Artificial Intelligence 174 (2010) 382–409

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

On the power of structural decompositions of graph-based representations of constraint problems

Gianluigi Greco [a], Francesco Scarcello [b,*]

[a] Dept. of Mathematics, Università della Calabria, I-87036 Rende, Italy
[b] DEIS, Università della Calabria, I-87036 Rende, Italy

ARTICLE INFO

Article history:
Received 21 January 2009
Received in revised form 14 December 2009
Accepted 17 December 2009
Available online 4 January 2010

Keywords:
Constraint satisfaction
Decomposition methods
Hypergraphs
Dual graphs
Incidence graphs
Primal graphs
Treewidth

ABSTRACT

The Constraint Satisfaction Problem (CSP) is a central issue of research in Artificial Intelligence. Due to its intractability, many efforts have been made in order to identify tractable classes of CSP instances, and in fact deep and useful results have already been achieved. In particular, this paper focuses on structural decomposition methods, which are essentially meant to look for near-acyclicity properties of the graphs or hypergraphs that encode the structure of the constraints interactions. In general, constraint scopes comprise an arbitrary number of variables, and thus this structure may be naturally encoded via hypergraphs. However, in many practical applications, decomposition methods are applied over suitable graph representations of the (possibly non-binary) CSP instances at hand. Despite the great interest in such binary approaches, a formal analysis of their power, in terms of their ability of identifying islands of tractability, was missing in the literature.

The aim of this paper is precisely to fill this gap, by studying the relationships among binary structural methods, and by providing a clear picture of the tractable fragments of CSP that can be specified with respect to each of these decomposition approaches, when they are applied to binary representations of non-binary CSP instances. In particular, various long-standing questions about primal, dual and incidence graph encodings are answered. The picture is then completed by comparing methods on binary encodings with methods specifically conceived for non-binary constraints.

1. Introduction and summary of results

Constraint satisfaction is a central issue of research in Artificial Intelligence and other areas of computer science and it has, in fact, an impressive spectrum of applications ranging from scheduling with preferences and deadlines, to temporal reasoning, machine learning, and plan design, just to cite a few. Formally, a constraint (S_i, R_i) consists of a *constraint scope* S_i, i.e., a list of variables, and of an associated *constraint relation* r_i containing the legal combinations of values for the variables in S_i. An instance of a *constraint satisfaction problem* (CSP), also called *constraint network* [6], 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. A *solution* to a CSP instance is a substitution $\vartheta : Var \longrightarrow U$, such that all constraints are simultaneously satisfied, i.e., for each $1 \leqslant i \leqslant q$, $S_i\vartheta \in r_i$. By *solving* a CSP we mean determining whether the problem has a solution at all (i.e., checking for *constraint satisfiability*) and, if so, computing one solution.

* Corresponding author. Tel.: +39 0984 494752, fax: +39 0984 494713.
E-mail addresses: ggreco@mat.unical.it (G. Greco), scarcello@deis.unical.it (F. Scarcello).

0004-3702/$ – see front matter
doi:10.1016/j.artint.2009.12.004

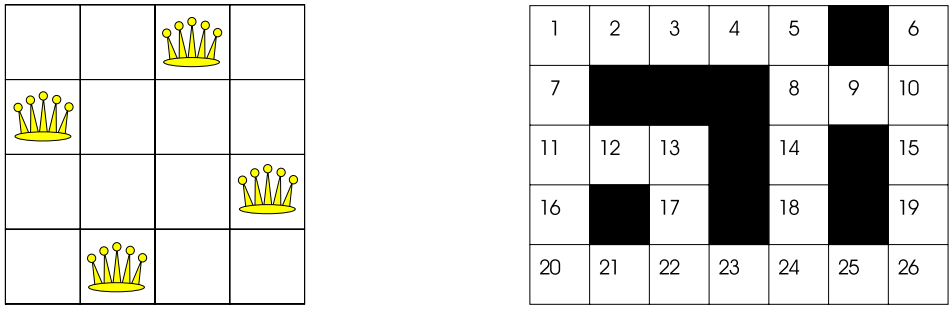


Fig. 1. (a) A solution to the 4-queens problem, and (b) a crossword puzzle.

Example 1.1 *(n-queens).* Consider the problem of placing n queens on (the n rows of) a chessboard so that no queen can capture any other queen. This problem can be formalized as a constraint satisfaction problem as follows. The set Var contains a variable Q_i for each queen to be placed in the i-th row, $U = \{1, \ldots, n\}$ represents the columns where each queen can be placed, and for each pair of distinct queens Q_i and Q_j, the set $\mathcal{C}$ contains the constraint $C_{i,j} = ((Q_i, Q_j), \{\langle p_i, p_j\rangle \mid p_i \neq p_j \wedge |p_i - p_j| \neq |i - j|\})$ stating, intuitively, that any two queens can be placed neither on the same column, nor on the same diagonal of the board.

As an example, for $n = 3$ the problem does not admit solutions. Instead, for $n = 4$, a possible solution is the substitution ϑ such that $Q_1\vartheta = 3$, $Q_2\vartheta = 1$, $Q_3\vartheta = 4$, and $Q_4\vartheta = 2$. This solution is depicted in Fig. 1(a).

Note that in the n-queens problem each constraint is defined over two variables at most, and this is therefore an example of *binary* CSP. In other scenarios, instead, it is much more natural to define constraints over more than two variables, thereby leading to the general case of *non-binary* CSPs.

Example 1.2 *(Crossword puzzle).* Fig. 1(b) shows a combinatorial crossword puzzle (taken from [10]), which is a typical non-binary CSP [6]. 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 can be recast in a CSP 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, and the relation of C contains the legal words for D.

For the example, in Fig. 1, 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 crossword puzzles. A possible instance for the relation r_{1H} is $\{\langle h, o, u, s, e\rangle, \langle c, o, i, n, s\rangle, \langle b, l, o, c, k\rangle\}$.

It is well known and easy to see that constraint satisfiability is an NP-complete problem, even when restricted over binary instances (since it can encode, for example, graph colouring). Hence, much effort has been spent to identify *tractable classes* of CSPs, and deep and useful results have already been achieved in the literature. In fact, the various successful approaches to single out tractable CSP classes can be divided into two main groups (see, e.g., [4,10]):

(i) Techniques that look for tractable classes on the basis of the structure of the constraint scopes $\{S_1, \ldots, S_q\}$, independently of the actual constraint relations $r_1, \ldots, r_q$; and
(ii) Techniques that exploit peculiar properties (such as combinatorial properties of the underlying algebras) of the constraint relations $r_1, \ldots, r_q$.

In this paper, we shall deal with the former kind of techniques, usually called *structural decomposition methods*.

1.1. Structural decomposition methods

Much of the nature of constraint scope interactions can be captured using the constraint hypergraph $\mathcal{H}(I) = (V, H)$ associated to any CSP instance $I = (Var, U, \mathcal{C})$, where $V = Var$ and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, and $var(S)$ denotes the set of variables in the scope S of the constraint C—in the following, we often denote the set of vertices V by $\mathcal{N}(\mathcal{H})$ and the set of hyperedges H by $\mathcal{E}(\mathcal{H})$. For instance, Fig. 2 shows the hypergraph $\mathcal{H}_{cp}$ associated to the crossword puzzle in Example 1.2.

A fundamental property of hypergraphs is *acyclicity* (see, e.g., [2,9]). Indeed, it has been observed that constraint satisfiability is feasible in polynomial time on the class of those CSP instances whose associated hypergraphs are acyclic (see,

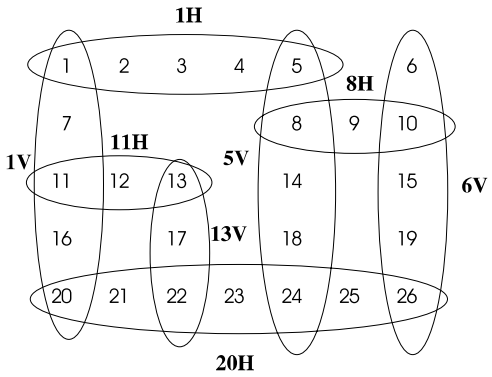


Fig. 2. Hypergraph $\mathcal{H}_{cp}$ of the crossword puzzle in Example 1.2.

e.g., [10]). However, in practice, constraint hypergraphs are hardly acyclic (for instance, the hypergraph $\mathcal{H}_{cp}$ in Fig. 2 is not acyclic), even though not very intricate. Therefore, there is a great deal of interest in the literature in identifying tractable classes of constraint satisfaction problem instances by looking for *nearly* acyclic structures, as for they can be characterized via structural decomposition methods (see, e.g., [5,6,10,22]). These methods aim at transforming a given cyclic hypergraph into an acyclic one, by organizing its edges or its nodes into a polynomial number of clusters, and by suitably arranging these clusters as a tree, called a decomposition tree. The original problem instance can then be evaluated over the decomposition tree, with a cost that is exponential in the cardinality of the largest cluster, also called *width* of the decomposition.

In fact, the earliest decomposition techniques were designed to solve binary CSPs only, such as the n-queens problem of Example 1.1; that is, they have been designed to deal with scenarios where the constraint hypergraph is actually a graph (call them *binary* or *graph methods*). Subsequently, methods have been proposed which are capable of working on the constraint hypergraph without assuming any bound on the number of variables involved in each of the constraints and, hence, without limiting their applicability to the special case of binary CSPs. In particular, much research is currently aimed at defining completely novel methods exploiting the whole information encoded in the constraint hypergraph (call them *non-binary* or *hypergraph-based methods*). However, many attempts to deal with general (i.e., non-binary) constraint problems have historically been conceived to reuse existent methods for binary CSPs, by representing any instance I by some graph, rather than by the hypergraph $\mathcal{H}(I)$. With this respect, a natural idea (widely exploited in the literature) is to use, as representative graph, the primal graph of $\mathcal{H}(I)$ defined as follows:

- **Primal-graph representation**. Given a hypergraph $\mathcal{H}$, its *primal graph* (*also Gaifman graph*), denoted by $G(\mathcal{H}) = (N, E)$, is the graph whose set of nodes N is the set of variables $\mathcal{N}(\mathcal{H})$, and whose edges connect each pair of nodes (i.e., variables) occurring together in some constraint of I, that is $E = \{\{V_1, V_2\} \mid V_1, V_2 \in \mathcal{N}(\mathcal{H})$ and $\exists h \in \mathcal{E}(\mathcal{H})$ s.t. $\{V_1, V_2\} \subseteq h\}$. For example, Fig. 3(a) shows the primal graph of $\mathcal{H}_{cp}$.

Clearly, there is an evident loss of information in using the primal graph instead of the constraint hypergraph. For instance, each constraint scope of I induces a clique in the primal graph. If one looks at the graph only, there is no way to understand whether such a clique comes from a single scope, or actually from some intricate interactions among many constraints scopes.

In fact, a deep comparison among various structural decomposition methods applied to the primal graph has been carried out in [10], where it is moreover shown that the *hypertree* [12] decomposition method (short: HYPERTREE)—which instead directly applies to the constraint hypergraph—is more powerful than all the (known) techniques working on primal graphs.

Besides the primal-graph representation, however, two other graph-based representations of non-binary CSP instances have been described and used in the literature, which were instead only marginally considered in [10]:

- **Dual-graph representation** [6]. Given a hypergraph $\mathcal{H}$, its *dual graph*, denoted by $dual(\mathcal{H}) = (N, E)$, is the graph whose set of nodes N is the set of hyperedges $\mathcal{E}(\mathcal{H})$, and whose edges connect each pair of nodes (i.e., hyperedges) having some variable in common, that is $E = \{\{h_1, h_2\} \mid h_1, h_2 \in \mathcal{E}(\mathcal{H})$ and $h_1 \cap h_2 \neq \emptyset\}$. For example, Fig. 3(b) shows the dual graph of $\mathcal{H}_{cp}$.
- **Incidence-graph representation**[1] [23,3]. Given a hypergraph $\mathcal{H}$, we define its *incidence graph* as the bipartite graph $inc(\mathcal{H}) = (N, E)$, where $N = \mathcal{E}(\mathcal{H}) \cup \mathcal{N}(\mathcal{H})$, and $E = \{\{h, a\} \mid h \in \mathcal{E}(\mathcal{H})$ and $a \in h\}$, i.e. it contains an edge from h to a if and only if a is a node of h. For example, Fig. 3(c) shows the incidence graph of $\mathcal{H}_{cp}$.

[1] Also known as hidden-variable encoding.

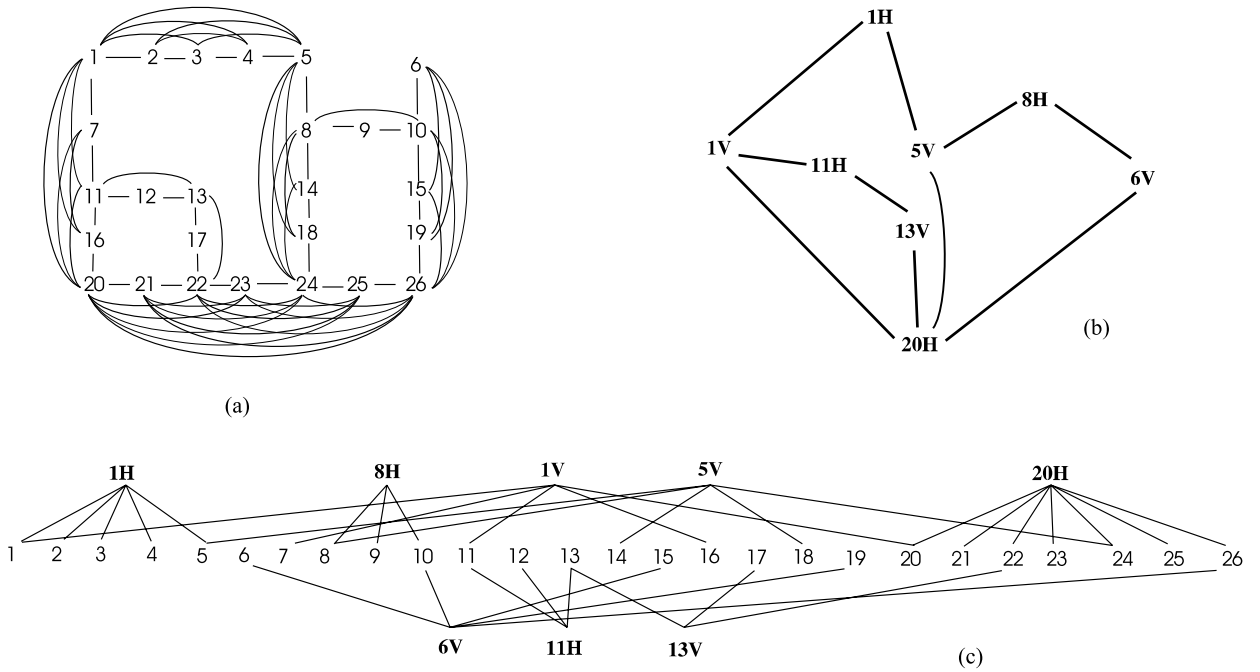


Fig. 3. Binary representations for $\mathcal{H}_{cp}$: (a) primal graph, (b) dual graph, and (c) incidence graph.

A first comparison of dual-graph and incidence-graph encodings has been carried out in [1], in the light of assessing the performances of forward checking and maintaining arc consistency algorithms. However, no formal comparison has been reported in the literature, which is focused on the intrinsic "power" of structural decomposition methods based on these binary encodings. In fact, there are several long-standing questions about the relationships among these methods, as well as about their relationships with hypergraph-based techniques. In particular, dual-graph encodings received much attention in the literature, but a deep understanding of their properties is still missing.

One of the major difficulties in doing a precise and formal analysis of dual-graph based methods is due to an important feature of this encoding: Some edges of the dual graph can safely be removed, thereby making the evaluation of CSPs easier. Indeed, even if $dual(\mathcal{H})$ appears very intricate, sometimes it is possible to find suitable simplifications that make it much more useful (as we shall formally discuss in Section 3). Such simplified graphs are called *reducts* of $dual(\mathcal{H})$. For instance, if $\mathcal{H}$ is acyclic, it is well known that there is a polynomial time algorithm that builds an acyclic reduct of its dual graph. However, when generalizations of acyclicity are concerned, dealing with reducts is not that easy, because different removal choices may lead to different performances of evaluation algorithms. Therefore, the following challenge comes into play when analyzing decomposition methods based on the dual-graph encoding:

(1) The efficiency of any technique based on the dual graph crucially depends on the availability of a good algorithm/methodology for its simplification. But, finding an "optimal reduct" (w.r.t. the technique at hand) is a difficult task and, for some relevant decomposition techniques, is currently not known to be feasible in polynomial time. As a matter of fact, most of the results in the literature only provide rough bounds on the power of decomposition methods applied to dual graphs, by pragmatically not allowing any simplification in the theoretical analysis, or by exploiting just heuristic approaches. As an example, Gyssens et al. [15] contrasted the notion of hinge decompositions (short: HINGE) with the notion of biconnected components [8] of the dual graph (short: $\texttt{BICOMP}^d$). It turned out that HINGE is a generalization of $\texttt{BICOMP}^d$, and thus the hinge decomposition technique is not worse than the biconnected components technique. However, the precise relationship between these methods remained an open question. Indeed, it was observed that the biconnected components—and thus their largest cardinalities—may differ very much among the many possible reducts of the dual graph.

Similarly, very few results were known about the power of decomposition methods applied to incidence-graph encodings and, in particular, on how they compare with other binary encodings. Indeed,

(2) It is well known that the tree decomposition method [20] applied to the incidence graph (short: $\texttt{TREEDECOMP}^{in}$) may be used for identifying tractable classes of non-binary CSPs [18], as well as the tree decomposition applied to the primal and to the dual graph (short: $\texttt{TREEDECOMP}^{p}$ and $\texttt{TREEDECOMP}^{d}$, respectively). However, a thorough analysis of

the power of these three methods has never been carried out, and it was not known whether any of these encodings is definitively better than the other possible applications of tree decompositions.

Finally, very little was known about how binary methods compare with hypergraph-based techniques. Indeed,

(3) While it is known that TREEDECOMP^{optd} is definitively less powerful than HYPERTREE (cf. [10]), the actual reason for such a big difference between them is not well-understood, since at a first glance the kind of tree labelling in these methods seems rather similar. In addition, nothing was known about how HYPERTREE compares with TREEDECOMP^{in}, and about how other hypergraph-based methods, such the spread cuts [5] decomposition method (short: SPREADCUT), or the component hypertree [13] decomposition method (short: CHYPERTREE) compare in general with binary methods.

1.2. Main results

In this paper, we embark on a systematic analysis of the formal properties and of the decomposition power of both dual and incidence-graph encodings, thereby complementing the results of [10], which were mainly focused on the primal graph representation. In particular, we face the issue (1) above pertaining dual-graph encodings, and we fill the gaps discussed in (2) and (3) about incidence graphs and other hypergraph-based methods. Moreover, we investigate decomposition methods earlier introduced in the literature, so that the combination of all our results leads to a complete and clear picture of the relationships among decomposition methods applied to binary representations of general CSP instances, and of the relationships of these methods with hypergraph-based techniques.

In order to carry out our analysis, we adopt the criteria introduced in [10], and later used in other papers about structural methods (see, e.g., [5]), where any pair of decomposition methods are compared according to their ability of identifying tractable classes of CSPs. Informally, a method $\mathtt{D}_2$ *generalizes* a method $\mathtt{D}_1$ if each CSP that is tractable according to $\mathtt{D}_1$ is also tractable according to $\mathtt{D}_2$ (the $\mathtt{D}_2$-width is smaller than the $\mathtt{D}_1$-width on every instance); $\mathtt{D}_2$ *beats* $\mathtt{D}_1$ if some CSPs are tractable according to $\mathtt{D}_2$ but not according to $\mathtt{D}_1$; $\mathtt{D}_2$ *strongly generalizes* $\mathtt{D}_1$ if $\mathtt{D}_2$ both generalizes and beats $\mathtt{D}_1$; and $\mathtt{D}_1$ and $\mathtt{D}_2$ are *equivalent* if $\mathtt{D}_1$ generalizes $\mathtt{D}_2$ and $\mathtt{D}_2$ generalizes $\mathtt{D}_1$.

Note that results comparing the widths of structural methods also provide information on how well these methods decompose the given instances—preparatory to solving them—rather than just saying which tractable classes are specifiable. Indeed, the running time of every algorithm solving CSPs using any method considered in this paper must be exponential in the corresponding notion of width, unless some unexpected event occurs in the theory of parameterized complexity [7]. This can be easily seen by observing that the k-clique problem on a graph $G = (V, E)$ may be encoded as a binary CSP instance $I(G)$ with k variables $X_1, \ldots, X_k$ over the domain V, and the $k \times (k-1)/2$ constraints $((X_i, X_j), E)$, $\forall 1 \leqslant i < j \leqslant k$. By exploiting this encoding, it immediately follows that k-clique is *fp*-reducible to solving bounded D-width CSP instances, where D is any method considered in this paper, and thus solving CSPs by any of these techniques is W[1]-hard.[2] Therefore, unless some collapse occurs in the weft hierarchy of complexity classes (see, e.g., [7]), any algorithm for solving these instances must run in time $O(n^{f(w)})$, where w is the D-width and $f(w)$ is a function—that is believed to be $\Omega(\sqrt{w})$ for dual-graph methods and $\Omega(w)$ for all the others, unless the barrier of $O(n^{\Omega(k)})$ is broken for deeply studied W[1]-hard problems such as k-clique, k-independent set, and k-dominating set. Therefore, the typical dynamic programming approach based on Yannakakis's algorithm, whose cost is $O(n^{w+1} \log n)$—see, e.g., [22], is likely almost optimal, in that no algorithmic breakthrough is expected that may improve significantly its running time. In particular, under the above complexity assumptions, it is not possible to find any algorithm where the exponential dependency from the width is a factor of some polynomial of the input size, e.g., has the better form $O(c_1^w n^{c_2})$, with c_1 and c_2 being fixed constants—see [14] and [21], for results on these issues and some good news for classes of instances where additional parameters are fixed (besides the D-width).

Thus, for a given class of CSP instances it is meaningful to consider not only whether the $\mathtt{D}_1$-width of these instances is bounded (according to some structural method $\mathtt{D}_1$), but also how large these widths are, especially if compared with the widths of some other method $\mathtt{D}_2$. In particular, a very relevant relationship for this paper is the following. Consider a pair of structural methods $\mathtt{D}_1$ and $\mathtt{D}_2$ such that $\mathtt{D}_1$ beats $\mathtt{D}_2$, and the $\mathtt{D}_1$-width is at most exponentially larger than the $\mathtt{D}_2$-width on any CSP instance. Therefore, $\mathtt{D}_2$ cannot beat $\mathtt{D}_1$, because any class of CSP instances whose $\mathtt{D}_2$-width is bounded by some constant k have the $\mathtt{D}_1$-width bounded by some constant $k' \leqslant \exp(k)$. Hence, every class that is tractable according to $\mathtt{D}_2$ is also tractable according to $\mathtt{D}_1$. However, assume that we know that the above upper-bound on the width is also strict, in that there is some class $\mathcal{C}$ of CSP instances whose $\mathtt{D}_2$-width is bounded by some $k \geqslant 1$, and such that the $\mathtt{D}_1$-width is exponentially larger than the $\mathtt{D}_2$-width. Even if $\mathcal{C}$ is tractable according to $\mathtt{D}_1$, in practice the method $\mathtt{D}_2$ should be preferred to $\mathtt{D}_1$ on this class. Indeed, from the above lower bounds we expect that, by using any algorithm based on $\mathtt{D}_1$, the (worst case) solving of CSP instances from $\mathcal{C}$ will perform quite bad, if compared to algorithms exploiting $\mathtt{D}_2$.

Interestingly, many occurrences of the above relationship between structural methods come out while comparing techniques based on different graph encodings. To properly deal with them, we formally define the notion of *weak generalization*

[2] In fact, this is the well-known fixed-parameter intractability proof used for k-bounded treewidth instances (see, e.g., [14]).

and we say, for a pair $\mathtt{D}_1$, $\mathtt{D}_2$ as above, that $\mathtt{D}_1$ *weakly generalizes* $\mathtt{D}_2$. Note that the term "generalization" is due to the fact that a class is tractable according to $\mathtt{D}_1$ whenever it is tractable according to $\mathtt{D}_2$, while the term "weak" recalls that such $\mathtt{D}_1$ is quite useless for some class of instances where $\mathtt{D}_2$ is useful, instead. Thus, while in theory $\mathtt{D}_1$ is slightly better than $\mathtt{D}_2$, in practice the two methods are somehow incomparable, with the choice between them depending on the specific kind of problems to be solved.

Armed with the above notions and comparison criteria, we can now outline our contribution.

In the first part of the paper we study the dual-graph encoding. Note that when a method of finding the reduct to be used is not specified we can only compare decomposition methods loosely. To deal with this issue, we preliminarily define, for each method D, the method $\mathtt{D}^{optd}$ as the method D applied to an optimal reduct of the dual graph with respect to D (i.e., to any reduct of the dual graph having the minimum possible D-width). We first show in Theorem 3.3 that when a decomposition method (possibly using an arbitrary reduction algorithm) beats (and/or generalizes) another, then the same relationship will be true when considering optimal reducts. Moreover, the widths of decomposition methods applied to optimal reducts of dual graphs are well-defined, so that these methods may be precisely compared with other methods, according to the above criteria. Then, based on this notion,

- ▷ We answer a question firstly faced in Gyssens et al. [15], by proving that $\mathtt{BICOMP}^{optd}$ is equivalent to HINGE. In fact, we show that any hinge decomposition corresponds to the biconnected-components tree of some reduct of the dual graph. It is worthwhile noting that, as a corollary of this result, we obtain that, for the $\mathtt{BICOMP}^{optd}$ method, an optimal reduct of the dual graph can be computed in polynomial time, since a hinge decomposition can be computed in polynomial time [15].
- ▷ We consider the tree decomposition method applied to an optimal reduct of the dual graph (short: $\mathtt{TREEDECOMP}^{optd}$), and we show that this method is definitively better than any other method D applied to an optimal reduct of the dual graph w.r.t. D. In particular, we show that $\mathtt{TREEDECOMP}^{optd}$ strongly generalizes the biconnected-components method $\mathtt{BICOMP}^{optd}$ (and, hence, HINGE).
- ▷ We establish the precise relationship between $\mathtt{BICOMP}^{optd}$ and various methods applied to the primal graph (and analyzed in [10]), namely the biconnected-components method on the primal graph (short: $\mathtt{BICOMP}^{p}$), the cutset [16] decomposition on the primal graph (short: $\mathtt{CUTSET}^{p}$), and the tree decomposition method [20] applied to the primal graph ($\mathtt{TREEDECOMP}^{p}$).
- ▷ We introduce a novel decomposition method, called *weak query decomposition* (short: weakQUERYDECOMP), to shed some light on the precise decomposition power of $\mathtt{TREEDECOMP}^{optd}$, and to overcome some technical difficulties in comparing this notion with other techniques. It is defined on the constraint hypergraph and has essentially the same decomposition power of $\mathtt{TREEDECOMP}^{optd}$ (formally, weakQUERYDECOMP is a 2-approximation of $\mathtt{TREEDECOMP}^{optd}$). Therefore, this method does not require the computation of any reduct, and it can be used in place of $\mathtt{TREEDECOMP}^{optd}$, because it is equivalent to it. This relationship is also interesting because weakQUERYDECOMP is a tractable notion, while finding the most appropriate way of simplifying the dual-graph w.r.t. the tree decomposition method is a well-known challenging problem (cf. [18]).

In the second part of the paper, we turn to the analysis of the incidence-graph representation, and specifically of the tree decomposition method applied to it ($\mathtt{TREEDECOMP}^{in}$). In particular,

- ▷ We show that $\mathtt{TREEDECOMP}^{in}$ is incomparable with HINGE, that is, HINGE beats $\mathtt{TREEDECOMP}^{in}$ but $\mathtt{TREEDECOMP}^{in}$ beats in turn HINGE. This result evidences that $\mathtt{TREEDECOMP}^{in}$ behaves significantly different than $\mathtt{TREEDECOMP}^{optd}$, given that this latter method strongly generalizes HINGE.
- ▷ We compare $\mathtt{TREEDECOMP}^{in}$ with $\mathtt{TREEDECOMP}^{optd}$. It turns out that there are CSP classes that are tractable according to $\mathtt{TREEDECOMP}^{optd}$ but are not tractable according to $\mathtt{TREEDECOMP}^{in}$. However, we show that $\mathtt{TREEDECOMP}^{optd}$ does not strongly generalize $\mathtt{TREEDECOMP}^{in}$. Indeed, there are classes of CSPs whose incidence-graph treewidth is bounded by a constant k, but the width of some optimal reduct of the dual graph is exponentially larger. Thus, even if such classes are tractable, their evaluation can be more efficient by using $\mathtt{TREEDECOMP}^{in}$. It follows that either of these methods may be useful for some kind of CSP instances, and hence there is no definitely better choice between them, even though $\mathtt{TREEDECOMP}^{optd}$ seems a bit more powerful.
- ▷ We show that the above relationships between $\mathtt{TREEDECOMP}^{in}$ with $\mathtt{TREEDECOMP}^{optd}$ hold on any pair of binary decomposition methods applied to the dual and the incidence graph, respectively. Therefore, either encodings turn out to be useful for some kind of CSP instance.
- ▷ We exploit the connections between $\mathtt{TREEDECOMP}^{in}$, $\mathtt{TREEDECOMP}^{optd}$, and weakQUERYDECOMP in order to further analyze the properties of the tree decomposition method, when applied on dual and incidence graphs. We show that any width-k decomposition of the incidence graph can be transformed into a weak query decomposition having width at most 2^{k+1} and, hence, into a tree decomposition of the dual graph whose width is bounded by $2 \times 2^{k+1}$. In fact, we show that $\mathtt{TREEDECOMP}^{optd}$ weakly generalizes $\mathtt{TREEDECOMP}^{in}$, thereby exactly capturing the relationship between the two notions.

In the third part of the paper, we consider the primal-graph representation, and we compare it with the various methods on dual and incidence graphs we have discussed earlier. Specifically,

- ▷ We consider the problem of comparing the tree decomposition method on the primal graph ($\texttt{TREEDECOMP}^{p}$) with the same method on the incidence graph ($\texttt{TREEDECOMP}^{in}$), raised in [18]. Indeed, we formalize and adapt some of the hints in [18] to formally show that $\texttt{TREEDECOMP}^{in}$ strongly generalizes $\texttt{TREEDECOMP}^{p}$.
- ▷ We show that no method applied to the dual graph is definitively superior to any method applied to the primal graph, precisely as in the case of the comparison between dual-graph and incidence-graph encodings. In particular, we show that $\texttt{TREEDECOMP}^{optd}$ weakly generalizes $\texttt{TREEDECOMP}^{p}$, $\texttt{CUTSET}^{p}$, and $\texttt{BICOMP}^{p}$.

Finally, we consider the problem of deciding whether any of the binary methods studied in the paper is superior to some of the above mentioned hypergraph-based decomposition methods. With this respect,

- ▷ We first focus on the (hypergraph-based) query decomposition [3] method (short: QUERYDECOMP), and we show that QUERYDECOMP strongly generalizes the TREEDECOMP method independent of whether this latter is applied to the primal, the dual, or the incidence graph. Then, since all the tractable hypergraph-based decomposition methods that we consider are defined as to generalize QUERYDECOMP, we conclude that each of them, in its turn, strongly generalizes $\texttt{TREEDECOMP}^{p}$, $\texttt{TREEDECOMP}^{optd}$, and $\texttt{TREEDECOMP}^{in}$.
- ▷ Finally, we shed some light on the big difference between $\texttt{TREEDECOMP}^{optd}$ (which emerged in our analysis as the more powerful method on binary encodings) and the hypergraph-based methods. To this end, we exploit again the weakQUERYDECOMP method (that approximates $\texttt{TREEDECOMP}^{optd}$), which is a special case of QUERYDECOMP.

1.3. The overall picture

A summary of our results is illustrated in Fig. 4, where QUERYDECOMP is depicted in gray because it is not a tractable notion [12]. Three kinds of arrows, dotted, dashed, and solid, are used to denote the relationships of generalization, weak generalization, and strong generalization, respectively. Note that this picture is *complete*. For instance, strong generalization is a transitive relationship, and thus a path of solid lines from $\texttt{D}_1$ to $\texttt{D}_2$ implicitly means that $\texttt{D}_2$ strongly generalizes $\texttt{D}_1$. Moreover, if $\texttt{D}_2$ neither strongly nor weakly generalizes $\texttt{D}_1$, then it is actually shown in the paper (or, it is known from the literature) that $\texttt{D}_2$ and $\texttt{D}_1$ are incomparable, and thus no arrow connects them in the picture.

Observe that the power of all the binary methods is essentially bounded by weakQUERYDECOMP, whereas QUERYDECOMP is generalized by the other hypergraph-based methods. Moreover, the picture evidences that any binary method applied to whatever kind of encoding is strongly generalized by QUERYDECOMP. Then, it can be argued that the superior power of hypergraph-based methods precisely lies in the capability of jointly using sets of hyperedges to decompose the CSP instance at hand, a capability that is in fact missing in weakQUERYDECOMP (and below). As for the relationships among binary methods, it emerges that $\texttt{TREEDECOMP}^{in}$ is better than any method applied to the primal graph. However, note that $\texttt{TREEDECOMP}^{optd}$ weakly generalizes all these graph methods, and it strongly generalizes $\texttt{BICOMP}^{optd}$.

Organization. The rest of the paper is organized as follows. In Section 2, we introduce some preliminaries on decomposition methods and we define the criteria according to which these methods will be formally compared. Then, in Section 3, we start our investigation with the dual-graph encoding of non-binary CSPs. Section 4 contains the comparison of the method $\texttt{TREEDECOMP}^{in}$ with both HINGE and $\texttt{TREEDECOMP}^{optd}$, while in Section 5 we complete the picture by investigating the power of decomposition methods working on primal graphs. Section 6 reports the formal comparison of binary methods with hypergraph-based decomposition methods. Finally, in Section 7, we draw our conclusions.

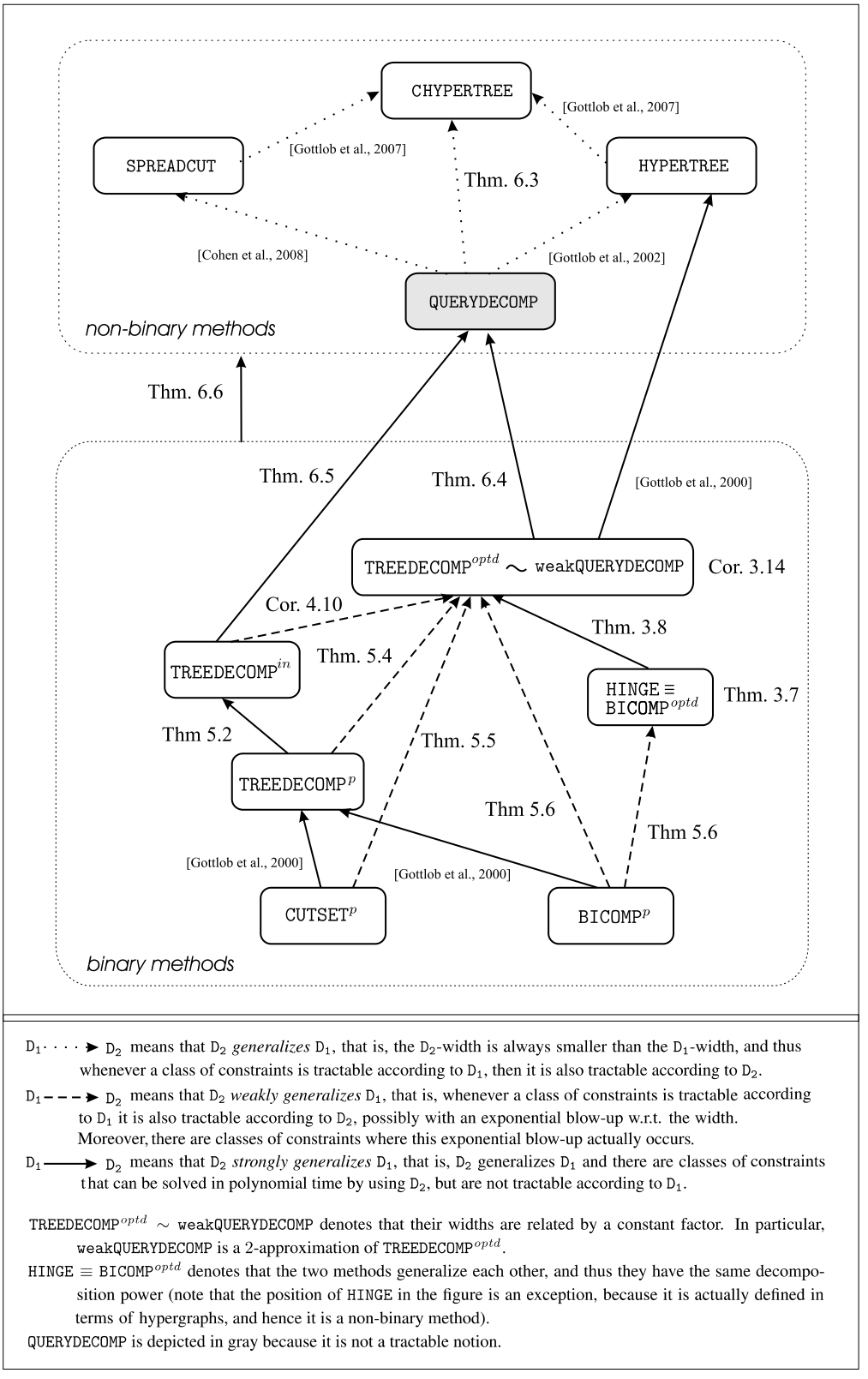
2. Structural decomposition methods

In this section, we introduce the various decomposition methods and comparison criteria that will be the subject of our research in this paper.

2.1. Tractable classes of constraint satisfaction problems

It is well known that CSPs with *acyclic* constraint hypergraphs are polynomially solvable [6]. In fact, most of the known structural properties that lead to tractable CSP classes are (explicitly or implicitly) based on some generalization of acyclicity. In particular, each structural decomposition method D 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. This (possibly infinite) set of CSPs is called the *tractability class* of D w.r.t. k, and is denoted by $C(\texttt{D}, k)$. Actually, for the sake of simplicity and whenever no confusion arises, we shall often use also $C(\texttt{D}, k)$ to denote the class of hypergraphs associated with those instances.

We next recall the definitions of some relevant concepts of widths and structural decomposition methods (see, e.g., [10])—further hypergraph-based methods will be discussed in Section 6.



$D_1 \cdots\!\blacktriangleright\ D_2$ means that D_2 *generalizes* D_1, that is, the D_2-width is always smaller than the D_1-width, and thus whenever a class of constraints is tractable according to D_1, then it is also tractable according to D_2.

$D_1 \dashrightarrow D_2$ means that D_2 *weakly generalizes* D_1, that is, whenever a class of constraints is tractable according to D_1 it is also tractable according to D_2, possibly with an exponential blow-up w.r.t. the width. Moreover, there are classes of constraints where this exponential blow-up actually occurs.

$D_1 \longrightarrow D_2$ means that D_2 *strongly generalizes* D_1, that is, D_2 generalizes D_1 and there are classes of constraints that can be solved in polynomial time by using D_2, but are not tractable according to D_1.

TREEDECOMP^{optd} $\sim$ weakQUERYDECOMP denotes that their widths are related by a constant factor. In particular, weakQUERYDECOMP is a 2-approximation of TREEDECOMP^{optd}.

HINGE $\equiv$ BICOMP^{optd} denotes that the two methods generalize each other, and thus they have the same decomposition power (note that the position of HINGE in the figure is an exception, because it is actually defined in terms of hypergraphs, and hence it is a non-binary method).

QUERYDECOMP is depicted in gray because it is not a tractable notion.

Fig. 4. Summary of the results.

CUTSET [6]. Let $G = (V, E)$ be a graph. A *cycle cutset* of G is a set $S \subseteq V$ such that the subgraph of G (vertex-)induced by $V - S$ is acyclic. That is, after deleting the vertices in S, the graph of G becomes acyclic. The *cutset width* of G is 1 if G is acyclic; otherwise, it is the minimum cardinality over all its possible cycle cutsets.

TREEDECOMP [20]. 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 labelling function associating to each vertex $p \in N$ a set of nodes $\chi(p) \subseteq V$, 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—this latter condition is usually known as the *connectedness condition*. 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.

BICOMP [8]. 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. The *biconnected width* of G is the cardinality of the largest *biconnected component* of G.

HINGE [15,16]. Let $\mathcal{H}$ be a hypergraph. For each set of hyperedges $H \subseteq \mathcal{E}(\mathcal{H})$, with a slight abuse of notation, $\mathcal{N}(H)$ will from now on denote the set of nodes $\bigcup_{h \in H} h$. Let $H \subseteq \mathcal{E}(\mathcal{H})$, and $F \subseteq \mathit{edges}(\mathcal{H}) - H$. Then, F is called *connected with respect to* H if, for any two edges $e, f \in F$, there exists a sequence $e_1, \ldots, e_n$ of edges in F such that (i) $e_1 = e$; (ii) for $i = 1, \ldots, n-1$, $e_i \cap e_{i+1}$ is not contained in $\bigcup_{h \in H} h$; and (iii) $e_n = f$. Let $C_1, \ldots, C_m$ be the maximal connected subsets of $\mathcal{E}(\mathcal{H}) - H$ with respect to H. Then, H is a *hinge* if, for $i = 1, \ldots, m$, there exists an edge $h_i \in H$ such that $\mathcal{N}(C_i) \cap \mathcal{N}(H) \subseteq h_i$. A hinge is *minimal* if it does not contain any other hinge. The hinge width of $\mathcal{H}$ is defined as the cardinality of the largest *minimal hinge* of $\mathcal{H}$.

QUERYDECOMP [3]. The notion of query decomposition was originally conceived in the database context of *conjunctive query optimization*. Following [12], we actually provide here a slight variation of the original definition of query decomposition, where the tree vertices are labelled by hyperedges only. Indeed, from the results in [12], we know that this simplified notion is equivalent to the original one.

A *query decomposition* of a hypergraph $\mathcal{H}$ is a pair $\langle T, \lambda \rangle$, where $T = (N, E)$ is a tree, and λ is a labelling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$, such that the following conditions hold: (1) for each hyperedge h of $\mathcal{H}$, there exists $v \in N$ such that $h \in \lambda(v)$; (2) for each hyperedge h of $\mathcal{H}$, the set $\{v \in N \mid h \in \lambda(v)\}$ induces a (connected) subtree of T; (3) for each pair of vertices v and v' of T, for each pair of hyperedges $h \in \lambda(v)$ and $h' \in \lambda(v')$, and for each vertex v'' in the path connecting v and v' in T, $h \cap h' \subseteq \mathcal{N}(\lambda(v''))$. The *width* of the query decomposition $\langle T, \lambda \rangle$ is $\max_{v \in N} |\lambda(p)|$. The *query width* of $\mathcal{H}$ is the minimum width over all its query decompositions.

We point out that QUERYDECOMP and HINGE are the only of the above decomposition methods defined to be directly applied to the constraint hypergraph. All the other methods are, instead, binary; hence, we shall consider their application to some binary encodings of the constraint hypergraph. In particular, we find in the literature applications of CUTSET to the primal graph (CUTSET^p), of BICOMP to the primal and the dual graph (BICOMP^p and BICOMP^d), and of TREEDECOMP to the primal, the dual, and the incidence graph (TREEDECOMP^p, TREEDECOMP^d and TREEDECOMP^{in}).

2.2. Comparison criteria

Any pair of decomposition methods $\mathtt{D}_1$ and $\mathtt{D}_2$ can be compared according to their ability of identifying tractable classes of CSPs. This approach has firstly been formalized in [10], where the following criteria have been proposed:

Generalization. $\mathtt{D}_2$ *generalizes* $\mathtt{D}_1$ ($\mathtt{D}_1 \preceq \mathtt{D}_2$) if there exists a constant δ such that for each $k > 0$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, k + \delta)$. In practical terms, this means that whenever a class of constraints is tractable according to method $\mathtt{D}_1$, it is also tractable according to $\mathtt{D}_2$.

Beating. $\mathtt{D}_2$ *beats* $\mathtt{D}_1$, denoted by $\mathtt{D}_2 \rhd \mathtt{D}_1$, if there exists an integer $k > 0$ and a set $\mathcal{C}$ of instances such that $\mathcal{C} \subseteq C(\mathtt{D}_2, k)$, and $\mathcal{C} \nsubseteq C(\mathtt{D}_1, m)$ for any $m > 0$. Hence, some classes of problems are tractable according to $\mathtt{D}_2$ but not according to $\mathtt{D}_1$. For such classes, using $\mathtt{D}_2$ is thus better than using $\mathtt{D}_1$.

Strong generalization. $\mathtt{D}_2$ *strongly generalizes* $\mathtt{D}_1$, denoted by $\mathtt{D}_1 \lessdot\lessdot \mathtt{D}_2$, if $\mathtt{D}_2$ generalizes $\mathtt{D}_1$ and $\mathtt{D}_2$ beats $\mathtt{D}_1$. This means that $\mathtt{D}_2$ is really more powerful, given that whenever $\mathtt{D}_1$ guarantees polynomial runtime for constraint solving, then also $\mathtt{D}_2$ guarantees tractable constraint solving, but there are classes of constraints that can be solved in polynomial time by using $\mathtt{D}_2$, but are not tractable according to $\mathtt{D}_1$.

Equivalence. $\mathtt{D}_1$ and $\mathtt{D}_2$ are *equivalent*, denoted by $\mathtt{D}_1 \equiv \mathtt{D}_2$, if $\mathtt{D}_1$ generalizes $\mathtt{D}_2$ and $\mathtt{D}_2$ generalizes $\mathtt{D}_1$. Intuitively, this means that these two methods do not differ significantly from each other.

Finally, the decomposition methods $\mathtt{D}_1$ and $\mathtt{D}_2$ are *strongly incomparable* if both $\mathtt{D}_1 \rhd \mathtt{D}_2$ and $\mathtt{D}_2 \rhd \mathtt{D}_1$.

Next, we state a few simple properties about the notions of generalization, beating, and strong generalization, which will be exploited in the paper.[3]

Proposition 2.1. *Let* $\mathtt{D}_1$, $\mathtt{D}_2$, *and* $\mathtt{D}_3$ *be three decomposition methods. Then,*

(1) *If* $\mathtt{D}_2 \rhd \mathtt{D}_1$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_3 \rhd \mathtt{D}_1$;
(2) *If* $\mathtt{D}_1 \preceq \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_1 \preceq \mathtt{D}_3$; *and*
(3) *If* $\mathtt{D}_1 \prec\!\!\prec \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_1 \prec\!\!\prec \mathtt{D}_3$.

2.3. Weak generalization

The above comparison criteria have been used in [10] to completely classify the power of the methods $\mathtt{CUTSET}^p$, $\mathtt{BICOMP}^p$, $\mathtt{TREEDECOMP}^p$, HINGE, HYPERTREE, and, partially, of the method $\mathtt{TREEDECOMP}^d$. Some of the results obtained in [10] emerge from Fig. 4 (see the citations on the arc labels).

In fact, in order to extend the analysis of [10] to the dual and the incidence-graph encodings, we need to introduce another criterium that is finer than the notions of generalization and beating. Indeed, these criteria are not appropriate to capture those scenarios where a method $\mathtt{D}_2$ does not generalize $\mathtt{D}_1$, but it is still such that any decomposition of width k w.r.t. $\mathtt{D}_1$ can be mapped into a decomposition of width k' w.r.t. $\mathtt{D}_2$, where k' is a natural number that depends on k only.

Formally, let f be a non-decreasing function from (positive) natural numbers to (positive) natural numbers. Then, $\mathtt{D}_2$ f-*generalizes* $\mathtt{D}_1$ ($\mathtt{D}_1 \preceq_f \mathtt{D}_2$) if, for each $k > 1$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, f(k))$ holds. Thus, for any given hypergraph $\mathcal{H}$, if there exists a decomposition for $\mathcal{H}$ of width k w.r.t. $\mathtt{D}_1$, then there exists a decomposition for $\mathcal{H}$ of width $f(k)$ w.r.t. $\mathtt{D}_2$.[4]

The following are simple, yet relevant properties of f-generalizations.

Proposition 2.2. *Let* $\mathtt{D}_1$ *and* $\mathtt{D}_2$ *be two decomposition methods. Then,*

(1) $\mathtt{D}_1 \preceq \mathtt{D}_2$ *implies that there is a function* f *such that* $\mathtt{D}_1 \preceq_f \mathtt{D}_2$; *and*
(2) $\mathtt{D}_1 \preceq_f \mathtt{D}_2$ *implies that* $\mathtt{D}_1 \rhd \mathtt{D}_2$ *and* $\mathtt{D}_2 \prec\!\!\prec \mathtt{D}_1$ *do not hold.*

Since we are mainly interested in upper bounds that are also strict, we next define the new notion of f-beating, which intuitively means that, for some class of instances, such a function $f(\cdot)$ provides a lower bound for the relationship between the widths according to the given methods.

Let $\mathcal{C}$ be a countably infinite class of hypergraphs where every hypergraph $\mathcal{H}_i$ is identified by its associated natural number i, and let $\mathtt{D}$ be any decomposition method. Define $w^{\mathcal{C}}_{\mathtt{D}}(\cdot)$ as the width function such that $w^{\mathcal{C}}_{\mathtt{D}}(n)$ is the $\mathtt{D}$-width of hypergraph $\mathcal{H}_n$ in $\mathcal{C}$. Let $\mathtt{D}_1$ and $\mathtt{D}_2$ be two decomposition methods. We say that $\mathtt{D}_1$ f-*beats* $\mathtt{D}_2$ if there exists a countably infinite class $\mathcal{C}$ of hypergraphs such that $w^{\mathcal{C}}_{\mathtt{D}_2}(n)$ is $\Omega(f(w^{\mathcal{C}}_{\mathtt{D}_1}(n)))$.

An important case of f-generalization and f-beating occurs when using $\mathtt{D}_2$ instead of $\mathtt{D}_1$ leads to an exponential blow-up in the width. From now on, $exp_\delta(\cdot)$ will denote the exponential function $exp_\delta(n) = 2^{\lfloor n^\delta \rfloor}$, where $\delta > 0$ is some fixed rational number. For instance, $exp_{\frac{1}{2}}(n) = 2^{\lfloor \sqrt{n} \rfloor}$. Moreover, whenever we say that $\mathtt{D}_1$ *exp*-generalizes (resp., *exp*-beats) $\mathtt{D}_2$, we mean that there exists some fixed rational $\delta > 0$ such that $\mathtt{D}_1$ exp_δ-generalizes (resp., exp_δ-beats) $\mathtt{D}_2$.

Definition 2.3 *(Weak generalization).* Let $\mathtt{D}_1$ and $\mathtt{D}_2$ be two decomposition methods. Then, $\mathtt{D}_2$ *weakly generalizes* $\mathtt{D}_1$, denoted by $\mathtt{D}_1 \precsim \mathtt{D}_2$, if

(i) $\mathtt{D}_2$ beats $\mathtt{D}_1$;
(ii) $\mathtt{D}_2$ *exp*-generalizes $\mathtt{D}_1$; and
(iii) $\mathtt{D}_1$ *exp*-beats $\mathtt{D}_2$.

Thus, $\mathtt{D}_2$ weakly generalizes $\mathtt{D}_1$ intuitively means that $\mathtt{D}_2$ is slightly more powerful than $\mathtt{D}_1$. Indeed, there are classes of constraints that can be solved in polynomial time by using $\mathtt{D}_2$ but that are not tractable according to $\mathtt{D}_1$—cf. condition (i) above. And, any class of constraints that is solvable in polynomial time with $\mathtt{D}_1$ is solvable in polynomial time by applying the method $\mathtt{D}_2$ as well, possibly with an exponential blow-up w.r.t. the width—cf. (ii). Moreover, we know that this exponential blow-up actually occurs for some class of instances—cf. (iii), where using $\mathtt{D}_1$ should be thus preferable to $\mathtt{D}_2$. In practice this means that in this case there is no clear winner between the two methods, and the actual choice will depend on the specific class of problems to be solved. Of course, weak and strong generalizations are mutually exclusive, because of the *exp*-beating condition. Some further properties of weak generalizations are stated below.

[3] Proofs in this preliminary section are rather simple and hence are omitted. However, they are reported in Appendix A, for the sake of completeness.

[4] Note that the f-relationship is required to hold for any $k > 1$, instead of $k > 0$ as in the case of plain generalization [10]. Indeed, f is here a generic function, and it is convenient, for (clear enough) technical reasons, to avoid that the value $k = 1$ occurs as the base in possible exponential expressions. This is not a substantial change, since $C(\mathtt{D}_1, 1) \subseteq C(\mathtt{D}_1, 2) \subseteq C(\mathtt{D}_2, f(2))$ holds.

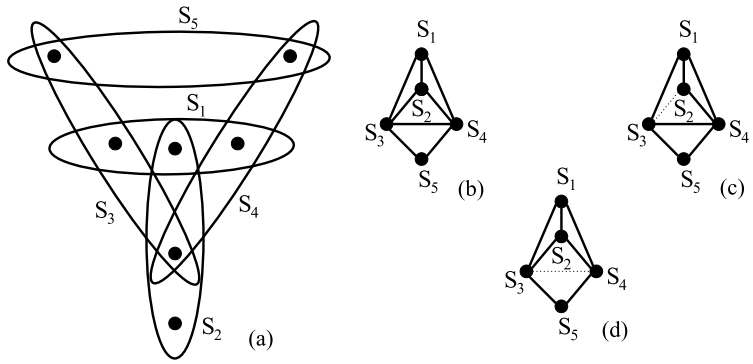


Fig. 5. (a) A hypergraph, (b) its dual graph, (c) a reduct having treewidth 2, and (d) a reduct having treewidth 3.

Proposition 2.4. *Let* $\mathtt{D}_1$, $\mathtt{D}_2$, *and* $\mathtt{D}_3$ *be three decomposition methods. Then,*

(1) $\mathtt{D}_1$ *does not weakly generalize* $\mathtt{D}_1$, *i.e.,* $\precsim$ *is antireflexive*;
(2) *If* $\mathtt{D}_1 \precsim \mathtt{D}_2$, *then* $\mathtt{D}_2 \precsim \mathtt{D}_1$ *does not hold, i.e.,* $\precsim$ *is antisymmetric*; *and*
(3) *If* $\mathtt{D}_1 \precsim \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$ (*or,* $\mathtt{D}_1 \preceq \mathtt{D}_2$ *and* $\mathtt{D}_2 \precsim \mathtt{D}_3$), *then* $\mathtt{D}_3$ *exp-generalizes* $\mathtt{D}_1$.

3. Dual-graph representation

In this section, we formalize the notion of reduct of the dual graph, i.e., we define how the dual graph $dual(\mathcal{H})$ of a hypergraph $\mathcal{H}$ can be simplified by safely removing some edges. Based on this notion, we then investigate the power of some binary decomposition techniques defined on (some optimal reduct of) the dual graph.

3.1. Reducts and basic properties

Definition 3.1 *(Reduct).* Let $dual(\mathcal{H}) = (N, E)$ be the dual graph of a hypergraph $\mathcal{H}$. A *reduct* G' of $dual(\mathcal{H})$ is a graph (N', E') satisfying the following three conditions:

(1) $N' = N$;
(2) $E' \subseteq E$; and
(3) for each edge $\{h, h'\}$ belonging to $E - E'$, there exists in G' a path $h = h_1, \ldots, h_n = h'$, such that all variables in $h \cap h'$ are included in $h_i \cap h_{i+1}$, for each $1 \leqslant i < n$. That is, if all the variables shared by h and h' occur in some other path between h and h', then the edge connecting them can be deleted from the dual graph.

It has been observed in the literature (see, e.g., [15]) that any CSP instance can be solved by applying binary methods originally conceived for the dual graph (such as $\mathtt{BICOMP}^d$ and $\mathtt{TREEDECOMP}^d$) to an arbitrary reduct of the dual graph of the given instance. This may be a great advantage, because the various edge deletions might significantly simplify the intricacy of the dual graph (and, hence, its width). However, different removal choices may lead to different widths, in general. Consider, for instance, the hypergraph shown in Fig. 5(a) and the associated dual graph in (b). Then, it is easy to see that the graphs reported in (c) and (d) are both reducts of this dual graph; however, the reduct in (c) has treewidth 2, while the reduct in (d) has treewidth 3. In this case, if we use $\mathtt{TREEDECOMP}^d$, it is convenient to solve the CSP instance over the reduct in (c). Note that this conclusion depends on the technique we want to use with that graph. It is possible that, for a different dual-graph technique $\mathtt{D}$, another reduct has a smaller $\mathtt{D}$-width than the reduct (c), and it is thus preferable w.r.t. $\mathtt{D}$.

Therefore, in order to get the maximum power from a binary decomposition method $\mathtt{D}$ applied to the dual graph $dual(\mathcal{H})$, we define the $\mathtt{D}^{optd}$-width of $dual(\mathcal{H})$ as the minimum $\mathtt{D}$-width w over all its possible reducts. Any reduct of $dual(\mathcal{H})$ whose $\mathtt{D}$-width is w is said an *optimal reduct*. Moreover, the method $\mathtt{D}$ applied to any optimal reduct of $dual(\mathcal{H})$ is denoted by $\mathtt{D}^{optd}$.

Note that applying two graph methods $\mathtt{D}_1$ and $\mathtt{D}_2$ to the respective optimal reducts does not alter their relative power, i.e., ${\mathtt{D}_1}^{optd}$ remains preferable to ${\mathtt{D}_2}^{optd}$, whenever $\mathtt{D}_1$ is preferable to $\mathtt{D}_2$. This is formalized below, after a useful construction about irreducible dual graphs.

Lemma 3.2. *For any graph* G, *there exists a hypergraph* $\mathcal{H}(G)$ *such that*:

- G *is isomorphic to* $dual(\mathcal{H}(G))$; *and*
- $dual(\mathcal{H}(G))$ *is irreducible.*

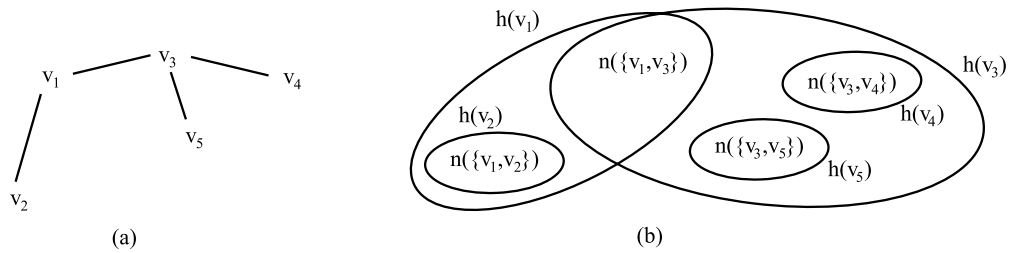


Fig. 6. Example construction in Lemma 3.2: (a) graph G, and (b) the corresponding hypergraph $\mathcal{H}(G)$.

Proof. Let G be a graph and let $\mathcal{H}(G)$ be the hypergraph defined as follows: for each edge q in G, $\mathcal{H}(G)$ contains the node $n(q)$; for each node v of G, $\mathcal{H}(G)$ contains the hyperedge $h(v)$ consisting of all those nodes $n(q)$ such that $v \in q$; there are no further nodes or hyperedges in $\mathcal{H}(G)$. An example construction is illustrated in Fig. 6.

Note that, in the dual graph $dual(\mathcal{H}(G))$, there is an edge between $h(v_1)$ and $h(v_2)$ if and only if a node $n(q)$ occurs in $h(v_1) \cap h(v_2)$, i.e., if and only if $v_1 \in q$ and $v_2 \in q$, i.e., if and only if $q = \{v_1, v_2\}$. Moreover, an edge in $dual(\mathcal{H}(G))$ between $h(v_1)$ and $h(v_2)$ cannot be removed, since $n(q)$ is not contained in any further hyperedge of $\mathcal{H}(G)$, by construction. □

Theorem 3.3. *Let* $\mathtt{D}_1$ *and* $\mathtt{D}_2$ *be two graph decomposition methods. Then,*

(1) $\mathtt{D}_2 \preceq \mathtt{D}_1$ *implies* ${\mathtt{D}_2}^{optd} \preceq {\mathtt{D}_1}^{optd}$;
(2) $\mathtt{D}_1 \rhd \mathtt{D}_2$ *implies* ${\mathtt{D}_1}^{optd} \rhd {\mathtt{D}_2}^{optd}$; *and*
(3) $\mathtt{D}_2 \lessdot\!\lessdot \mathtt{D}_1$ *implies* ${\mathtt{D}_1}^{optd} \lessdot\!\lessdot {\mathtt{D}_2}^{optd}$.

Proof.

(1) Recall that $\mathtt{D}^{optd}$ represents the method $\mathtt{D}$ that, given a hypergraph $\mathcal{H}$, is applied to a reduct of the dual graph having the minimum width (w.r.t. $\mathtt{D}$) over all the possible reducts of $dual(\mathcal{H})$. We denote such an optimal reduct by $opt{\mathtt{D}}^d(\mathcal{H})$. Since $\mathtt{D}_2 \preceq \mathtt{D}_1$, there is by definition a constant δ such that, for each k, $C(\mathtt{D}_2, k) \subseteq C(\mathtt{D}_1, k+\delta)$. We claim that for the same constant δ it also holds that, for each k, $C({\mathtt{D}_2}^{optd}, k) \subseteq C({\mathtt{D}_1}^{optd}, k+\delta)$. To show that the claim holds, consider an instance $\mathcal{H}$ whose width w.r.t. ${\mathtt{D}_2}^{optd}$ is $k' \leqslant k$, i.e., $\mathcal{H}$ belongs to $C({\mathtt{D}_2}^{optd}, k)$.
By definition of method ${\mathtt{D}_2}^{optd}$, k' is the width w.r.t. $\mathtt{D}_2$ of $opt{\mathtt{D}_2}^d(\mathcal{H})$. Since $\mathtt{D}_2 \preceq \mathtt{D}_1$, the width w.r.t. $\mathtt{D}_1$ of the graph $opt{\mathtt{D}_2}^d(\mathcal{H})$ is bounded by $k' + \delta$, and hence by $k + \delta$. Thus, there is a reduct of $dual(\mathcal{H})$ whose $\mathtt{D}_1$-width is at most $k + \delta$. It follows that $k + \delta$ is an upper bound for the width of the optimal reducts of $dual(\mathcal{H})$ w.r.t. $\mathtt{D}_1$. Then, $\mathcal{H}$ belongs to $C({\mathtt{D}_1}^{optd}, k + \delta)$, as well.
(2) Since $\mathtt{D}_1$ beats $\mathtt{D}_2$, there exists an integer k and a set of graphs $\mathcal{C}_1 \subseteq C(\mathtt{D}_1, k)$ such that $\mathcal{C}_1 \nsubseteq C(\mathtt{D}_2, m)$ for any $m > 0$. Then, let $m' > 0$ be a natural number, and $G_{m'}$ be a graph in $\mathcal{C}_1$ that does not belong to $C(\mathtt{D}_2, m')$. From Lemma 3.2, there exists a hypergraph $\mathcal{H}(G_{m'})$ such that $G_{m'}$ is isomorphic to $dual(\mathcal{H}(G_{m'}))$, and $dual(\mathcal{H}(G_{m'}))$ is the best possible reduct.
Then, observe that the application of the graph method $\mathtt{D}_1$ (resp., $\mathtt{D}_2$) to the dual graph $dual(\mathcal{H}(G_{m'}))$ coincides with the application of ${\mathtt{D}_1}^{optd}$ (resp., ${\mathtt{D}_2}^{optd}$) to $\mathcal{H}(G_{m'})$, because no reduction of its dual graph is possible. Since $dual(\mathcal{H}(G_{m'}))$ is isomorphic to $G_{m'}$, applying such graph techniques to $dual(\mathcal{H}(G_{m'}))$ is clearly the same as working with $G_{m'}$. It follows that $\mathcal{H}(G_{m'})$ belongs to $C({\mathtt{D}_1}^{optd}, k)$, but $\mathcal{H}(G_{m'})$ does not belong to $C({\mathtt{D}_2}^{optd}, m')$. Therefore, the class $\mathcal{C}_2 = \{\mathcal{H}(G) \mid G \in \mathcal{C}_1\}$ is in $C(\mathtt{D}_1, k)$, but it is not included in $C({\mathtt{D}_2}^{optd}, m)$, for any $m > 0$. Thus, ${\mathtt{D}_1}^{optd} \rhd {\mathtt{D}_2}^{optd}$.
(3) Immediately follows from the previous two points, by definition of $\lessdot\!\lessdot$. □

In the light of the above characterizations, we conclude that all the results proven in [10] for the primal graph encoding (and hence implicitly for binary CSPs) hold, in fact, when these methods are applied to the optimal reduct of the dual graph, too. As an example, given that $\mathtt{BICOMP} \lessdot\!\lessdot \mathtt{TREEDECOMP}$ is shown to hold, we get the following.

Corollary 3.4. $\mathtt{BICOMP}^{optd} \lessdot\!\lessdot \mathtt{TREEDECOMP}^{optd}$.

In particular, since TREEDECOMP strongly generalizes all the other decomposition methods tailored for binary CSPs [10], $\mathtt{TREEDECOMP}^{optd}$ is definitively the best method to be applied to non-binary CSPs among these methods that work on dual-graph encodings.

However, recall that all the known algorithms for computing a k-bounded tree decomposition of a graph are exponential in k (even if they are polynomial for any fixed constant k), while computing the biconnected components of a graph is a linear task, independently of their width [15]. This latter technique can be therefore very useful if the size of the structure or the bound k are large, and more powerful methods like TREEDECOMP are too expensive. Yet, in order to practically

exploit BICOMP on the dual graph, it remains to assess whether an optimal reduct w.r.t. this method can also be efficiently computed, which is the question we shall face next.

3.2. Biconnected components versus hinges

In [15], it has been shown that the HINGE method generalizes BICOMP applied to any reduct of the dual graph. However, in the same paper, it is observed that a fine comparison between the two methods is quite difficult, as there is no obvious way to find a suitable reduct of the dual graph to keep the biconnected width small. Here, we solve this question by showing that it is always possible to find a reduct of the dual graph whose biconnected width is equal to the hinge width of $\mathcal{H}$.

First, we recall from [5] the definition of hinge-tree decomposition, just slightly adapted to the notations used in this paper. A *hinge-tree decomposition* of a hypergraph $\mathcal{H}$ is a pair $\langle T, \lambda\rangle$, where $T = (N, E)$ is a tree, and λ is a labelling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$, such that the following conditions hold: (1) for each hyperedge h of $\mathcal{H}$, there exists $v \in N$ such that $h \in \lambda(v)$; (2) for each node b in $\mathcal{N}(\mathcal{H})$, the set $\{v \in N \mid \exists h \in \lambda(v),\ b \in h\}$ induces a (connected) subtree of T; (3) for each pair of vertices v and v' of T, there is an edge $h \in \lambda(v)$ such that $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v')) \subseteq h$. The *width* of the hinge-tree decomposition $\langle T, \lambda\rangle$ is $\max_{v\in N} |\lambda(v)|$. The *hinge width* of $\mathcal{H}$ coincides with the minimum width over all its hinge-tree decompositions [5].

In order to establish the main result of this section, we need to preliminary state a few properties of hinge-tree decompositions of *reduced* hypergraphs (that is, of hypergraphs such that there is no hyperedge properly contained in any other hyperedge) and of *connected* hypergraphs (thats is, of hypergraphs such that each pair of nodes is connected via a path in the primal graph).

Lemma 3.5. *Let $\mathcal{H}$ be a hypergraph, let $\langle T, \lambda\rangle$ be a hinge-tree decomposition of $\mathcal{H}$, and let v and v' be two distinct vertices of T.*

(1) *If $\mathcal{H}$ is reduced, then $\lambda(v) \cap \lambda(v') \subseteq \lambda(v'')$ holds for each vertex v'' in the path connecting in T the vertices v and v';*
(2) *If $\mathcal{H}$ is connected and $|\lambda(v)| \geqslant 2$, then for each hyperedge $h \in \lambda(v)$ there is a hyperedge $\bar{h} \in \lambda(v) - \{h\}$ such that $h \cap \bar{h} \neq \emptyset$. That is, there are no isolated hyperedges in λ labels.*

Proof.

(1) Let $v = v_1, \ldots, v_n = v'$ (with $n > 1$) be the path in T connecting the two distinct vertices v and v'. Let $\bar{h}$ be a hyperedge in $\lambda(v) \cap \lambda(v')$. By condition (2) in the definition of hinge-tree decompositions, for each $1 \leqslant i < n$, $\bar{h} \subseteq \mathcal{N}(\lambda(v_i)) \cap \mathcal{N}(\lambda(v_{i+1}))$. Moreover, by condition (3) in the same definition, for the two vertices v_i and v_{i+1} of T, there is a hyperedge $h_i \in \lambda(v_i)$ such that $\mathcal{N}(\lambda(v_i)) \cap \mathcal{N}(\lambda(v_{i+1})) \subseteq h_i$. Therefore, $\bar{h} \subseteq h_i$ holds. In fact, since $\mathcal{H}$ is a reduced hypergraph, the latter entails that $\bar{h} = h_i$ and, hence, that $\bar{h}$ occurs in $\lambda(v_i)$, for each $1 \leqslant i \leqslant n$.
(2) Assume, for the sake of contradiction, that the statement does not hold. Let v be a vertex such that $|\lambda(v)| \geqslant 2$, and let $h \in \lambda(v)$ be a hyperedge such that $h \cap \bar{h} = \emptyset$, for each $\bar{h} \in \lambda(v) - \{h\}$. Since $\mathcal{H}$ is connected, there is some hyperedge $h'' \in \mathcal{E}(\mathcal{H})$ such that $h'' \cap h \neq \emptyset$. From condition (1) in the definition of hinge-tree decompositions, there is a vertex v'' in T such that $h'' \in \lambda(v'')$. In particular, note that $v'' \neq v$ holds. Thus, let v' be the neighbor of v (possibly coinciding with v'') that belongs to the path in T from v to v''. From condition (2) in the definition of hinge-tree decompositions, $h \cap h'' \subseteq \mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v'))$ and, hence, $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v')) \cap h \neq \emptyset$ holds since $h'' \cap h \neq \emptyset$. Moreover, from condition (3) in the same definition, $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v'))$ should be included in some hyperedge of $\lambda(v)$. Because h has no intersection with any other hyperedge in $\lambda(v)$, and given that $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v')) \cap h \neq \emptyset$, h is the only hyperedge that may play such a role. We conclude that $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v')) \subseteq h$. It follows that there is no path in $\mathcal{H}$ from any node in $\mathcal{N}(\lambda(v'))$ to any node in $\mathcal{N}(\lambda(v) - \{h\})$, which contradicts the fact that $\mathcal{H}$ is connected. □

We are now in the position of characterizing the power of BICOMP^{optd}. We recall the notion of decomposition based on biconnected components given in [5]. Let G be a graph. A triple $\langle T, \chi, \lambda\rangle$ where $\langle T, \chi\rangle$ is a tree-decomposition of G and, for each vertex v of T, $\chi(v) = \mathcal{N}(\lambda(v))$, is said an *edge-defined decomposition* of G. Then, a *biconnected decomposition* of a graph G is an edge-defined decomposition $\langle T, \chi, \lambda\rangle$ of G such that the following additional condition hold: (4) for each pair of vertices v and v' of T, $|\chi(v) \cap \chi(v')| \leqslant 1$. The width of this decomposition is given by the maximum cardinality of the χ labelling over the vertices of T. A graph G has a width-k biconnected decomposition if and only if the biconnected width of G is at most k [5].

Lemma 3.6. HINGE $\preceq$ BICOMP^{optd}.

Proof. Let $\mathcal{H}'$ be a given hypergraph, and assume without loss of generality that $\mathcal{H}'$ is connected. Let $\mathcal{H}$ be the reduced hypergraph obtained by removing from $\mathcal{H}'$ any hyperedge h which is a (proper) subset of another hyperedge h'. It is easy to see that HINGE-*width*$(\mathcal{H})$ = HINGE-*width*$(\mathcal{H}')$ and BICOMP^{optd}-*width*$(\mathcal{H})$ = BICOMP^{optd}-*width*$(\mathcal{H}')$, that is, non-maximal hyperedges have no impact on the widths, according to these notions. For the sake of completeness, we observe that this is not always true for other notions that are not based on edge-defined decompositions, such as HYPERTREE.

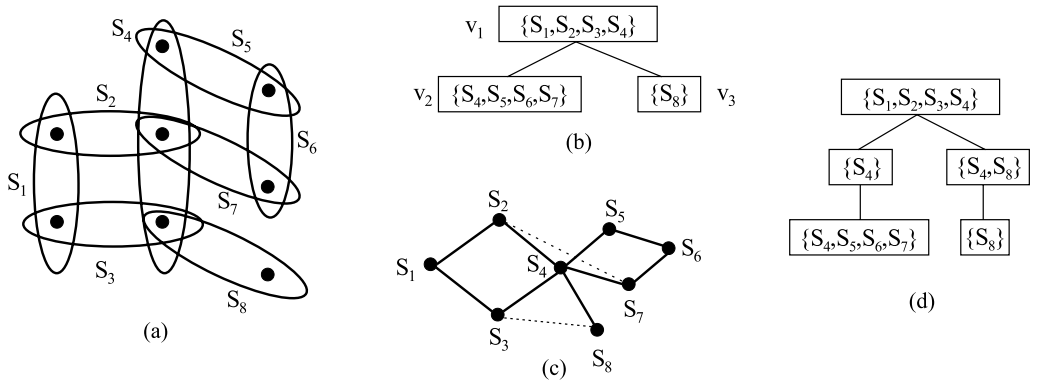


Fig. 7. (a) A hypergraph $\mathcal{H}$, (b) a hinge-tree decomposition T of $\mathcal{H}$, (c) the graph $G(T)$, and (d) a tree decomposition of $G(T)$.

Let $\langle T, \lambda\rangle$ be a width-k hinge-tree decomposition of $\mathcal{H}$. Based on T, we build a reduct $G(T)$ of $\mathit{dual}(\mathcal{H})$ having a width-k biconnected decomposition of $G(T)$.

For each (ordered) pair of adjacent vertices v and v' of T, let $h_{v,v'} \in \lambda(v)$ denote a hyperedge such that $\mathcal{N}(\lambda(v)) \cap \mathcal{N}(\lambda(v')) \subseteq h_{v,v'}$; note that $h_{v,v'}$ is well-defined due to condition (3) in the definition of hinge-tree decompositions. Then, let us build the graph $G(T)$ by removing from the dual graph $\mathit{dual}(\mathcal{H})$ each edge $\{h, h'\}$ such that the following conditions hold: (1) there is no vertex v of T such that $\{h, h'\} \subseteq \lambda(v)$, and (2) for each pair of adjacent vertices v and v' of T, $\{h, h'\} \neq \{h_{v,v'}, h_{v',v}\}$. As an example construction, Fig. 7 shows a hypergraph $\mathcal{H}$ in (a), a hinge-tree decomposition of $\mathcal{H}$ in (b), and the graph $G(T)$—built for $h_{v_1,v_2} = h_{v_2,v_1} = h_{v_1,v_3} = \{S_4\}$, and $h_{v_3,v_1} = \{S_8\}$—in (c). In particular, those edges of the dual graph that do not occur in $G(T)$ are depicted with dotted lines.

We now prove that $G(T)$ is indeed a reduct of $\mathit{dual}(\mathcal{H})$, i.e., that for each edge $\{h, h'\}$ of $\mathit{dual}(\mathcal{H})$ that does not occur in $G(T)$, condition (3) in Definition 3.1 is satisfied. To this end, given such an edge $\{h, h'\}$, by condition (1) in the definition of hinge-tree decompositions, there are two vertices v and v' of T such that $h \in \lambda(v)$ and $h' \in \lambda(v')$. Moreover, because of point (1) in the construction of $G(T)$, it must be the case that $v \neq v'$, for otherwise $\{h, h'\}$ would be an edge of $G(T)$. Let $v = v_1, \ldots, v_n = v'$ (with $n > 1$) be the path in T connecting v and v', and note that by condition (2) in the definition of hinge-tree decompositions, $h \cap h' \subseteq \mathcal{N}(\lambda(v_i))$, for each $1 \leqslant i \leqslant n$. Thus, $h \cap h' \subseteq \mathcal{N}(\lambda(v_i)) \cap \mathcal{N}(\lambda(v_{i+1}))$ holds, for each $1 \leqslant i < n$. Therefore, $h \cap h' \subseteq h_{v_i,v_{i+1}}$ and $h \cap h' \subseteq h_{v_{i+1},v_i}$ hold as well, by definition of $h_{v_i,v_{i+1}}$ and h_{v_{i+1},v_i}, for each $1 \leqslant i < n$. By construction of $G(T)$, let us now observe that $\{h, h_{v_1,v_2}\}$ and $\{h', h_{v_n,v_{n-1}}\}$ are two edges of $G(T)$, and that either $h_{v_i,v_{i+1}} = h_{v_{i+1},v_i}$ or $\{h_{v_i,v_{i+1}}, h_{v_{i+1},v_i}\}$ is an edge of $G(T)$, for each $1 \leqslant i < n$. Thus, the path $h, h_{v_1,v_2}, \ldots, h_{v_i,v_{i+1}}, h_{v_{i+1},v_i}, \ldots, h_{v_n,v_{n-1}}, h'$ witnesses that condition (3) in Definition 3.1 is satisfied for $\{h, h'\}$.

Based on $\langle T, \lambda\rangle$, let us now build a tree decomposition $\langle \bar{T}, \bar{\chi}\rangle$ of $G(T)$ having width bounded by k—this tree decomposition will serve as the basis to computing the biconnected decomposition. For each vertex v of T, $\bar{T}$ contains a vertex q_v with $\bar{\chi}(q_v) = \lambda(v)$; for each edge $\{v, v'\}$ in T, $\bar{T}$ contains a vertex $q_{\{v,v'\}}$ with $\bar{\chi}(q_{\{v,v'\}}) = \{h_{v,v'}, h_{v',v}\}$, connected to the two vertices q_v and $q_{v'}$ associated with its endpoints (that is, for any edge $\{v, v'\}$ of T, $\bar{T}$ contains two edges $\{q_v, q_{\{v,v'\}}\}$ and $\{q_{\{v,v'\}}, q_{v'}\}$). No further node or edge is in $\bar{T}$. Note that, by construction, $\bar{T}$ is a tree, because it is obtained from the tree T by just breaking any edge into a pair of consecutive edges. Moreover, $\langle \bar{T}, \bar{\chi}\rangle$ is a tree decomposition of $G(T)$, since the following conditions are satisfied:

(1) *For each node h of $G(T)$, there is a vertex p of $\bar{T}$ such that $h \in \bar{\chi}(p)$.*
Indeed, by condition (1) in the definition of hinge-tree decompositions, for each hyperedge h of $\mathcal{H}$, there is a vertex v of T such that $h \in \lambda(v)$; hence, the result follows for $p = q_v$, by definition of the $\bar{\chi}$ labelling.

(2) *For each edge $\{h, h'\}$ of $G(T)$, there is a vertex p of $\bar{T}$ such that $\{h, h'\} \subseteq \bar{\chi}(p)$.*
Indeed, in the case where there is a vertex v such that $\{h, h'\} \subseteq \lambda(v)$, the result is immediate by definition of the $\bar{\chi}$ labelling. Otherwise, by construction of $G(T)$, there is a pair of adjacent vertices v and v' of T such that $\{h, h'\} = \{h_{v,v'}, h_{v',v}\}$. In this latter case, $\{h, h'\} \subseteq \bar{\chi}(q_{\{v,v'\}})$ holds.

(3) *For each node h in $G(T)$, the set $\{p \in \bar{T} \mid h \in \bar{\chi}(p)\}$ induces a connected subtree of $\bar{T}$.*
We start by considering any pair of vertices of the form q_v and $q_{v'}$ in $\bar{T}$ such that $h \in \bar{\chi}(q_v) \cap \bar{\chi}(q_{v'})$. By construction of $\bar{\chi}$, and because of Lemma 3.5(1), each vertex of the form q_w in the path in $\bar{T}$ between q_v and $q_{v'}$ is such that $h \in \bar{\chi}(q_w)$. Moreover, consider any vertex of the form $q_{\{w',w''\}}$ in this path. By construction, it is associated with an edge $\{w', w''\}$ in the corresponding path from v to v' in T. Therefore, $h = h_{w',w''} = h_{w'',w'}$ holds because h belongs to $\lambda(w') \cap \lambda(w'')$ by Lemma 3.5(1), which implies $h \subseteq h_{w',w''}$ and $h \subseteq h_{w'',w'}$, where the containment cannot be strict since $\mathcal{H}$ is reduced. We thus get $\bar{\chi}(q_{\{w',w''\}}) = \{h_{w',w''}, h_{w'',w'}\} = \{h\}$.
Consider now the case where q_v and $q_{\{v',v''\}}$ are two vertices of $\bar{T}$ such that $h \in \bar{\chi}(q_v) \cap \bar{\chi}(q_{\{v',v''\}})$ and $h \notin \bar{\chi}(q_{v''})$, where $q_{v''}$ and $q_{v'}$ are the (only) two vertices connected to $q_{\{v',v''\}}$. Because $h \notin \bar{\chi}(q_{v''})$, $h = h_{v',v''}$ and hence $h \in \bar{\chi}(q_{v'})$ hold by construction of $\bar{\chi}(q_{\{v',v''\}})$ and by definition of the hyperedge $h_{v',v''}$. From what we have seen above in this

proof for vertices of the form q_v and $q_{v'}$, each vertex in the path from q_v to $q_{v'}$ satisfies the connectedness condition for h over the $\bar{\chi}$ labelling. This condition thus holds for the path from q_v to $q_{\{v',v''\}}$, since $q_{\{v',v''\}}$ is directly connected to $q_{v'}$.

To conclude the proof, observe that the above two scenarios cover all the possible cases, since $h \in \bar{\chi}(q_{\{v,v'\}})$ implies, by construction of the $\bar{\chi}$ labelling, that $h \in \chi(q_v) \cup \chi(q_{v'})$ holds.

We next show that the tree decomposition $\langle \bar{T}, \bar{\chi} \rangle$ satisfies the following additional property:

(4) *For each pair of distinct vertices p, q of $\bar{T}$, $|\bar{\chi}(p) \cap \bar{\chi}(q)| \leqslant 1$.*
Consider a pair of distinct vertices p, q of $\bar{T}$, and assume by contradiction that $|\bar{\chi}(p) \cap \bar{\chi}(q)| > 1$ holds. Because of the above condition (3), this implies the existence of a pair of adjacent vertices q_w and $q_{\{w,w'\}}$ (in the path from p to q in $\bar{T}$) such that $|\bar{\chi}(q_{\{w,w'\}}) \cap \bar{\chi}(q_w)| > 1$. That is, $\{h_{w,w'}, h_{w',w}\} \subseteq \bar{\chi}(q_w)$ and $h_{w,w'} \neq h_{w',w}$ must hold. Moreover, $h_{w',w} \in \bar{\chi}(q_{w'})$ holds by definition. In particular, recall that $\langle \bar{T}, \bar{\chi} \rangle$ is built from $\langle T, \lambda \rangle$, where the above relationships entail $h_{w',w} \in \lambda(w) \cap \lambda(w')$. By definition of such hyperedges, $\mathcal{N}(\lambda(w')) \cap \mathcal{N}(\lambda(w)) \subseteq h_{w,w'}$ and $\mathcal{N}(\lambda(w')) \cap \mathcal{N}(\lambda(w)) \subseteq h_{w',w}$. Since $h_{w',w} \in \lambda(w) \cap \lambda(w')$, the converse of the latter inclusion holds too, and hence $\mathcal{N}(\lambda(w')) \cap \mathcal{N}(\lambda(w)) = h_{w',w}$. Thus, $\mathcal{N}(\lambda(w')) \cap \mathcal{N}(\lambda(w)) \subseteq h_{w,w'}$ implies $h_{w',w} \subseteq h_{w,w'}$, which in turn entails $h_{w,w'} = h_{w',w}$, because $\mathcal{H}$ is reduced. Contradiction.

As an example, Fig. 7 shows in (d) the tree decomposition of $G(T)$ associated with the hinge-tree decomposition depicted in (b). The reader may check that the additional condition (4) holds on it.

To conclude the proof, we now build a biconnected decomposition of $G(T)$, based on $\langle \bar{T}, \bar{\chi} \rangle$. To this end, observe preliminary that for some vertex p of $\bar{T}$, it is possible that $\bar{\chi}(p)$ is a singleton $\{h\}$. In this case, since $\mathcal{H}$ and thus $G(T)$ are connected, an edge of the form $\{h, h'\}$ exists in $G(T)$, which is moreover covered in the $\bar{\chi}$ labelling of some other vertex of $\bar{T}$ due to condition (2). Therefore, by condition (3), h must occur in some vertex p' connected to p, so that p may be safely deleted from $\bar{T}$ by contracting this edge $\{p, p'\}$ (that is, we get a new tree where p' is connected to the other neighbors of p in $\bar{T}$). Let $\langle \bar{T}', \bar{\chi}' \rangle$ be the new tree decomposition of $G(T)$ obtained by applying this simplification procedure until there are no more vertices with singleton labels. In particular, note that $\bar{\chi}'$ is the restriction of $\bar{\chi}$ to the vertices whose labelling is not a singleton. Hence, condition (4) still holds for the vertices in $\bar{T}'$, and the width of $\langle \bar{T}, \bar{\chi}' \rangle$ is still bounded by k.

Eventually, it only remains to show that a $\bar{\lambda}'$ labelling can be defined such that $\langle \bar{T}', \bar{\chi}', \bar{\lambda}' \rangle$ is an edge-defined decomposition. For every vertex p of $\bar{T}'$, let $\bar{\lambda}'(p)$ contain all those edges $\{h, h'\}$ of $G(T)$ such that $\{h, h'\} \subseteq \bar{\chi}'(p)$. By construction, $\mathcal{N}(\bar{\lambda}'(p)) \subseteq \bar{\chi}'(p)$. We claim that $\mathcal{N}(\bar{\lambda}'(p)) \supseteq \bar{\chi}'(p)$ holds, too. Indeed, this is immediate on a vertex $p = q_{\{w,w'\}}$, since $\bar{\chi}'(p) = \{h_{w,w'}, h_{w',w}\}$ and since $\{h_{w,w'}, h_{w',w}\}$ is in fact an edge of $G(T)$ (recall that the are no singleton in $\bar{T}'$). Consider then a vertex $p = q_v$ and note from Lemma 3.5(2) that the λ label of the hinge-tree decomposition $\langle T, \lambda \rangle$ cannot contain hyperedges disconnected from the rest of the label. By definition of $G(T)$, no connection is deleted between nodes associated with hyperedges in the same λ label. This property is thus inherited by $\bar{\chi}'$ where by construction, for every vertex q_v of $\bar{T}'$, $|\bar{\chi}'(q_v)| \geqslant 2$. It follows that, for any $h \in \bar{\chi}'(q_v)$, there is some neighbor h' of h in $G(T)$ such that $h' \in \bar{\chi}'(q_v)$, and hence these nodes will be covered by the edge $\{h, h'\} \subseteq \bar{\lambda}'(q_v)$. □

By combining the lemma above with the fact that $\texttt{BICOMP}^{optd} \preceq \texttt{HINGE}$ [15], we conclude that these two methods single out the same classes of tractable CSPs.

Theorem 3.7. $\texttt{BICOMP}^{optd} \equiv \texttt{HINGE}$.

It is worthwhile noting that, given any hinge-tree decomposition of a hypergraph $\mathcal{H}$, the proof of Lemma 3.6 provides an algorithm for computing an optimal reduct of *dual*($\mathcal{H}$) with respect to the BICOMP method. Since biconnected components can be computed in linear time, it follows that a $\texttt{BICOMP}^{optd}$ decomposition of a hypergraph $\mathcal{H}$ can be computed in time $O(|\mathcal{N}(\mathcal{H})||\mathcal{E}(\mathcal{H})|^2)$, which is the best-known upper bound for computing a hinge-tree decomposition [15].

Eventually, we leave this section by noticing that Theorem 3.7, Corollary 3.4, and the results in [10] (stating that HINGE is strongly incomparable with $\texttt{TREEDECOMP}^{p}$ and $\texttt{CUTSET}^{p}$) immediately lead to establish the following relationships.

Corollary 3.8. $\texttt{HINGE} \prec\!\!\prec \texttt{TREEDECOMP}^{optd}$.

Corollary 3.9. $\texttt{BICOMP}^{optd}$ *and* $\texttt{TREEDECOMP}^{p}$ *are strongly incomparable.*

Corollary 3.10. $\texttt{BICOMP}^{optd}$ *and* $\texttt{CUTSET}^{p}$ *are strongly incomparable.*

3.3. A closer look at tree decompositions

We continue the analysis of the dual-graph encoding, by looking in more detail at the $\texttt{TREEDECOMP}^{optd}$ method. In fact, while finding the most appropriate reduction w.r.t. the biconnected components methods emerged to be an easy task,

finding the most appropriate way of simplifying the dual-graph w.r.t. the tree decomposition method is a well-known challenging problem (cf. [18]), which makes the analysis of the decomposition power of TREEDECOMP^{optd} rather complex.

Next, we show how this problem can be circumvented by introducing a novel decomposition method that has essentially the same decomposition power as TREEDECOMP^{optd}, while getting rid of the need for finding an appropriate simplification of the dual graph. In particular, this new notion will be a crucial technical tool in this paper, as it is exploited in the most intricate proofs involving TREEDECOMP^{optd}.

Note that the method below is defined directly on the constraint hypergraph, and it can be seen as a modification of QUERYDECOMP (see Section 2).

Definition 3.11 *(*weakQUERYDECOMP*).* A *weak query decomposition* of a hypergraph $\mathcal{H}$ is a pair $\langle T, \lambda\rangle$, where $T = (N, E)$ is a tree, and λ is a labelling function which associates to each vertex $p \in N$ a set $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$, such that the following conditions hold:

(1) for each edge h of $\mathcal{H}$, there exists $p \in N$ such that $h \in \lambda(p)$;
(2) for each edge h of $\mathcal{H}$, the set $\{p \in N \mid h \in \lambda(p)\}$ induces a (connected) subtree of T; and
(3) for each pair of vertices v, v' of T, for each pair of edges $h \in \lambda(v)$, $h' \in \lambda(v')$, and for each vertex v'' in the path connecting v and v' in T, there is $h'' \in \lambda(v'')$ such that $h \cap h' \subseteq h''$.

The *width* of the weak query decomposition $\langle T, \lambda\rangle$ is $\max_{p\in N} |\lambda(p)|$. The *weak query-width* of $\mathcal{H}$ is the minimum width over all its weak query decompositions.

Note, in particular, that condition (3) in the definition above entails condition (3) in the definition of QUERYDECOMP. The impact of this modification on the decomposition power of QUERYDECOMP will be illustrated in Section 6. Here, we just show that weakQUERYDECOMP has the same decomposition power of TREEDECOMP^{optd}.

First, we observe that weakQUERYDECOMP generalizes TREEDECOMP^{optd}.

Theorem 3.12. TREEDECOMP$^{optd} \preceq$ weakQUERYDECOMP.

Proof. Let $\mathcal{H}$ be a hypergraph and $TD = \langle T, \chi\rangle$ a tree decomposition of some reduct G of *dual*$(\mathcal{H})$. We claim that TD is also a weak query decomposition of $\mathcal{H}$. To this end, consider the following conditions in Definition 3.11.

(1) *For each edge h of $\mathcal{E}(\mathcal{H})$, there exists a vertex v_h of T such that $h \in \chi(v_h)$.*
Indeed, for each hyperedge h of $\mathcal{E}(\mathcal{H})$, there is a node v_h in T such that $h \in \chi(v_h)$, by definition of tree decomposition and since h is a node of the dual graph.
(2) *For each edge h of $\mathcal{E}(\mathcal{H})$, the set of vertices of T whose χ labelling contains h is a subtree of T.*
Indeed, the connectedness condition trivially holds on TD, since TD is a tree decomposition.
(3) *For each pair of vertices v, v' of T, for each pair of edges $h \in \chi(v)$, $h' \in \chi(v')$, and for each vertex v'' in the path connecting v and v' in T, there is $h'' \in \chi(v'')$ such that $h \cap h' \subseteq h''$.*
Let v and v' be two vertices of T, and let $h \in \chi(v)$ and $h' \in \chi(v')$ with $h \cap h' \neq \emptyset$. In the case where $\{h, h'\}$ is an edge of G, then there is a vertex $v_{h,h'}$ such that $\chi(v_{h,h'}) \supseteq \{h, h'\}$. Moreover, by the connectedness condition, each vertex v'' in the path connecting v (resp., v') and $v_{h,h'}$ is such that $h \in \chi(v'')$ (resp., $h' \in \chi(v'')$), and the result immediately follows.
Assume then that $\{h, h'\}$ is not an edge of G. Then, there must be a path of the form $h = h_1, \ldots, h_n = h'$ satisfying condition (3) in Definition 3.1. In particular, observe that for each $1 \leqslant i \leqslant n$, $h \cap h' \subseteq h_i$ holds.
Since TD is a tree decomposition of G, for each edge $\{h_i, h_{i+1}\}$, there is a vertex $v_{h_i,h_{i+1}}$ in T such that $\chi(v_{h_i,h_{i+1}}) \supseteq \{h_i, h_{i+1}\}$. The result then follows since
- for each vertex v'' in the path between $v_{h_i,h_{i+1}}$ and $v_{h_{i+1},h_{i+2}}$, h_{i+1} is in $\chi(v'')$;
- for each vertex v'' in the path between v and v_{h_1,h_2}, $h_1 = h$ is in $\chi(v'')$; and
- for each vertex v'' in the path between v_{h_{n-1},h_n} and v', $h_n = h'$ is in $\chi(v'')$. □

On the other hand, weakQUERYDECOMP is not too much more powerful than TREEDECOMP^{optd}.

Theorem 3.13. *For each $k > 0$, C*(weakQUERYDECOMP, k) $\subseteq C$(TREEDECOMP^{optd}, $2 \times k - 1$).

Proof. Let $\langle T, \lambda\rangle$ be a weak query decomposition of a hypergraph $\mathcal{H}$ such that its width is bounded by k. Let us arbitrarily root T at some vertex. Then, we build a labelled tree $\langle T, \chi\rangle$ such that for each vertex r of T and for each child s of r in T, $\chi(s) = \lambda(s) \cup \lambda(r)$, i.e., s contains the same labelling as in λ plus the labelling of its father in T. Note that the cardinality of the χ labels over the vertices of T is bounded by $2 \times k$, and thus the width of $\langle T, \chi\rangle$ is bounded by $2 \times k - 1$ (recall the -1 in the standard definition of treewidth).

Then, define a graph $G(\langle T, \chi\rangle)$, whose set of nodes is the same as the dual graph $dual(\mathcal{H})$, and such that $\{h, h'\}$ is an edge of $G(\langle T, \chi\rangle)$ if and only if there is a vertex v of T such that $\{h, h'\} \subseteq \chi(v)$. We claim that $G(\langle T, \chi\rangle)$ is a reduct of $dual(\mathcal{H})$.

To prove the claim, consider an edge $\{h, h'\}$ of the dual graph that does not occur in $G(\langle T, \chi\rangle)$. Let v and v' be two vertices of T such that $h \in \chi(v)$ and $h' \in \chi(v')$, and let $v = v_1, \ldots, v_n = v'$ be the path in T between v and v'. Since $h \neq h'$, by condition (3) in the definition of weak query decompositions, for each $1 \leqslant i < n$, there is a hyperedge $h_i \in \lambda(v_i)$ such that $h \cap h' \subseteq h_i$. Hence, by construction of $\langle T, \chi\rangle$, for each $1 \leqslant i < n$, it is the case that $\{h_i, h_{i+1}\} \subseteq \chi(v'')$ for some vertex in the path $v_1, \ldots, v_n$. Thus, $\{h_i, h_{i+1}\}$ is an edge of $G(\langle T, \chi\rangle)$, for each $1 \leqslant i < n$. It follows that $h_1, \ldots, h_n$ witness that the edge $\{h, h'\}$ can safely be removed according to condition (3) in Definition 3.1.

To conclude the proof, we show that $\langle T, \lambda\rangle$ is in fact a tree decomposition of $G(\langle T, \chi\rangle)$. Note that each node of $G(\langle T, \chi\rangle)$ is covered by some vertex of $\langle T, \lambda\rangle$, because each node corresponds to a hyperedge, and hence is covered in $\langle T, \lambda\rangle$, by definition of weak query decomposition. Moreover, each edge of $G(\langle T, \chi\rangle)$ is trivially covered by some vertex of $\langle T, \lambda\rangle$, by construction of the graph $G(\langle T, \chi\rangle)$. Finally, as for the connectedness condition over λ, note that this condition holds on the λ labelling because of condition (2) in Definition 3.11. □

To summarize,

Corollary 3.14. *For each $k > 1$,*

$$C\big(\texttt{TREEDECOMP}^{optd}, k-1\big) \subseteq C(\texttt{weakQUERYDECOMP}, k) \subseteq C\big(\texttt{TREEDECOMP}^{optd}, 2 \times k - 1\big).$$

Remark 3.15. Note that, while QUERYDECOMP is intractable, it is not difficult to show, by exploiting the techniques discussed in [11], that for any fixed $k > 0$ we can check in polynomial time whether the weak query-width of a given hypergraph is bounded by k. Thus, weakQUERYDECOMP can be seen as a polynomial method for computing a 2-approximation of $\texttt{TREEDECOMP}^{optd}$. This is interesting, in the light that computing a reduct of a dual graph such that its treewidth is bounded by k is not known to be feasible in polynomial-time. In fact, it is an open problem raised by Kolaitis and Vardi [18].

4. Incidence-graph representation

In this section, we focus on the incidence-graph representation, by studying, in particular, the decomposition power of $\texttt{TREEDECOMP}^{in}$.

First, we compare this notion with HINGE, and we prove that there is no definitively best method between them. Then, we carry out the comparison with $\texttt{TREEDECOMP}^{optd}$, by showing that $\texttt{TREEDECOMP}^{optd}$ is slightly more powerful than $\texttt{TREEDECOMP}^{in}$.

4.1. Tree decomposition versus hinges

We start by showing that HINGE ($\texttt{BICOMP}^{optd}$) beats $\texttt{TREEDECOMP}^{in}$.

Theorem 4.1. $\texttt{HINGE}(\texttt{BICOMP}^{optd}) \rhd \texttt{TREEDECOMP}^{in}$.

Proof. We have to show that there is a class of hypergraphs that is tractable w.r.t. HINGE ($\texttt{BICOMP}^{optd}$), but not w.r.t. $\texttt{TREEDECOMP}^{in}$.

For each $m > 0$, let $Rose(m)$ be the hypergraph having m edges $\{S_1, \ldots, S_m\}$, and defined over the nodes $\{X_1, \ldots, X_m, Y_1, \ldots, Y_m\}$ such that $S_i = \{X_1, \ldots, X_m, Y_i\}$, for each $0 < i \leqslant m$. As an example, Fig. 8 shows the hypergraph $Rose(4)$ in (a), its incidence graph in (b), and a reduct of its dual graph in (c).

Observe, now, that:

- The graph $inc(Rose(m))$ is a bipartite graph, such that each S_i is connected to Y_i and to all the nodes in $\{X_1, \ldots, X_m\}$. Therefore, the treewidth of this graph is m.
- The graph $dual(Rose(m))$ is a clique of size m, such that each vertex S_i is connected to all the other nodes of the form S_j, with $j \neq i$. Moreover, $S_i \cap S_j = \{X_1, \ldots, X_m\}$, for each $j \neq i$. Therefore, we can built a reduct of $dual(Rose(m))$ as a tree rooted in S_1, such that each node of the form S_j with $j \neq 1$ is connected with an edge to h_1. Fig. 8 reports in (c) this reduct, by representing the edges removed from the dual graph with dotted lines.

Thus, for each $m > 0$, there is an instance $Rose(m+1)$ such that $Rose(m+1) \in C(\texttt{BICOMP}^{optd}, 1)$ (i.e., $Rose(m+1)$ is tractable according to $\texttt{BICOMP}^{optd}$), whereas $Rose(m+1) \notin C(\texttt{TREEDECOMP}^{in}, m)$ (i.e., $Rose(m+1)$ is not tractable according to $\texttt{TREEDECOMP}^{in}$). □

The picture is next completed, by showing that $\texttt{TREEDECOMP}^{in}$ may be better than HINGE.

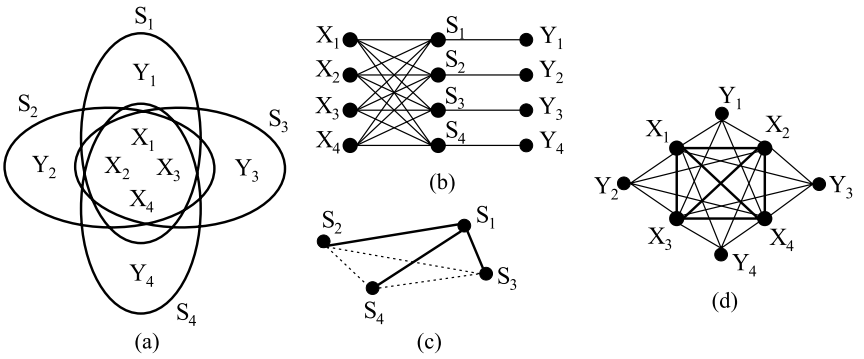


Fig. 8. (a) The hypergraph *Rose*(4), (b) its incidence graph, (c) a reduct of *dual*(*Rose*(4)), and (d) its primal graph.

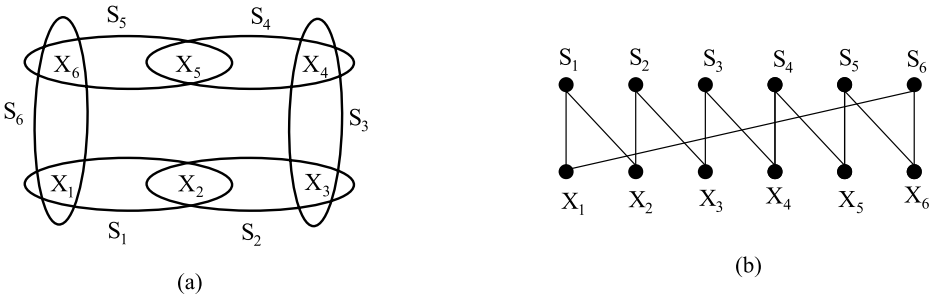


Fig. 9. (a) The hypergraph *Circle*(6), and (b) its incidence graph.

Theorem 4.2. $\texttt{TREEDECOMP}^{in} \rhd \texttt{HINGE}(\texttt{BICOMP}^{optd})$.

Proof. We show that there is a class of hypergraphs that is tractable w.r.t. $\texttt{TREEDECOMP}^{in}$, but not w.r.t. HINGE. For any $m > 2$, let *Circle*(m) be the hypergraph having m hyperedges $\{S_1, \ldots, S_m\}$ over the nodes $\{X_1, \ldots, X_m\}$ such that: $S_i = \{X_1, X_{i+1}\}$, for each $0 < i < m$; and $S_m = \{X_m, X_1\}$. As an example, Fig. 9 shows the hypergraph *Circle*(6) in (a) and its incidence graph in (b).

Note that the graph *inc*(*Circle*(m)) is a chain, whose treewidth is 2. Thus, the class $\{\textit{Circle}(m) \mid m > 0\}$ is tractable according to $\texttt{TREEDECOMP}^{in}$. Instead, in [15] it has been observed that this class is not tractable according to HINGE, because the width of *Circle*(m) w.r.t. HINGE is m. □

Corollary 4.3. $\texttt{HINGE}(\texttt{BICOMP}^{optd})$ *and* $\texttt{TREEDECOMP}^{in}$ *are strongly incomparable.*

4.2. Comparison with dual-graph representation

It is well known that both the dual-graph and the incidence-graph representations may be used for identifying tractable classes of non-binary CSPs according to the tree decomposition method (see, e.g., [18]). However, it was not clear whether either of these methods generalizes the other one or beats the other one on some classes of CSPs.

We start our analysis by showing that $\texttt{TREEDECOMP}^{optd}$ beats $\texttt{TREEDECOMP}^{in}$.

Theorem 4.4. $\texttt{TREEDECOMP}^{optd} \rhd \texttt{TREEDECOMP}^{in}$.

Proof. The result immediately follows from Theorem 4.1 ($\texttt{BICOMP}^{optd} \rhd \texttt{TREEDECOMP}^{in}$), Corollary 3.4 ($\texttt{BICOMP}^{optd} \lessdot \texttt{TREEDECOMP}^{optd}$, and hence $\texttt{BICOMP}^{optd} \preceq \texttt{TREEDECOMP}^{optd}$), and Proposition 2.1(1). □

Even though $\texttt{TREEDECOMP}^{optd}$ beats $\texttt{TREEDECOMP}^{in}$, we next show that the former method is not much more powerful than the latter, formally, that $\texttt{TREEDECOMP}^{optd}$ does not generalize $\texttt{TREEDECOMP}^{in}$. In fact, there are cases when moving to the dual-graph encoding causes an exponential blow-up of the width.

Theorem 4.5. $\texttt{TREEDECOMP}^{in}$ *exp-beats* $\texttt{TREEDECOMP}^{optd}$.

Proof. Consider the countably infinite class $\mathcal{C}$ of hypergraphs where the n-th element, $\forall n \geqslant 2$, is the hypergraph *ManySubsets*(n) such that:

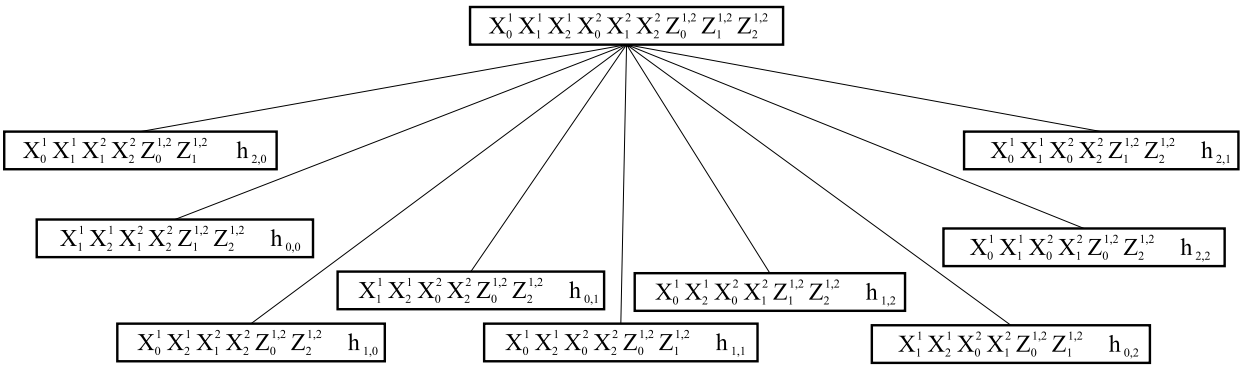


Fig. 10. A tree decomposition of $inc(ManySubsets(2))$ having width 9.

- $\mathcal{N}(ManySubsets(n))$ is the following set of nodes:

$$\bigcup_{1\leqslant j\leqslant n} \bar{X}^j \cup \bigcup_{1\leqslant p<q\leqslant n} \bar{Z}^{p,q}, \quad \text{where } \bar{X}^j = \{X_0^j, X_1^j, X_2^j\} \text{ and } \bar{Z}^{p,q} = \{Z_0^{p,q}, Z_1^{p,q}, Z_2^{p,q}\}.$$

  Note that $|\mathcal{N}(ManySubsets(n))| = 3 \times n + 3 \times \frac{n\times(n-1)}{2}$.

- Hyperedges in $ManySubsets(n)$ are built as follows. For each $0 \leqslant i < 3$, let $\bar{X}^j(i) = \bar{X}^j - \{X_i^j\}$ and $\bar{Z}^{p,q}(i) = \bar{Z}^{p,q} - \{Z_i^{p,q}\}$. Then, for each n-ple $0 \leqslant a_1, a_2, \ldots, a_n < 3$, $\mathcal{E}(ManySubsets(n))$ contains the hyperedge:

$$h_{a_1,\ldots,a_n} = \bigcup_{1\leqslant j\leqslant n} \bar{X}^j(a_j) \cup \bigcup_{1\leqslant p<q\leqslant n} \bar{Z}^{p,q}\big(c(a_p, a_q)\big),$$

  where $c(a_p, a_q) \triangleq (a_p + a_q) \mod 3$. In fact, there are exactly 3^n hyperedges in $\mathcal{E}(ManySubsets(n))$.

As an example, Fig. 10 shows a tree decomposition of the incidence graph of $ManySubsets(2)$, where the labelling of each leaf vertex exactly contains a node of the form $h_{a,b}$ with $0 \leqslant a, b < 3$, and all the nodes of $ManySubsets(2)$ contained in the hyperedge $h_{a,b} \in \mathcal{E}(ManySubsets(2))$.

Let us first focus on the incidence graph. The number of nodes in $ManySubsets(n)$ provides an upper bound on the treewidth of its incidence graph. Therefore, the width of $ManySubsets(n)$ w.r.t. $\texttt{TREEDECOMP}^{in}$ is bounded by $3 \times n + 3 \times \frac{n\times(n-1)}{2} \leqslant (3 \times n)^2$.

Consider now the dual graph, and notice that it consists of a clique over 3^n nodes. Indeed, for each pair of nodes $h_{a_1,\ldots,a_n}$ and $h_{a'_1,\ldots,a'_n}$ in $dual(ManySubsets(n))$, it holds that $h_{a_1,\ldots,a_n} \cap h_{a'_1,\ldots,a'_n} \neq \emptyset$, since (for instance) $\bar{X}^1(a_1) \cap \bar{X}^1(a'_1) \neq \emptyset$. In fact, we claim that no edge can be simplified from the dual graph, thereby obtaining that the width of $ManySubsets(n)$ w.r.t. $\texttt{TREEDECOMP}^{optd}$ is $3^n - 1$.

Assume, for the sake of contradiction, that an edge $\{h_{a_1,\ldots,a_n}, h_{a'_1,\ldots,a'_n}\}$ can be simplified. By condition (3) in Definition 3.1, there must exist a hyperedge $h_{a''_1,\ldots,a''_n}$ in $ManySubsets(n)$, with $h_{a''_1,\ldots,a''_n} \neq h_{a_1,\ldots,a_n}$ and $h_{a''_1,\ldots,a''_n} \neq h_{a'_1,\ldots,a'_n}$, such that $h_{a_1,\ldots,a_n} \cap h_{a'_1,\ldots,a'_n} \subseteq h_{a''_1,\ldots,a''_n}$ holds. By construction of the hyperedges in $ManySubsets(n)$, this inclusion entails that:

- for each $1 \leqslant j \leqslant n$, $\bar{X}^j - \{X_{a_j}^j, X_{a'_j}^j\} \subseteq \bar{X}^j - \{X_{a''_j}^j\}$; and
- for each $1 \leqslant p < q \leqslant n$, $\bar{Z}^{p,q} - \{Z_{c(a_p,a_q)}^{p,q}, Z_{c(a'_p,a'_q)}^{p,q}\} \subseteq \bar{Z}^{p,q} - \{Z_{c(a''_p,a''_q)}^{p,q}\}$.

Thus, the following two conditions hold: (i) for each $1 \leqslant j \leqslant n$, $X_{a''_j}^j \in \{X_{a_j}^j, X_{a'_j}^j\}$; and (ii) for each $1 \leqslant p < q \leqslant n$, $Z_{c(a''_p,a''_q)}^{p,q} \in \{Z_{c(a_p,a_q)}^{p,q}, Z_{c(a'_p,a'_q)}^{p,q}\}$.

Since $h_{a_1,\ldots,a_n}$ is distinct from $h_{a''_1,\ldots,a''_n}$, we have that there is an index $1 \leqslant p \leqslant n$ such that $a_p \neq a''_p$. By condition (i) above, $a_p \neq a''_p$ entails $X_{a''_p}^p = X_{a'_p}^p$ and, hence, $a''_p = a'_p$. Similarly, since $h_{a'_1,\ldots,a'_n}$ is distinct from $h_{a''_1,\ldots,a''_n}$, we have that there is an index $1 \leqslant q \leqslant n$ such that $a'_q \neq a''_q$. By condition (i) above, this entails $X_{a''_q}^q = X_{a_q}^q$ and, hence, $a''_q = a_q$. In addition, $p \neq q$ holds, for otherwise we would have $a''_q = a_q = a'_q$.

Assume without loss of generality that $p < q$ (otherwise, just swap the roles of the two indices). Then, $Z_{c(a''_p,a''_q)}^{p,q}$ coincides with $Z_{c(a'_p,a_q)}^{p,q}$, and hence condition (ii) can be rephrased as $Z_{c(a'_p,a_q)}^{p,q} \in \{Z_{c(a_p,a_q)}^{p,q}, Z_{c(a'_p,a'_q)}^{p,q}\}$, which entails that either $a'_p = a_p$ or $a_q = a'_q$. In the former case, $a'_p = a_p$ combined with $a''_p = a'_p$ is impossible because $a_p \neq a''_p$, by choice of the index p. In the latter case, $a_q = a'_q$ combined with $a''_q = a_q$ contradicts the fact that $a'_q \neq a''_q$ holds, by choice of the index q.

Input: A tree decomposition $\langle T, \chi \rangle$ of $inc(\mathcal{H})$;
Output: A weak query decomposition $\langle T, \lambda \rangle$ of $\mathcal{H}$;
var: $cover : 2^{\mathcal{N}(\mathcal{H})} \mapsto 2^{\mathcal{E}(\mathcal{H})}$;

begin
  for each $V' \subseteq \mathcal{N}(\mathcal{H})$ s.t. there is $h \in \mathcal{E}(\mathcal{H})$ with $V' \subseteq h$ **do**
    choose any $h \in \mathcal{E}(\mathcal{H})$ s.t. $V' \subseteq h$;
    $cover(V') := \{h\}$;
  end for
  for each vertex p of T **do**
    $\lambda(p) := \chi(p) \cap \mathcal{E}(\mathcal{H})$;
    for each $V' \subseteq \chi(p) \cap \mathcal{N}(\mathcal{H})$ s.t. there is $h \in \mathcal{E}(\mathcal{H})$ with $V' \subseteq h$ **do**
      $\lambda(p) := \lambda(p) \cup cover(V')$;
  end for
  return $\langle T, \lambda \rangle$;
end.

Fig. 11. Algorithm incidenceToWeakQueryDecomposition.

In summary, for the class of hypergraphs $\mathcal{C}$, $\forall n > 1$, $w^{\mathcal{C}}_{in}(n) \leqslant 9n^2$ and $w^{\mathcal{C}}_{optd}(n) = 3^n - 1$, where the former denote the width function w.r.t. TREEDECOMP^{in} and the latter the width function w.r.t. TREEDECOMP^{optd}. Thus, $w^{\mathcal{C}}_{optd}(n)$ is $\Omega(3^{\frac{\sqrt{w^{\mathcal{C}}_{in}(n)}}{3}})$, and thus it is also $\Omega(2^{(w^{\mathcal{C}}_{in}(n))^{\delta}})$, where δ is any rational number such that $0 < \delta < \frac{1}{2}$, which means that TREEDECOMP^{in} *exp*-beats TREEDECOMP^{optd}. □

To complete the picture of the relationship between TREEDECOMP^{optd} and TREEDECOMP^{in}, we next show that TREEDECOMP^{optd} *exp*-generalizes TREEDECOMP^{in}.

Technically, we exploit here the relationship between these methods and the weakQUERYDECOMP method. Indeed, we first show that any tree decomposition of the incidence graph with width k can be transformed into a weak query decomposition having width at most 2^{k+1}.

The algorithm performing such a construction is shown in Fig. 11. It takes as its input a tree decomposition $TD = \langle T, \chi \rangle$ of $inc(\mathcal{H})$, and outputs a weak query decomposition $TD' = \langle T, \lambda \rangle$ of $\mathcal{H}$. Basically, for each node p of T, it computes the labelling $\lambda(p)$ of this node in the weak query decomposition TD' from its label $\chi(p)$ in the given tree decomposition TD as follows: Let $\chi(p) = V_p \cup E_p$, where $V_p \subseteq \mathcal{N}(\mathcal{H})$ and $E_p \subseteq \mathcal{E}(\mathcal{H})$. Then, $\lambda(p)$ contains all the hyperedges in E_p plus one arbitrarily chosen hyperedge $h_{V'}$ that covers V' (i.e., such that $V' \subseteq h_{V'}$), for each subset V' of V_p.

Fig. 12 shows an example of the application of this algorithm.

Lemma 4.6. *Let $\mathcal{H}$ be a hypergraph and $\langle T, \chi \rangle$ a tree decomposition of inc($\mathcal{H}$). Let p be a vertex of T, $h \in \mathcal{E}(\mathcal{H})$, $\bar{X} \subseteq \chi(p)$, $\bar{X} \subseteq h$, and $h \notin \chi(p)$. Moreover, let q be the first vertex in the path Π of T connecting p to any vertex q' with $h \in \chi(q')$. Then, either $h \in \chi(q)$, or $\bar{X} \subseteq \chi(q)$.*

Proof. By contradiction, assume the statement is false and let q be a vertex in the path Π such that $h \notin \chi(q)$ and $X \notin \chi(q)$, for some node $X \in \bar{X}$. Since T is a tree, dropping the vertex q gives two connected components, say $C_{q'}$ and C_p, where the former contains the vertex q' and the latter the vertex p. Now, consider the subgraph $T_X = (V_X, E_X)$ of T induced by those vertices v such that $X \in \chi(v)$. From the connectedness condition of tree decompositions for node X, its vertices should belong to the component C_p, i.e., $V_X \subseteq C_p$. Indeed, observe that $p \in C_p$ and $X \in \chi(p)$, and $q \notin V_X$, because $X \notin \chi(q)$. Similarly, from the connectedness condition for node h, the subgraph $T_h = (V_h, E_h)$ of T induced by those vertices v such that $h \in \chi(v)$ is such that $V_h \subseteq C_{q'}$ holds. Indeed, $q' \in C_{q'}$ and $h \in \chi(q')$, and $q \notin V_X$, because $h \notin \chi(q)$. However, since $X \in h$, there is an edge $\{X, h\}$ in the incidence graph $inc(\mathcal{H})$, and thus there is some vertex p' in the tree decomposition $\langle T, \chi \rangle$ such that $\{X, h\} \subseteq \chi(p')$. Therefore, p' should belong to both V_X and V_h, which is impossible, because $C_p \cap C_{q'} = \emptyset$. □

Lemma 4.7. *Let $\mathcal{H}$ be a hypergraph and $\langle T, \chi \rangle$ a tree decomposition of inc($\mathcal{H}$). Let p and q be two distinct vertices of T and Π the path connecting them in T. Moreover, let $h_1, h_2 \in \mathcal{E}(\mathcal{H})$ be two hyperedges of $\mathcal{H}$ with a non-empty intersection $\bar{X} = h_1 \cap h_2$, and such that $h_1 \in \chi(p)$ and $h_2 \in \chi(q)$. Then, for every vertex p' in the path Π such that $h_1 \notin \chi(p')$ and $h_2 \notin \chi(p')$, $\bar{X} \subseteq \chi(p')$ holds.*

Proof. By contradiction, assume the statement is not true and let p' be a vertex in the path Π such that $h_1 \notin \chi(p')$, $h_2 \notin \chi(p')$, and $X \notin \chi(p')$, for some node $X \in \bar{X}$.

Since $X \in h_1$ and $X \in h_2$, there are two edges $\{X, h_1\}$ and $\{X, h_2\}$ in the incidence graph $inc(\mathcal{H})$, and thus two vertices v_1 and v_2 (not necessarily distinct) in the tree decomposition $\langle T, \chi \rangle$ such that $\{X, h_1\} \subseteq \chi(v_1)$ and $\{X, h_2\} \subseteq \chi(v_2)$. From

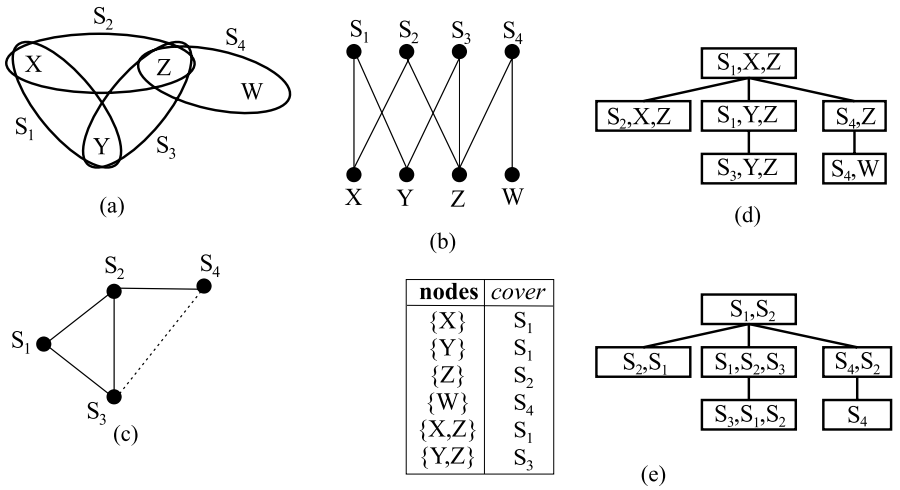


Fig. 12. (a) A hypergraph $\mathcal{H}$, (b) its incidence graph, (c) a reduct of the dual graph, (d) a tree decomposition of $inc(\mathcal{H})$, and (e) a weak query decomposition of $\mathcal{H}$ built using the algorithm in Fig. 11.

the connectedness condition of tree decompositions for node X, there is a (possibly empty) path from v_1 to v_2 in T such that all its vertices have X in their χ label. Similarly, from the connectedness condition of tree decompositions for node h_1 (respectively, h_2), there is a (possibly empty) path from v_1 to p (respectively, from v_2 to q) such that all its vertices have h_1 (respectively, h_2) in their χ label. Thus, there is a path Π' from p to q (possibly consisting of a single edge connecting them) such that, for all vertices v in Π', $X \in \chi(v)$, or $h_1 \in \chi(v)$, or $h_2 \in \chi(v)$. Since none of these nodes belong to the label $\chi(p')$, the paths Π' and Π from p to q in T are distinct, which is impossible, because T is a tree. □

Lemma 4.8. *Let $\mathcal{H}$ be a hypergraph. Given a tree decomposition $\langle T, \chi\rangle$ of $inc(\mathcal{H})$ whose width is k, the algorithm in Fig.* 11 *outputs a weak query decomposition $\langle T, \lambda\rangle$ of $\mathcal{H}$, whose width is 2^{k+1} at most.*

Proof. Let $\mathcal{H}$ be a hypergraph, let $\langle T, \chi\rangle$ be a tree decomposition of the graph $inc(\mathcal{H})$, with $T = (V, E)$, and let $\langle T, \lambda\rangle$ be the output of the algorithm in Fig. 11. We show that $\langle T, \lambda\rangle$ fulfills all the three conditions in Definition 3.11 and thus it is a weak query decomposition of $\mathcal{H}$.

(1) *For each edge h of $\mathcal{H}$, there exists $p \in N$ such that $h \in \lambda(p)$.*
This is trivial, because every $h \in \mathcal{E}(\mathcal{H})$ is a vertex of $inc(\mathcal{H})$ connected to some (node-)vertex of $inc(\mathcal{H})$. Thus, h occurs in the χ label of some vertex p of the T and hence in $\lambda(p)$.

(2) *For each edge h of $\mathcal{H}$, the set $\{p \in N \mid h \in \lambda(p)\}$ induces a (connected) subtree of T.*
Consider such an (hyper)edge h and the set of nodes $C \subseteq V$ where h occurs in some λ label. Since T is a tree, it suffices to show that the subgraph of T induced by C is connected. Let $C' \subseteq C$ be the set of vertices where h already occurs in the χ labels, that is, $C' = \{p \in V \mid h \in \chi(p)\}$. Note that the subgraph of T induced by C' is connected, because of the connectedness condition of tree decompositions. Also, note that $C' \neq \emptyset$, as observed at point (1) above.
Assume by contradiction that the subgraph induced by the full set C is not connected, and let $C'' \subset V$ be another (non-empty maximal) connected component of this subgraph, with $C'' \neq C'$ (and clearly $C'' \cap C' = \emptyset$). Then, there exists a borderline vertex $p \in C''$ connected through a non-empty path Π to some vertex of C', where Π does not contain any vertex of C''. In particular, if q is the first vertex of Π, that is, the vertex adjacent to p in T, we have that $q \notin C''$. Note that $h \in \lambda(p)$ and $h \notin \chi(p)$ because $p \in C - C'$. Then, it should be the case that $h = cover(\bar{X})$, for some $\bar{X} \subseteq h$ and $\bar{X} \subseteq \chi(p)$, by the construction in Algorithm incidenceToWeakQueryDecomposition. However, this immediately leads to a contradiction to the existence of the vertex q in the path Π. Indeed, from Lemma 4.6 either $h \in \chi(q)$, and thus $h \in \lambda(q)$, or $\bar{X} \subseteq \chi(q)$ and again $h \in \lambda(q)$, because $h = cover(\bar{X})$. Both cases entail $q \in C''$ from the maximality of the component C'', which is impossible, by the choice of p and q.

(3) *For each pair of vertices v, v' of T, for each pair of edges $h \in \lambda(v)$, $h' \in \lambda(v')$, and for each vertex v'' in the path connecting v and v' in T, there is $h'' \in \lambda(v'')$ such that $h \cap h' \subseteq h''$.*
Consider such a pair of vertices $v, v' \in V$, with $h \in \lambda(v)$ and $h' \in \lambda(v')$, let Π be the path connecting them in T, and let $\bar{X} = h \cap h'$. Assume that $\bar{X} \neq \emptyset$ and that there exists v'' in Π such that both $h \notin \lambda(v'')$ and $h' \notin \lambda(v'')$, otherwise condition (3) trivially holds. Since $\langle T, \chi\rangle$ is a tree decomposition of $inc(\mathcal{H})$ and h and h' are two nodes occurring in some edges of this graph, of course there are two vertices $v_1, v_2 \in V$ such that $h \in \chi(v_1)$ and $h' \in \chi(v_2)$. Then, according to the above algorithm, $h \in \lambda(v_1)$ and $h' \in \lambda(v_2)$, too. Moreover, according to condition (2) above, the connectedness condition holds for the λ labelling in $\langle T, \lambda\rangle$. It follows that v'' belongs also to the path connecting v_1 and v_2 in T. From

Lemma 4.7, $\bar{X} \subseteq \chi(v'')$ and thus, according to Algorithm INCIDENCETOWEAKQUERYDECOMPOSITION, there is a hyperedge $h'' = cover(\bar{X})$ such that $\bar{X} \subseteq h''$ and $h'' \in \lambda(v'')$.

In order to complete the proof, observe that the width of the decomposition $\langle T, \lambda \rangle$ is bounded by 2^{k+1}, if k is the width of $\langle T, \chi \rangle$. Indeed, let v be a vertex of T, and let $V_p = \chi(p) \cap \mathcal{N}(\mathcal{H})$ and $E_p = \chi(p) \cap \mathcal{E}(\mathcal{H})$. Then, $\lambda(v)$ contains (at most) one hyperedge for each subset of the variables in V_p plus the hyperedges in E_p. That is, $|\lambda(v)| \leqslant 2^{|V_p|} + |E_p| \leqslant 2^{k+1}$, because $|\chi(p)| \leqslant k+1$. □

Note that, in some cases, the algorithm in Fig. 11 may directly output a tree decomposition. This is for instance the case of the example in Fig. 12, where the decomposition (e) produced by the algorithm is actually a tree decomposition of the reduct in (c). In any case, transforming a weak query decomposition into a tree decomposition can easily be done (without a large increase in the width). Indeed, from Lemma 4.8 and the relationship between $\texttt{TREEDECOMP}^{optd}$ and weakQUERYDECOMP, we easily get the following result.

Theorem 4.9. $\texttt{TREEDECOMP}^{optd}$ *exp-generalizes* $\texttt{TREEDECOMP}^{in}$.

Proof. Let $\mathcal{H}$ be a hypergraph and $\langle T, \chi \rangle$ a tree decomposition of $inc(\mathcal{H})$ whose width is k. From Lemma 4.8, there is a weak query decomposition of $\mathcal{H}$ having width $w = 2^{k+1}$, at most. Therefore, from Theorem 3.13, there is a tree decomposition of some reduct of the dual graph whose width is bounded by $2 \times w - 1 = 2^{k+2} - 1$. □

Putting Theorems 4.4, 4.9, and 4.5 together, the precise relationship between $\texttt{TREEDECOMP}^{in}$ and $\texttt{TREEDECOMP}^{optd}$ derives.

Corollary 4.10. $\texttt{TREEDECOMP}^{in} \precsim \texttt{TREEDECOMP}^{optd}$.

4.3. General relationships between incidence-graph and dual-graph representations

We leave this section by noticing that the properties stated in Theorems 4.4 and 4.5 between $\texttt{TREEDECOMP}^{in}$ and $\texttt{TREEDECOMP}^{optd}$ are in fact representative of the relationship between any pair of binary decomposition methods applied to the incidence and the dual graph, respectively.

Theorem 4.11. *Let* $\texttt{D}_1$ *and* $\texttt{D}_2$ *be two methods in* {TREEDECOMP, BICOMP, CUTSET}. *Then,*

(1) ${\texttt{D}_1}^{optd} \rhd {\texttt{D}_2}^{in}$; *and*
(2) ${\texttt{D}_2}^{in}$ *exp-beats* ${\texttt{D}_1}^{optd}$.

Proof.

(1) Consider the class of hypergraphs $\{Rose(m) \mid m > 0\}$, defined in Theorem 4.4, and recall that the treewidth of the incidence graph is m, whereas there is a reduct of the dual graph that is acyclic. Therefore, for any method $\texttt{D}_1$ that generalizes acyclicity and in particular for $\texttt{D}_1 \in$ {TREEDECOMP, BICOMP, CUTSET}, the width of $\texttt{D}_1$ applied to the dual graph is 1. However, the width w.r.t. any method $\texttt{D}_2$ applied to the incidence graph is at least m, since this is the case for TREEDECOMP, which generalizes all the other graph methods.
(2) Consider the class of hypergraphs $\{ManySubsets(n) \mid n > 0\}$, defined in Theorem 4.5. The width w.r.t any method $\texttt{D}_2$ on the incidence graph is bounded by $(3 \times n)^2$, which is indeed an upper bound on the total number of nodes. Moreover, recall from the proof of Theorem 4.5 that, for any $n > 0$, the dual graph of $ManySubsets(n)$ cannot be simplified. Thus every method ${\texttt{D}_1}^{optd}$ will deal with the same (dual) graph, and hence $3^n - 1$ will be a lower bound for its width for this graph, as for $\texttt{TREEDECOMP}^{optd}$, which generalizes all the other graph methods. Then, the *exp*-beating relationship is formally proved precisely with the same reasoning as in the proof of Theorem 4.5. □

5. Primal-graph representation

In this section, we study the primal-graph representation, by first focusing on the decomposition power of $\texttt{TREEDECOMP}^{p}$. In particular, we compare this notion with the tree decomposition method applied to the incidence-graph representation, and then with the tree decomposition method applied to the dual-graph representation. Further relationships with the other decomposition methods over the primal graph will also be discussed, to give a complete picture.

5.1. Comparison with the incidence-graph representation

We start our analysis by showing that $\texttt{TREEDECOMP}^{in}$ beats $\texttt{TREEDECOMP}^{p}$.

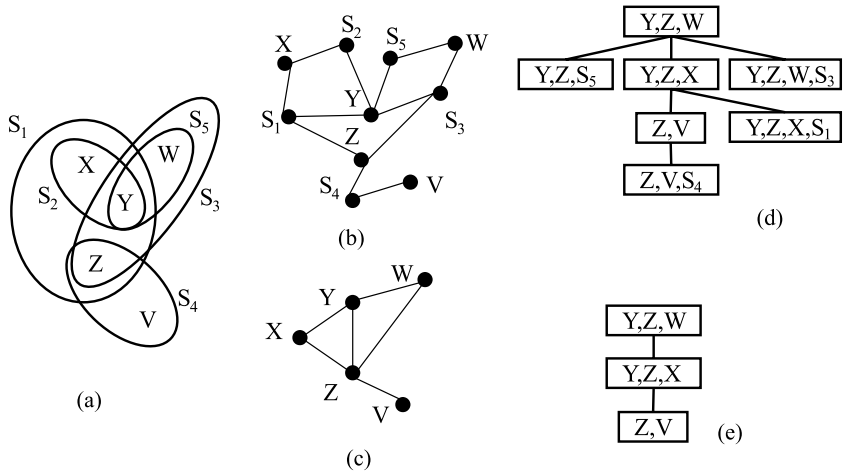


Fig. 13. (a) An hypergraph $\mathcal{H}$, (b) its incidence graph, (c) its primal graph, (d) a tree decomposition *TD* of the primal graph, and (e) the corresponding tree decomposition *TD*′ of the incidence graph, according to the construction in Theorem 5.2.

Theorem 5.1. $\texttt{TREEDECOMP}^{in} \rhd \texttt{TREEDECOMP}^{p}$.

Proof. We show that there is a class of hypergraphs that is tractable w.r.t. $\texttt{TREEDECOMP}^{in}$, but not w.r.t. $\texttt{TREEDECOMP}^{p}$. For any $n > 0$, let *Single*(n) be the hypergraph having one hyperedge only over n different nodes. Then, the primal graph of *Single*(n) is a clique of n nodes, whose treewidth is $n-1$. Conversely, the incidence graph of *Single*(n) is trivially acyclic, and hence it has treewidth 1. □

In fact, by adapting the arguments suggested in [18], we next show that $\texttt{TREEDECOMP}^{in}$ strongly generalizes $\texttt{TREEDECOMP}^{p}$.

Theorem 5.2. $\texttt{TREEDECOMP}^{p} \lessdot\lessdot \texttt{TREEDECOMP}^{in}$.

Proof. After Theorem 5.1, it remains to show that $\texttt{TREEDECOMP}^{p}$ generalizes $\texttt{TREEDECOMP}^{in}$. To this end, we claim that, for each $k > 0$, $C(\texttt{TREEDECOMP}^{p}, k) \subseteq C(\texttt{TREEDECOMP}^{in}, k+1)$.

Let $\mathcal{H}$ be a hypergraph and assume that $\mathcal{H} \in C(\texttt{TREEDECOMP}^{p}, k)$. Then, consider a tree decomposition $TD = \langle T, \chi \rangle$ of the primal graph of $\mathcal{H}$, whose width is bounded by k. Based on *TD*, we build a labelled tree $TD' = \langle T', \chi' \rangle$ as follows—see Fig. 13, for an illustration.

- T' contains all the vertices in T, and indeed the subgraph of T' induced over the nodes in T precisely coincides with T. In particular, for each vertex v in both T and T', we have that $\chi(v) = \chi'(v)$.
- For each hyperedge $h \in \mathcal{E}(\mathcal{H})$ an arbitrary vertex v_h of T is selected such that $\chi(v_h) \supseteq h$—note that one such vertex v_h exists, since nodes in h form a clique in the primal graph and hence, as is well known, there must be a vertex covering all of them in any tree decomposition. Then, a new vertex v'_h is added to T' as a child of v_h, with $\chi'(v'_h) = \chi(v_h) \cup \{h\}$.

We next show that *TD*′ is a tree decomposition of the incidence graph *inc*$(\mathcal{H})$.

First, we note that each node of *inc*$(\mathcal{H})$ is covered in the χ' labelling of some vertex of T'. Indeed, this is obvious for the nodes in $\mathcal{N}(\mathcal{H})$ that are already covered in the χ' labelling. Moreover, for each hyperedge h, the property holds as well, since by construction of T' the vertex v'_h is such that $h \in \chi'(v'_h)$.

Second, consider an edge $\{X, h\}$ of *inc*$(\mathcal{H})$. Note that $X \in h$, by definition of incidence graph. Then, $\{X, h\}$ is covered in the vertex v'_h, since $\chi(v_h)$ is actually a superset of h and hence contains X.

And, third, note that the connectedness condition of tree decomposition is satisfied in *TD*, and hence it is also satisfied in *TD*′ over the nodes in $\mathcal{H}$, because for each vertex v occurring in both T and T', $\chi(v) = \chi'(v)$, and because for each other vertex v_h added in T' as a child of r, it holds that $\chi'(v'_h) \supseteq \chi(r)$ and v'_h is the only vertex of T' where h is covered.

To conclude the proof, it is then sufficient to observe that the width of *TD*′ is bounded by $k+1$. □

5.2. Comparison with the dual-graph representation

Similarly to the case of the relationship between dual-graph and incidence-graph encodings, we next show that there is no definitively best binary representation, if we consider graph methods applied either to dual graphs or to primal graphs.

Theorem 5.3. *Let* $\mathtt{D}_1$ *and* $\mathtt{D}_2$ *be two methods in* {TREEDECOMP, BICOMP, CUTSET}. *Then,*

(1) ${\mathtt{D}_1}^{optd} \rhd {\mathtt{D}_2}^{p}$; *and*
(2) ${\mathtt{D}_2}^{p}$ *exp-beats* ${\mathtt{D}_1}^{optd}$.

Proof.

(1) Consider again the class of hypergraphs $\{Single(n) \mid n > 0\}$, defined in the proof of Theorem 5.1. The result then follows since the primal graph of $Single(n)$ is a clique of n nodes, whose treewidth is $n-1$, whereas the dual graph consists of a single node and, hence, it is trivially acyclic.
(2) Consider the class of hypergraphs $\{ManySubsets(n) \mid n > 0\}$, defined in Theorem 4.5. Even in this case, the width w.r.t. any method on the primal graph is bounded by $(3 \times n)^2$ which is indeed an upper bound on the total number of nodes. Then the statement follows from the same reasoning as in Theorem 4.11. □

We can now state the precise relationships between $\mathtt{TREEDECOMP}^{optd}$, $\mathtt{TREEDECOMP}^{p}$, $\mathtt{BICOMP}^{p}$, $\mathtt{CUTSET}^{p}$, and $\mathtt{BICOMP}^{optd}$.

Theorem 5.4. $\mathtt{TREEDECOMP}^{p} \precsim \mathtt{TREEDECOMP}^{optd}$.

Proof. In the light of Theorem 5.3, it only remains to show that $\mathtt{TREEDECOMP}^{optd}$ *exp*-generalizes $\mathtt{TREEDECOMP}^{p}$. To this end, observe that $\mathtt{TREEDECOMP}^{in} \precsim \mathtt{TREEDECOMP}^{optd}$ and $\mathtt{TREEDECOMP}^{p} \lessdot\!\lessdot \mathtt{TREEDECOMP}^{in}$ hold, by Theorem 4.10 and Theorem 5.2, respectively. In particular, $\mathtt{TREEDECOMP}^{p} \preceq \mathtt{TREEDECOMP}^{in}$. Hence, the result follows because of Proposition 2.4(3). □

Theorem 5.5. $\mathtt{CUTSET}^{p} \precsim \mathtt{TREEDECOMP}^{optd}$.

Proof. In the light of Theorem 5.3, it only remains to show that $\mathtt{TREEDECOMP}^{optd}$ *exp*-generalizes $\mathtt{CUTSET}^{p}$. To this end, recall that $\mathtt{CUTSET}^{p} \lessdot\!\lessdot \mathtt{TREEDECOMP}^{p}$ (and, hence, $\mathtt{CUTSET}^{p} \preceq \mathtt{TREEDECOMP}^{p}$) has been shown in [10], and that $\mathtt{TREEDECOMP}^{p} \precsim \mathtt{TREEDECOMP}^{optd}$ holds by Theorem 5.4. Thus, the result follows again by applying Proposition 2.4(3). □

We next focus on the decomposition method based on the notion of biconnected component. Recall that $\mathtt{BICOMP}^{optd}$ is equivalent to the hyperedge-based method HINGE, by Theorem 3.8. Moreover, if we consider binary CSPs, it is known from [10], that HINGE strongly generalizes $\mathtt{BICOMP}^{p}$. This result was based on the observation in [15] that every biconnected component C of a graph corresponds to a hinge consisting of those (hyper)edges containing the variables in C. What was not observed in [15] and [10] is that the cardinality of this hinge in the general non-binary case may be exponentially larger than the cardinality of C. Indeed, many hyperedges may contain these variables, in the worst case the power set of C (see, e.g., the hypergraph $ManySubsets(n)$, for such a pathological example). In fact, we next show that, in the general case, the precise relationship between these two approaches is the weak generalization.

Theorem 5.6. $\mathtt{BICOMP}^{p} \precsim \mathtt{HINGE}(\mathtt{BICOMP}^{optd})$.

Proof. In the light of Theorem 5.3, we have to show that $\mathtt{BICOMP}^{optd}$, or equivalently HINGE, by Theorem 3.8, *exp*-generalizes $\mathtt{BICOMP}^{p}$. Let $\mathcal{H}$ be a hypergraph and G its primal graph. Let $\langle T, \chi, \lambda \rangle$ be a biconnected decomposition of $\mathcal{H}$, and let k be the width of this decomposition, i.e., the maximum cardinality over the biconnected components of G [5]. Recall that, on binary CSPs, it has been shown that every biconnected component C of a graph is a hinge [15] or, more precisely, the set $H(C) = \{h \in \mathcal{E}(\mathcal{H}) \mid h \subseteq C\}$ is a hinge. By applying the same line of reasoning as in [15], it is easy to see that this property holds in the general case above, where the graph G is the primal graph of the hypergraph $\mathcal{H}$. Just observe that every hyperedge h such that $|h| > 2$ corresponds to a clique in G, and thus it is included in some biconnected component, because variables in h cannot be disconnected by dropping any single variable. It follows that even in this general case $\langle T, \chi, \lambda \rangle$ corresponds to a hinge decomposition of $\mathcal{H}$. Let C be a biconnected component of G. Then, the corresponding hinge $H(C)$ of $\mathcal{H}$ may contain at most $2^{|C|}$ hyperedges. Since the cardinality of the largest biconnected component is k, 2^k is clearly an upper bound for the width of the hinge-tree decomposition of $\mathcal{H}$ associated with $\langle T, \chi, \lambda \rangle$, and thus it is an upper bound for the HINGE width of $\mathcal{H}$, too. □

Eventually, we conclude with the relationship between $\mathtt{BICOMP}^{p}$ and $\mathtt{TREEDECOMP}^{optd}$.

Theorem 5.7. $\mathtt{BICOMP}^{p} \precsim \mathtt{TREEDECOMP}^{optd}$.

Proof. In the light of Theorem 5.3, it only remains to show that $\mathtt{TREEDECOMP}^{optd}$ *exp*-generalizes $\mathtt{BICOMP}^{p}$. To this end, recall that $\mathtt{BICOMP}^{optd} \lessdot\!\lessdot \mathtt{TREEDECOMP}^{optd}$ (and, hence, $\mathtt{BICOMP}^{optd} \preceq \mathtt{TREEDECOMP}^{optd}$) follows by Theorem 3.7 and

by Theorem 3.8. Moreover, $\texttt{BICOMP}^{p} \precsim \texttt{BICOMP}^{optd}$ holds by Theorem 5.6. Thus, the result follows by applying Proposition 2.4(3). □

6. Binary versus non-binary structural methods

In the previous sections, we have analyzed the decomposition power of several graph-based decomposition methods applied to binary representations of non-binary instances. In this section, we compare these methods with hypergraph-based methods.

In particular, we start by showing that no binary method is better than QUERYDECOMP. Then, we motivate this behavior by observing that the power of binary methods is basically bounded by the weakQUERYDECOMP method introduced in Section 3.3, which is indeed a relaxation of QUERYDECOMP.

6.1. Binary methods versus QUERYDECOMP

The idea that is common to all hypergraph-based decomposition methods is to exploit the power of hyperedges, which may contain many variables, to fulfill the connectedness condition while keeping the width small. This is, for instance, the case of the QUERYDECOMP method (see Section 2).

Indeed, observe that in the definition of QUERYDECOMP the width is measured in terms of "number of hyperedges" in the tree labels, rather than in terms of "number of variables," as it is usual in graph methods. In fact, it has been shown that CSP instances having such widths bounded by some constant may be solved in polynomial time, though with an exponential dependency on the width. Thus, these techniques exhibit the same kind of complexity bounds as typical graph-methods with variable-based widths, but usually with smaller—more efficient—widths for the same instances [10].

As a matter of fact, QUERYDECOMP is however hardly used in practice, because checking whether the query width of a hypergraph is bounded by k is an NP-hard problem, even for the fixed $k = 4$ [12]. Our interest w.r.t. this notion derives from the fact that all the (tractable) hypergraph-based decomposition methods proposed in the literature do generalize QUERYDECOMP. In particular, we next consider the hypertree decomposition method (HYPERTREE) [12], the spread-cuts method (SPREADCUT) [5], and the component hypertree decomposition method (CHYPERTREE) [13].

Theorem 6.1. *(See [12].)* $\texttt{QUERYDECOMP} \preceq \texttt{HYPERTREE}$.

Theorem 6.2. *(See [5].)* $\texttt{QUERYDECOMP} \preceq \texttt{SPREADCUT}$.

Theorem 6.3. $\texttt{QUERYDECOMP} \preceq \texttt{CHYPERTREE}$.

Proof. In [13], it has been shown that $\texttt{HYPERTREE} \preceq \texttt{CHYPERTREE}$. The result then follows by Theorem 6.1 and by Proposition 2.1(2). □

Note that in this paper we are mostly interested in graph-based representations, and thus the above relationships do suffice to our purpose, because we shall exploit QUERYDECOMP as an interface between graph methods and hypergraph methods. We thus omit a formal treatment of the notions of HYPERTREE, SPREADCUT, and CHYPERTREE, by referring the reader interested in expanding on this subject to [5,12,13]. Rather, we next shall focus on comparing QUERYDECOMP with the various binary method discussed in this paper, by showing that it strongly generalizes all of them. In particular, in the light of our previous results (see, also, Fig. 4), we shall just focus on the methods $\texttt{TREEDECOMP}^{optd}$, $\texttt{TREEDECOMP}^{in}$, and $\texttt{TREEDECOMP}^{p}$, which emerged as the most powerful binary methods.

Note that $\texttt{TREEDECOMP}^{optd}$ and HYPERTREE have already been compared in [10], and it turned out that $\texttt{TREEDECOMP}^{optd} \prec\prec \texttt{HYPERTREE}$. Actually, by inspecting the result in [10], it is immediate to check that the proof straightforwardly applies to QUERYDECOMP as well, and thus the following result holds, too.

Theorem 6.4. $\texttt{TREEDECOMP}^{optd} \prec\prec \texttt{QUERYDECOMP}$.

Thus, we next shall consider the comparison with $\texttt{TREEDECOMP}^{in}$.

Theorem 6.5. $\texttt{TREEDECOMP}^{in} \prec\prec \texttt{QUERYDECOMP}$.

Proof. Recall that $\texttt{TREEDECOMP}^{in} \precsim \texttt{TREEDECOMP}^{optd}$ (and, hence, $\texttt{TREEDECOMP}^{optd} \rhd \texttt{TREEDECOMP}^{in}$) and $\texttt{TREEDECOMP}^{optd} \prec\prec \texttt{QUERYDECOMP}$ (and, hence, $\texttt{TREEDECOMP}^{optd} \preceq \texttt{QUERYDECOMP}$) hold by Theorems 4.10 and 6.4, respectively. Hence, we can apply Proposition 2.1(1) for concluding that $\texttt{QUERYDECOMP} \rhd \texttt{TREEDECOMP}^{in}$ holds as well.

We conclude by recalling that $\texttt{TREEDECOMP}^{in} \preceq \texttt{QUERYDECOMP}$, that is, any class of constraints that is tractable according to $\texttt{TREEDECOMP}^{in}$ is also tractable according to QUERYDECOMP. Indeed, Chekuri and Rajaraman [3] proved that, if $\mathcal{H}$ is

a hypergraph and $TD = \langle T, \chi \rangle$ is a tree decomposition of $inc(\mathcal{H})$ whose width is k, then there exists a query decomposition QD of $\mathcal{H}$ having width $k+1$. □

Armed with the above theorems, we can now conclude that non-binary methods are definitively better than binary ones, as far as their ability to identify islands of tractability is concerned.

Theorem 6.6. *Let* D *be any decomposition method such that* QUERYDECOMP $\preceq$ D. *Then, all the following relationships hold*:

(1) TREEDECOMP^{in} $\prec\prec$ D;
(2) TREEDECOMP^{optd} $\prec\prec$ D; *and*
(3) TREEDECOMP^{p} $\prec\prec$ D.

Proof. By Theorems 6.5 and 6.4, we known that TREEDECOMP^{in} $\prec\prec$ QUERYDECOMP and TREEDECOMP^{optd} $\prec\prec$ QUERYDECOMP hold, respectively. Thus, we can apply Proposition 2.1(3) in order to conclude that TREEDECOMP^{in} $\prec\prec$ D and TREEDECOMP^{optd} $\prec\prec$ QUERYDECOMP hold, as well.

Eventually, the fact that TREEDECOMP^{p} $\prec\prec$ D derives by applying again Proposition 2.1(3) armed with the facts that TREEDECOMP^{in} $\prec\prec$ D (and, hence, TREEDECOMP^{in} $\preceq$ D) and that, by Theorem 5.2, TREEDECOMP^{p} $\prec\prec$ TREEDECOMP^{in}. □

In particular, we can contextualize the result to HYPERTREE, SPREADCUT, and CHYPERTREE.

Corollary 6.7. *Let* D_1 *be any method in* {HYPERTREE, SPREADCUT, CHYPERTREE} *and* D_2 *be any method in* {TREEDECOMP^{in}, TREEDECOMP^{optd}, TREEDECOMP^{p}}. *Then,* $D_2 \prec\prec D_1$.

Proof. It is sufficient to apply Theorem 6.6, by observing that HYPERTREE, SPREADCUT, and CHYPERTREE generalize QUERYDECOMP, by Theorems 6.1, 6.2, and 6.3, respectively. □

6.2. A note on the power of hypergraph-based methods

According to Theorem 6.6, any method that generalizes QUERYDECOMP is more powerful than any arbitrary binary method. At a first sight, this result may appear surprising, especially when comparing QUERYDECOMP with TREEDECOMP^{optd}, given that the kind of tree labelling in these methods (both hyperedge-based) seems rather similar. Next, we clarify why there is such a big difference between TREEDECOMP^{optd} and QUERYDECOMP, by exploiting the notion of weak query decomposition introduced in Section 3.3 as a technical tool for the comparison.

In fact, we have already observed that condition (3) in the definition of the weakQUERYDECOMP method (see Definition 3.11) entails condition (3) in the definition of the QUERYDECOMP method (see Section 2). Thus, any weak query decomposition of a hypergraph is also a query decomposition of the hypergraph.

However, it is worthwhile noting that weakQUERYDECOMP misses the ability of jointly using sets of arbitrary hyperedges to maintain the connectedness among the nodes that are to be decomposed. Indeed, in the standard query decomposition, variables are covered by the union of the hyperedges in the λ labelling, while in the above weaker notion only one hyperedge at a time may be used (see the role of h'' in condition (3) of Definition 3.11). As a matter of fact, this ability is crucial in the definition of hypergraph-based methods, given that QUERYDECOMP strongly generalizes all binary methods, and that weakQUERYDECOMP instead emerged to have the same decomposition power as TREEDECOMP^{optd} (cf. Corollary 3.14). Therefore, these observations suggest that the more liberal condition (3) in QUERYDECOMP is at the basis of the superior power of hypergraph-based methods (for non-binary instances).

7. Conclusions

In this paper, we have outlined a complete picture of the power of decomposition approaches applied to binary representations of non-binary CSP instances and of decomposition methods specifically tailored for non-binary encodings. Our research provides a clear and complete comparison of the structural decomposition techniques proposed in the literature so far, and answers some long-standing questions pertaining the relationships among such methods. For completeness we mention the recently proposed notion of *fractional hypertree decomposition* [17], which is not reported in Fig. 4. In fact, it is known that this notion strongly generalizes HYPERTREE, and thus it strongly generalizes all other structural methods considered in this paper, too. However, deciding whether a fractional hypertree decomposition of width k can be computed in polynomial time, for a fixed constant k, is an open and rather intriguing problem.[5] We feel that facing this problem might provide novel insights into the actual nature of hypergraph-based structural decomposition methods.

[5] An important advance in this respect is the $O(k^3)$ polynomial-time approximation algorithm provided by Marx [19].

Appendix A. Proofs in Section 2

Proposition 2.1. *Let* $\mathtt{D}_1$, $\mathtt{D}_2$, *and* $\mathtt{D}_3$ *be three decomposition methods. Then*,

(1) If $\mathtt{D}_2 \rhd \mathtt{D}_1$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_3 \rhd \mathtt{D}_1$;
(2) *If* $\mathtt{D}_1 \preceq \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_1 \preceq \mathtt{D}_3$; *and*
(3) *If* $\mathtt{D}_1 \lessdot \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$, *then* $\mathtt{D}_1 \lessdot \mathtt{D}_3$.

Proof.

(1) $\mathtt{D}_2 \rhd \mathtt{D}_1$ means that there is number $\bar{k} > 0$ and a set of instances $\mathcal{C} \subseteq C(\mathtt{D}_2, \bar{k})$ such that $\mathcal{C} \not\subseteq C(\mathtt{D}_1, m)$, for each $m > 0$. Moreover, since $\mathtt{D}_2 \preceq \mathtt{D}_3$ holds, there is a constant δ such that $C(\mathtt{D}_2, \bar{k}) \subseteq C(\mathtt{D}_3, \bar{k} + \delta)$, and hence $\mathcal{C} \subseteq C(\mathtt{D}_3, \bar{k} + \delta)$.
(2) Assume that $\mathtt{D}_1 \preceq \mathtt{D}_2$ and $\mathtt{D}_2 \preceq \mathtt{D}_3$ hold. Since $\mathtt{D}_1 \preceq \mathtt{D}_2$, there exists a constant δ_1 such that, for each k, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, k + \delta_1)$. Moreover, since $\mathtt{D}_2 \preceq \mathtt{D}_3$ holds, there is a constant δ_2 such that, for each k', $C(\mathtt{D}_2, k') \subseteq C(\mathtt{D}_3, k' + \delta_2)$.
By combining the above two relationships, we conclude that there is a constant $\delta = \delta_1 + \delta_2$ such that, for each k, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, k + \delta_1) \subseteq C(\mathtt{D}_3, k + \delta_1 + \delta_2) = C(\mathtt{D}_3, k + \delta)$. This means that $\mathtt{D}_3$ generalizes $\mathtt{D}_1$.
(3) It immediately follows from (1) and (2) above, and by the definition of the notion of strong generalization. □

Proposition 2.2. *Let* $\mathtt{D}_1$ *and* $\mathtt{D}_2$ *be two decomposition methods. Then*,

(1) $\mathtt{D}_1 \preceq \mathtt{D}_2$ *implies that there is a function* f *such that* $\mathtt{D}_1 \preceq_f \mathtt{D}_2$; *and*
(2) $\mathtt{D}_1 \preceq_f \mathtt{D}_2$ *implies that* $\mathtt{D}_1 \rhd \mathtt{D}_2$ *and* $\mathtt{D}_2 \lessdot \mathtt{D}_1$ *do not hold*.

Proof.

(1) This is immediate for the identity function $f(x) = x$.
(2) Let $\mathtt{D}_1$ and $\mathtt{D}_2$ be two decomposition methods such that $\mathtt{D}_1 \preceq_f \mathtt{D}_2$. Then, by definition of f-generalization, for each $k > 1$, $C(\mathtt{D}_1, k)$ must indeed be contained in $C(\mathtt{D}_2, f(k))$.
Assume, for the sake of contradiction, that $\mathtt{D}_1 \rhd \mathtt{D}_2$ holds, that is, there exists a positive integer $\bar{k}$ and a set $\mathcal{C} \subseteq C(\mathtt{D}_1, \bar{k})$ such that $\mathcal{C} \not\subseteq C(\mathtt{D}_2, m)$ for any $m > 0$. Since $C(\mathtt{D}_1, 1) \subseteq C(\mathtt{D}_1, 2)$, assume without loss of generality that $\bar{k} > 1$, and consider the value $m = f(\bar{k})$. Then, there is an instance $\bar{\mathcal{H}} \in \mathcal{C} \subseteq C(\mathtt{D}_1, \bar{k})$ such that $\bar{\mathcal{H}} \notin C(\mathtt{D}_2, m) = C(\mathtt{D}_2, f(\bar{k}))$. Contradiction, since for each $k > 1$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, f(k))$.
Eventually, from the fact that $\mathtt{D}_1 \rhd \mathtt{D}_2$ does not hold, we immediately entail that $\mathtt{D}_2 \lessdot \mathtt{D}_1$ does not hold as well. □

Proposition 2.4. *Let* $\mathtt{D}_1$, $\mathtt{D}_2$, *and* $\mathtt{D}_3$ *be three decomposition methods. Then*,

(1) $\mathtt{D}_1$ *does not weakly generalize* $\mathtt{D}_1$, *i.e.*, $\widetilde{\preceq}$ *is antireflexive*;
(2) If $\mathtt{D}_1 \widetilde{\preceq} \mathtt{D}_2$, *then* $\mathtt{D}_2 \widetilde{\preceq} \mathtt{D}_1$ *does not hold, i.e.*, $\widetilde{\preceq}$ *is antisymmetric*; *and*
(3) *If* $\mathtt{D}_1 \widetilde{\preceq} \mathtt{D}_2$ *and* $\mathtt{D}_2 \preceq \mathtt{D}_3$ (*or*, $\mathtt{D}_1 \preceq \mathtt{D}_2$ *and* $\mathtt{D}_2 \widetilde{\preceq} \mathtt{D}_3$), *then* $\mathtt{D}_3$ *exp-generalizes* $\mathtt{D}_1$.

Proof.

(1) The result follows by observing that condition (ii) in the definition of the weak generalization, i.e., $\mathtt{D}_1 \rhd \mathtt{D}_1$, cannot be satisfied since $\rhd$ is trivially antireflexive.
(2) Let $\mathtt{D}_1$ and $\mathtt{D}_2$ be two decomposition methods such that $\mathtt{D}_1 \widetilde{\preceq} \mathtt{D}_2$. Then, due to condition (i) in Definition 2.3, we have that $\mathtt{D}_2 \rhd \mathtt{D}_1$, which entails that $\mathtt{D}_2 \widetilde{\preceq} \mathtt{D}_1$ does not hold, after Proposition 2.2(2).
(3) Let us first consider the case where both $\mathtt{D}_1 \widetilde{\preceq} \mathtt{D}_2$ and $\mathtt{D}_2 \preceq \mathtt{D}_3$ hold. By condition (ii) in Definition 2.3, $\mathtt{D}_1 \widetilde{\preceq} \mathtt{D}_2$ implies that there exists a constant δ_1 such that, for each $k > 1$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, 2^{k^{\delta_1}})$. Moreover, since $\mathtt{D}_2 \preceq \mathtt{D}_3$ holds, there is a constant δ_2 such that, for each $k > 0$, $C(\mathtt{D}_2, k) \subseteq C(\mathtt{D}_3, k + \delta_2)$. By combining the above two relationships we get, for each $k > 1$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, 2^{k^{\delta_1}}) \subseteq C(\mathtt{D}_3, 2^{k^{\delta_1}} + \delta_2) \subseteq C(\mathtt{D}_3, 2^{k^{\delta}})$, where $\delta > \delta_1$ is some fixed positive rational greater than 1 and large enough so that $2^{2^{\delta}} \geqslant 2^{2^{\delta_1}} + \delta_2$. By definition, this means that $\mathtt{D}_3$ *exp*-generalizes $\mathtt{D}_1$.
Let us now consider the case where $\mathtt{D}_1 \preceq \mathtt{D}_2$ and $\mathtt{D}_2 \widetilde{\preceq} \mathtt{D}_3$ hold. By condition (ii) in Definition 2.3, $\mathtt{D}_2 \widetilde{\preceq} \mathtt{D}_3$ implies that there exists a constant δ_2 such that, for each $k > 1$, $C(\mathtt{D}_2, k) \subseteq C(\mathtt{D}_3, 2^{k^{\delta_2}})$. Moreover, since $\mathtt{D}_1 \preceq \mathtt{D}_2$ holds, there is a constant δ_1 such that, for each $k > 0$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, k + \delta_1)$, and of course this relationships holds for any number greater than δ_1, as well. We thus choose as δ_1 a number large enough so that $k^{\delta_1 - 1} > \delta_1$, for every $k > 1$. Note that, in particular, this entails $\delta_1 > 2$. By combining the above two relationships we get, for each $k > 1$, $C(\mathtt{D}_1, k) \subseteq C(\mathtt{D}_2, k + \delta_1) \subseteq C(\mathtt{D}_3, 2^{(k+\delta_1)^{\delta_2}}) \subseteq C(\mathtt{D}_3, 2^{(k\delta_1)^{\delta_2}}) \subseteq C(\mathtt{D}_3, 2^{k^{\delta_1 \delta_2}})$, because of the choice of δ_1. This means that $\mathtt{D}_3$ *exp*-generalizes $\mathtt{D}_1$. □

References

[1] F. Bacchus, X. Chen, P. van Beek, T. Walsh, Binary vs non-binary constraints, Artificial Intelligence 140 (1–2) (2002) 1–37.
[2] A. Bernstein, N. Goodman, The power of natural semijoins, SIAM Journal on Computing 10 (4) (1981) 751–771.
[3] Ch. Chekuri, A. Rajaraman, Conjunctive query containment revisited, Theoretical Computer Science 239 (2) (2000) 211–229.
[4] D. Cohen, P. Jeavons, The complexity of constraint languages, in: F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, 2006.
[5] D. Cohen, P. Jeavons, M. Gyssens, A unified theory of structural tractability for constraint satisfaction problems, Journal of Computer and System Sciences 74 (5) (2008) 721–743.
[6] R. Dechter, Constraint Networks, second edition, Encyclopedia of Artificial Intelligence, Wiley and Sons, 1992, pp. 276–285.
[7] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, New York, 1999.
[8] E.C. Freuder, A sufficient condition for backtrack-bounded search, Journal of the ACM 32 (4) (1985) 755–761.
[9] N. Goodman, O. Shmueli, Syntactic characterization of tree database schemas, Journal of the ACM 30 (4) (1983) 767–786.
[10] G. Gottlob, N. Leone, F. Scarcello, A comparison of structural CSP decomposition methods, Artificial Intelligence 124 (2) (2000) 243–282.
[11] G. Gottlob, N. Leone, F. Scarcello, Computing LOGCFL certificates, Theoretical Computer Science 270 (1–2) (2002) 761–777.
[12] G. Gottlob, N. Leone, F. Scarcello, Hypertree decompositions and tractable queries, Journal of Computer and System Sciences 64 (3) (2002) 579–627.
[13] G. Gottlob, Z. Miklos, T. Schwentick, Generalized hypertree decompositions: NP-hardness and tractable variants, in: Proc. of the 26th ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS'07), 2007, pp. 13–22.
[14] G. Gottlob, S. Szeider, Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems, The Computer Journal 51 (3) (2008) 303–325.
[15] M. Gyssens, P.G. Jeavons, D.A. Cohen, Decomposing constraint satisfaction problems using database techniques, Artificial Intelligence 66 (1994) 57–89.
[16] M. Gyssens, J. Paredaens, A decomposition methodology for cyclic databases, in: Advances in Database Theory, vol. 2, Plenum Press, New York, NY, 1984, pp. 85–122.
[17] M. Grohe, D. Marx, Constraint solving via fractional edge covers, in: Proc. of the 17th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'06), 2006, pp. 289–298.
[18] Ph.G. Kolaitis, M.Y. Vardi, Conjunctive-query containment and constraint satisfaction, Journal of Computer and System Sciences 61 (2) (2000) 302–332.
[19] D. Marx, Approximating fractional hypertree width, in: Proc. of the 19th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'09), 2009, pp. 902–911.
[20] N. Robertson, P.D. Seymour, Graph minors II. Algorithmic aspects of tree width, Journal of Algorithms 7 (1986) 309–322.
[21] M. Samer, S. Szeider, Constraint satisfaction with bounded treewidth revisited, Journal of Computer and System Sciences 76 (2) (2010) 103–114.
[22] F. Scarcello, G. Gottlob, G. Greco, Uniform constraint satisfaction problems and database theory, in: Complexity of Constraints, in: LNCS, vol. 5250, Springer, 2008, pp. 156–195.
[23] R. Seidel, A new method for solving constraint satisfaction problems, in: Proc. of the 7th International Joint Conference on Artificial Intelligence (IJCAI'81), 1981, pp. 338–342.