in order to obtain the $\mathtt{TCLUSTER}^d$ decomposition having the smallest width. In fact, it is currently not known whether, for a fixed k, deciding whether the $\mathtt{TCLUSTER}^d$ width of a hypergraph is at most k is feasible in polynomial time. Thus, compared to $\mathtt{TCLUSTER}^d$, HYPERTREE is strongly more general and k-bounded hypertree decompositions are efficiently computable.

Clearly, the above result holds for TREEWIDTH and w^*, too, given the equivalence of these methods (see Section 4).

10 Conclusion

In this paper we have established a framework for systematically comparing structural CSP decomposition methods with regard to their power of identifying large tractable classes of constraints. We have compared the main decomposition methods published in the AI literature. Moreover, we have adapted the method of hypertree decompositions, previously defined in the database context, to the CSP setting. We compared all methods both for CSPs of arbitrary arity and for binary CSPs. In both cases it turned out that the hypertree decomposition method is more general than the others; in the case of general CSPs this holds even in a very strong sense.

We have also shown that the method of hypertree decompositions is more general than any dualization method which applies a standard decomposition method to the dual graph of the constraint hypergraph of a CSP. We have derived the upper time bound $O(\|I\|^{k+1} \log \|I\|)$ for the solution of a CSP instance I having a k-width hypertree decomposition. Note that this bound is not worse than the bound for any other considered method of CSP decompositions. Thus, it appears that the method of hypertree decompositions is currently the most powerful CSP decomposition method.

The comparison results and complexity bounds presented in this paper are valid for general CSP instances whose domain size is unrestricted. Further work is needed both on suitable extensions or modifications of decomposition methods and on the comparison of the various methods for some relevant special cases, in particular, for CSPs with a *fixed domain size*. Moreover, as already remarked, both the `HINGE` and the `BICOMP` width of a hypergraph can be computed in polynomial time even if no fixed bound is given. Thus, these methods may be useful for providing in polynomial time a "measure of the cyclicity" of any arbitrary CSP instance. For some practical applications where the given CSP instances have large hypertree width, `HINGE` and `BICOMP` decompositions may be used for the fast identification of "easy" and "hard" modules (or clusters) of the constraint hypergraph. Moreover, the algorithm for computing hypertree decompositions itself may suitably be modified to identify and output clusters of low hypertree-width in case the entire hypergraph has a high width.

We believe that our comparison results provide insight into the relationship of various standard methods of constraint decomposition. Constraint satisfaction is a very lively field and several new methods and techniques for decomposing and solving CSPs are expected to be proposed in the years to come. We hope that the results of this paper, our comparison framework, and our proof techniques will be useful to other authors for assessing the relative strength of their methods, and for comparing them to existing methods.

Acknowledgements

We thank the anonymous referees for their useful comments and suggestions.

Research supported by *FWF (Austrian Science Funds)* under the project Z29-INF. Part of the work of Francesco Scarcello has been carried out while visiting the Technische Universität Wien. Part of the work of Nicola Leone has been carried out while he was with the Technische Universität Wien.

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

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

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

[4] W. Bibel. Constraint Satisfaction from a Deductive Viewpoint. *Artificial Intelligence*, 35,401–413, 1988.

[5] H.L. Bodlaender. Treewidth: Algorithmic Techniques and Results. In *Proceedings of MFCS'97*, Bratislava. LNCS 1295, Springer, pp. 19-36, 1997.

[6] Ch. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[7] R. Dechter. Constraint Networks. In *Encyclopedia of Artificial Intelligence*, second edition, Wiley and Sons, pp. 276–285, 1992.

[8] R. Dechter and J. Pearl. Network based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34(1):1–38, 1988.

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

[10] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.

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

[12] E.C. Freuder. Complexity of K-Tree Structured Constraint Satisfaction Problems. *Proceedings of AAAI'90*, 1990.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability. A Guide to the Theory of* NP-*completeness*. Freeman and Comp., NY, USA, 1979.

[14] N. Goodman, and O. Shmueli. Syntactic characterization of tree database schemas. *Journal of the ACM*, 30(4):767–786.

[15] G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries, in *Proceedings of FOCS'98*, pp.706–715, Palo Alto, CA, 1998. Full version: Technical Report DBAI-TR-98/17, available on the web as: `http://www.dbai.tuwien.ac.at/staff/gottlob/acyclic.ps`, or by email from the authors.

[16] G. Gottlob, N. Leone, and F. Scarcello. "Hypertree Decompositions and Tractable Queries," in *Proceedings of PODS'99*, Philadelphia, May, 1999. Full version to appear in *Journal of Computer and System Sciences*. A preprint of the full version is currently stored in *The Computer Research Repository*, http://xxx.lanl.gov/archive/cs.

[17] G. Gottlob, N. Leone, and F. Scarcello. “On Tractable Queries and Constraints,” in *Proceedings of the Conference on Database and Expert Systems Applications (DEXA’99*, Florence, 1999, LNCS 1677, pp. 1-15, Springer.

[18] M. Gyssens, P.G. Jeavons, and D.A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence*, 66:57–89, 1994.

[19] M. Gyssens and J. Paredaens. A Decomposition Methodology for Cyclic Databases. In *Advances in Database Theory*, volume 2, pp. 85-122. Plenum Press New York, NY, 1984.

[20] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4), 1997.

[21] Ph. G. Kolaitis and M. Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. In *Proc. of Symp. on Principles of Database Systems (PODS’98)*, pp.205–213, Seattle, Washington, 1998. Full version to appear in *Journal of Computer and System Sciences*.

[22] D. Maier, *The theory of relational databases*. Computer Science Press, Rockville, MD, 1986.

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

[24] N. Robertson and P.D. Seymour. Graph Minors II. Algorithmic aspects of tree width. *Journal of Algorithms*, 7:309–322, 1986.

[25] R. Seidel. A new method for solving constraint satisfaction problems. In *Proceedings of IJCAI’81*, 1981.

[26] M. Yannakakis. Algorithms for Acyclic Database Schemes. In *Proc. of Int. Conf. on Very Large Data Bases (VLDB’81)*, pp. 82–94, C. Zaniolo and C. Delobel Eds., Cannes, France, 1981.