Treewidth and Hypertree Width*

Georg Gottlob[1], Gianluigi Greco[2], and Francesco Scarcello[2]

Oxford University[1] and University of Calabria[2]
georg.gottlob@cs.ox.ac.uk, ggreco@mat.unical.it, scarcello@deis.unical.it

Abstract

The chapter covers methods for identifying islands of tractability for NP-hard combinatorial problems by exploiting suitable properties of their graphical structure. Acyclic structures are considered, as well as nearly acyclic ones identified by means of so-called structural decomposition methods. In particular, the chapter focuses on the tree decomposition method, which is the most powerful decomposition method for graphs, and on the hypertree decomposition method, which is its natural counterpart for hypergraphs. These problem-decomposition methods give rise to corresponding notions of width of an instance, namely, treewidth and hypertree width. It turns out that many NP-hard problems can be solved efficiently over classes of instances of bounded treewidth or hypertree width: deciding whether a solution exists, computing a solution, and even computing an optimal solution (if some cost function over solutions is specified) are all polynomial-time tasks. Example applications include problems from artificial intelligence, databases, game theory, and combinatorial auctions.

1 Introduction

Many NP-hard problems in different areas such as AI [42], Database Systems [7, 81], Game theory [45, 31, 20], and Network Design [34], are known to be efficiently solvable when restricted to instances whose underlying structures can be modeled via acyclic graphs or acyclic hypergraphs. For such restricted classes of instances, solutions can usually be computed via dynamic programming. However, as a matter of fact, (graphical) structures arising from real applications are in most relevant cases not properly acyclic. Yet, they are often not very intricate and exhibit some rather limited degree of cyclicity, which suffices to retain most of the nice properties of acyclic instances. Therefore, many efforts have been spent to investigate graph and hypergraph properties that are best suited to identify nearly-acyclic graph/hypergraphs, leading to the definition of a number of so-called *structural decomposition methods*.

In order to apply a decomposition method to a given problem, one first needs to describe the structure of the problem through a graph or a hypergraph. This means that, to each problem instance I, one associates a graph $G(I)$ or a hypergraph $H(I)$. Then, $G(I)$ or $H(I)$ is decomposed into possibly overlapping chunks that have a tree-like (i.e., acyclic) interconnection pattern. The resulting data structure is called a *graph or hypergraph decomposition*. The *width of a*

*Preliminary version of a chapter in "Tractability: Practical Approaches to Hard Problems", ed. Lucas Bordeaux, Youssef Hamadi and Pushmeet Kohli. Cambridge University Press, 2014.

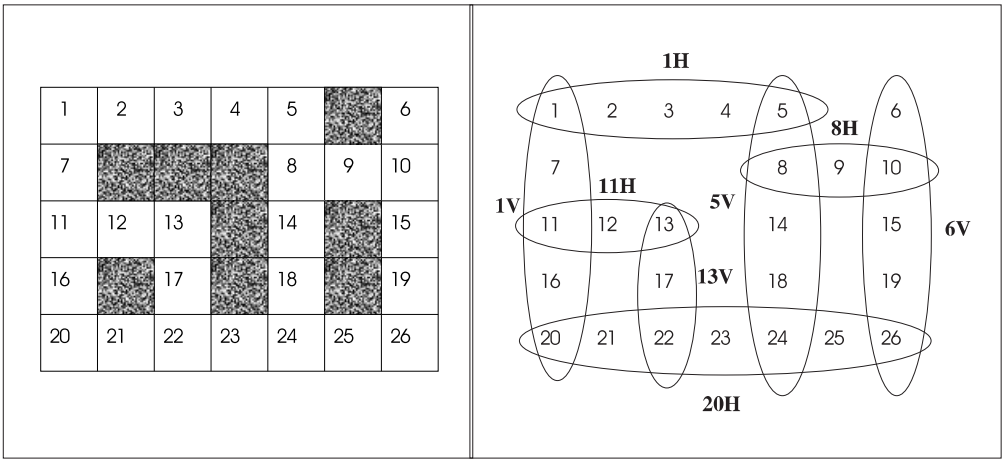

Figure 1: A crossword puzzle and its representation as a hypergraph $\mathcal{H}_0$.

decomposition of a problem instance I corresponds to the size of the largest chunk occurring in the decomposition. The *width of the instance* I is then defined as the minimum width over all decompositions of I.

An overwhelming number of relevant decision or computation problems admit solution algorithms whose runtime is exponential in $O(w)$ where w is the width of the input instance. This means that for classes of bounded width, these problems are solvable in polynomial time. Given that many practical problem instances occurring in real life applications tend to be of low width, decomposition methods are currently among the most effective weapons against NP-hardness.

The structure of many problems is adequately described by graphs. In particular, this is the case when a graph is explicitly part of the problem instance, such as in graph coloring or network problems, or when the problem is about a binary relationship, such as in matching problems, binary constraint networks, or precedence orderings, for example, for major versions of job shop scheduling. For many other problems, however, a graphical representation in terms of hypergraphs is more appropriate. This is usually the case when relations of unbounded arities or families of sets are part of the problem description. For example, in the crossword puzzle depicted on the left of Figure 1, empty fields, that are placeholders for letters, are grouped together to form placeholders for words. In general, a word-field consists of several letter-fields. The structure of such a puzzle is thus best described in terms of a hypergraph, as illustrated on the right of Figure 1. Examples for other problems whose structure is most adequately described by hypergraphs are general constraint satisfaction problems, conjunctive database queries, and combinatorial auctions, which will all be explained in Section 4. Examples where other notions of problem structures (not necessarily graph-based) are more useful for identifying tractable instances are described in other chapters of this book.

This chapter focuses on two relevant decomposition methods for (hyper)graph based structures: the *treewidth*, which is the most powerful decomposition method on graphs, and the *hypertree width*, which is its natural counter-part over hypergraphs. Both methods are specializations of a more general decomposition scheme called *tree projection*, which we will briefly illustrate in Section 5.

The rest of this chapter is organized as follows. In Section 2 we review the notion of *treewidth*, and in Section 3 the notion of *(generalized) hypertree width*, by providing their direct definitions, looking at their connections, and giving pointers to their most recent extensions. A number of

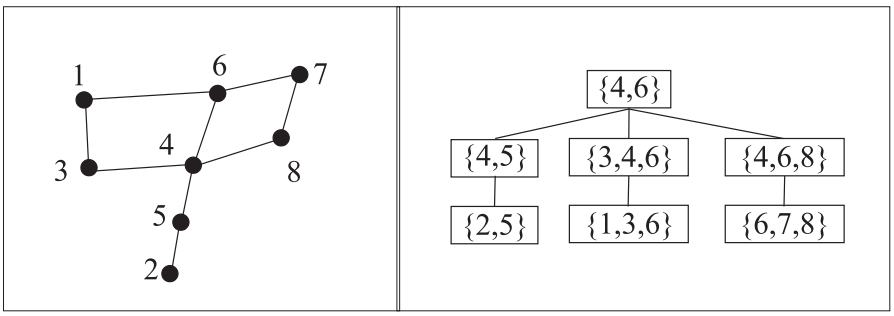

Figure 2: A graph G_0 and a tree decomposition for it.

applications of such decomposition methods are illustrated. In particular, Section 4 discusses tractability results for the *constraint satisfaction (optimization)* problem (CSP), as this is a fundamental framework which is able to express many problems from different fields. Moreover, our current knowledge on the tractability frontier for such problems is illustrated in Section 6.

2 Treewidth

The concept of treewidth [72], based on tree decompositions of graphs, constitutes a significant success story of Theoretical Computer Science.

There are different possible notions to measure how far a graph is from a tree, that is, to measure its degree of cyclicity or, dually, its tree-likeness (see, e.g., [36]). Among them, the treewidth is provably the most powerful one, in that it is able to extend the nice computational properties of trees to the largest possible classes of graphs, in many applications from different fields.

Definition 2.1 ([72])**.** A *tree decomposition* of a graph $G = (N, E)$ is a pair $\langle T, \chi \rangle$, where $T = (V, F)$ is a tree, and χ is a labeling function assigning to each vertex $p \in V$ a set of vertices $\chi(p) \subseteq N$, such that the following three conditions are satisfied: (1) for each node b of G, there exists $p \in V$ such that $b \in \chi(p)$; (2) for each edge $(b, d) \in E$, there exists $p \in V$ such that $\{b, d\} \subseteq \chi(p)$; and (3) for each node b of G, the set $\{p \in V \mid b \in \chi(p)\}$ induces a connected subtree of T.

The *width* of $\langle T, \chi \rangle$ is the number $\max_{p \in V}(|\chi(p)| - 1)$. The *treewidth* of G, denoted by $tw(G)$, is the minimum width over all its tree decompositions. □

Note that treewidth is a true generalization of graph acyclicity. Indeed, a graph G is acyclic if and only if $tw(G) = 1$.

For example, the graph G_0 reported in Figure 2 is cyclic and its treewidth is 2, as it is witnessed by the width-2 tree decomposition depicted in the same figure.

Complexity of Treewidth. To determine the treewidth of a graph G is NP-hard. However, for each fixed natural number k, checking whether $tw(G) \leq k$, and if so, computing a tree decomposition for G of optimal width, is achievable in linear time [9], and was recently shown to be achievable in logarithmic space [26]. Note that the multiplicative constant factor of Bodlaenders linear algorithm [9] is exponential in k. However, there are algorithms that find exact tree decompositions in reasonable time or good upper approximations in many cases of practical relevance—see, for example, [10, 11] and the references therein.

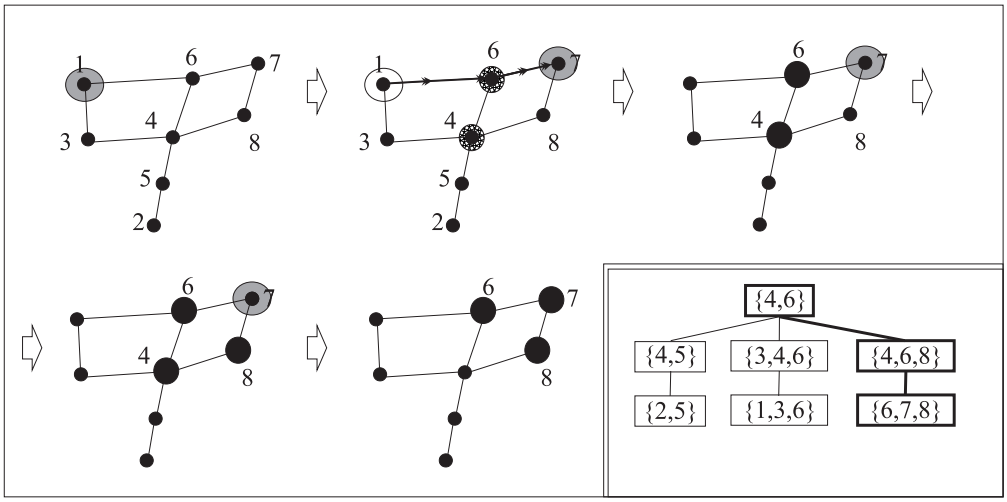

Figure 3: The robber and cop game played on the graph G_0 of Figure 2.

Game-Theoretic Characterization. An alternative definition of treewidth is based on the *robber and cops* game, which is played on a graph $G = (N, E)$ by a robber and a set of cops. The robber stands on a node and can run at great speed along the edges of G; however, she is not permitted to run trough a node that is controlled by a cop. Note that the robber is fast and may see cops that are entering in action. Therefore, while cops move, the robber may run trough those positions that are left by cops or not yet occupied. The goal of the cops is to occupy the vertex on which the robber stands, while the robber tries to avoid her capture. A graph has treewidth bounded by k if and only if $k + 1$ cops can capture the robber [78].

Example 2.2. Consider the robber and cops game played on the graph G_0 of Figure 2, and the moves illustrated in Figure 3: The robber initially stands on node 1. Then, two cops enter in action by occupying 4 and 6. While these cops are moving, the robber can go to node 7 in a very fast way. Then, another cop comes into play by occupying node 8, thus blocking the robber at node 7. Eventually, the cop that is currently placed at node 4 moves to node 7, and hence captures the robber. Note that the sequence of moves leading to the capture of the robber corresponds to one branch of the tree decomposition of G_0 depicted in Figure 2 (and replicated in Figure 3, with such branch being evidenced). In fact, the correspondence is not by chance: the depicted width-2 tree decomposition can be seen as encoding a "winning strategy" for 3 cops. $\lhd$

Note that in the game there is no restriction on the strategy employed by cops to capture the robber. In particular, they are not forced to play *monotonic strategies*, that is, to shrink the robber's escape space in a monotonically decreasing way. However, it was shown in [78] that playing non-monotonic strategies gives no more power to cops. Many results about treewidth are proved in a simple and elegant way by exploiting the game-theoretic characterization. In particular, the above equivalence between monotonic and non-monotonic capturing strategies turns out to be very useful, because good strategies for the robber may be easily characterized as those strategies that allow the robber to run forever. See [3], for an interesting application of the robber-and-cops game in proofs regarding the power of *k-Consistency* in *constraint satisfaction problems*.

2.1 Applications to Decision Problems

Tree decompositions and polynomial algorithms for bounded treewidth are among the most effective weapons to attack NP-hard problems, namely, by recognizing and efficiently solving large classes of tractable problem instances. In particular, the notion of treewidth is at the base of strong meta-theorems such as Courcelle's Theorem [18], which states that any problem expressible in monadic second-order logic (MSO) over structures of bounded treewidth can be solved in linear time. Many problems are easily expressed in terms of MSO, and thus Courcelle's theorem turns out to be a very effective tool for obtaining tractability results.

Finite Structures. The notion of treewidth is easily generalized from graphs to finite structures. A vocabulary τ is a finite set of relation symbols $R_1, \ldots, R_k$ of arities $a_1, \ldots, a_k$, respectively. A relational structure $\mathcal{A}$ over τ consists of a finite domain A and an a_i-ary relation $R_i^{\mathcal{A}} \subseteq A^{a_i}$, for each relation symbol R_i in τ. The *size* of $\mathcal{A}$, denoted by $||\mathcal{A}||$, is the value $||\mathcal{A}|| = |A| + \sum_{j=1}^{k} |R_j| \times a_j$. For further background on finite structures, the interested reader is referred to textbooks on finite model theory (e.g., [44]).

For instance, a graph $G = (N, E)$ can be viewed as a finite structure whose domain is N, and where E is a binary relation encoding its edges.

The *Gaifman graph* of a structure $\mathcal{A}$ is the undirected graph $G(\mathcal{A})$ whose vertices are the elements of the domain of $\mathcal{A}$, and where there is an edge between the elements e and e' if and only if there is a tuple of some relation of $\mathcal{A}$ where e and e' jointly occur. The treewidth of $\mathcal{A}$, denoted by $tw(\mathcal{A})$, is the treewidth of its Gaifman graph, i.e., $tw(\mathcal{A}) = tw(G(\mathcal{A}))$.

MSO. A First Order logic formula is made up of relation symbols, individual variables (usually denoted by lowercase letters), the logical connectives $\vee$, $\wedge$, and $\neg$, and the quantifiers $\exists$ and $\forall$. Monadic Second Order (MSO) enhances the expressiveness of first order logic by allowing the use of set variables (usually denoted by uppercase letters), of the membership relation $\in$, and of the quantifiers $\exists$ and $\forall$ over set variables. In addition, it is often convenient to use symbols like $\subseteq$, $\subset$, $\cap$, $\cup$, and $\rightarrow$ with their usual meaning, as abbreviations. When an MSO formula ϕ is evaluated over a finite structure $\mathcal{A}$, the relation symbols of ϕ are interpreted as the corresponding relations of $\mathcal{A}$ and the variables of ϕ range over the domain A of $\mathcal{A}$. The fact that an MSO formula ϕ holds over $\mathcal{A}$ is denoted by $\mathcal{A} \models \phi$. For a graph G (viewed as a finite structure), this is just meant to state that G satisfies the property expressed by the formula ϕ, as we illustrate below.

Example 2.3. Let $G = (N, E)$ be an undirected graph (interpreted as a finite structure). Then, the fact that G is *3-colorable* can be expressed via the following MSO formula:

$$\begin{aligned}
\exists R, B, Y, \quad & R \cup B \cup Y = N \wedge \\
& R \cap B = \emptyset \ \wedge \ R \cap Y = \emptyset \ \wedge \ B \cap Y = \emptyset \wedge \\
& \forall x, \ x \in B \rightarrow (\forall y, \{x, y\} \in E \rightarrow \neg(y \in B)) \ \wedge \\
& \forall x, \ x \in R \rightarrow (\forall y, \{x, y\} \in E \rightarrow \neg(y \in R)) \ \wedge \\
& \forall x, \ x \in Y \rightarrow (\forall y, \{x, y\} \in E \rightarrow \neg(y \in Y))
\end{aligned}$$

In particular, note that the formula checks whether there exists a partition of the nodes in N into three disjoint sets of nodes R, B, and Y, which respectively correspond to the nodes that are colored red, blue, and yellow. Moreover, the formula checks that for each node, all its adjacent nodes are colored with a different color. $\triangleleft$

The next theorem relates treewidth to MSO.

Theorem 2.4. *Let ϕ be a fixed* MSO *sentence, let k be a fixed constant, and let $\mathcal{C}_k$ be a class of finite structures having treewidth bounded by k. Then, for each finite structure $\mathcal{A} \in \mathcal{C}_k$, deciding whether $\mathcal{A} \models \phi$ holds is feasible in linear time* [18] *and logarithmic space* [26] *(w.r.t. $||\mathcal{A}||$).*

From the above theorem and Example 2.3, we can conclude that 3-colorability is a property that can be efficiently checked on classes of graphs having bounded treewidth, while, on arbitrary classes of graphs, the problem is a well-known NP-complete problem.

2.2 Applications to Optimization Problems

An important generalization of MSO formulae to *optimization* problems was presented by [5].

Let $\mathcal{A}$ be a finite structure over the domain A, and let w be a list of weights associated with the elements in A, such that $w(v)$ is a rational number for each $v \in A$. The pair $\langle \mathcal{A}, w \rangle$ is hereinafter called a *weighted finite structure*, and its size $||\langle \mathcal{A}, w \rangle||$ is defined as the size of $\mathcal{A}$ plus all the values (numerators and denominators) in w.

Let $\phi(\bar{X})$ be an MSO formula over $\mathcal{A}$, where $\bar{X}$ is the set of free second-order variables (i.e., set variables) occurring in ϕ. For an interpretation $\mathcal{I}$ mapping variables in $\bar{X}$ to subsets of A, we denote by $\phi[\mathcal{I}]$ the MSO formula (without free variables) where each variable $X \in \bar{X}$ is replaced by $\mathcal{I}(X)$.

A *solution* to ϕ over $\langle \mathcal{A}, w \rangle$ is an interpretation $\mathcal{I}$ such that $\mathcal{A} \models \phi[\mathcal{I}]$ holds. The *cost* of $\mathcal{I}$ is the value $\sum_{X \in \bar{X}} \sum_{v \in \mathcal{I}(X)} w(e)$. A solution of minimum cost is said *optimal*.

Example 2.5. Let $G = (N, E)$ be an undirected graph (interpreted as a finite structure). Then, the property that a set X of vertices is a *vertex cover*, i.e., a set such that each edge in E has at least one endpoint incident on it, can be expressed via the *vertexCover*(X) formula (where X is its free variable) defined as follows:

$$X \subseteq N \wedge (\forall x \in N \forall y \in N, \{x, y\} \in E \rightarrow (x \in X) \vee (y \in X))$$

By considering a list w of weights assigning 1 to each vertex in N, we have that an optimal solution to *vertexCover* over $\langle G, w \rangle$ is a minimum-cardinality vertex cover. $\lhd$

The result below shows that not only the decision problem, but even the associated problem of *computing* a solution of minimum cost is feasible in polynomial time on bounded-treewidth structures.

Theorem 2.6 (simplified from [5])**.** *Let ϕ be a fixed* MSO *sentence, let k be a fixed constant, and let $\mathcal{C}_k$ be a class of finite structures having treewidth bounded by k. Then, for each weighted finite structure $\langle \mathcal{A}, w \rangle$ such that $\mathcal{A} \in \mathcal{C}_k$, computing an optimal solution to ϕ over $\langle \mathcal{A}, w \rangle$ is feasible in polynomial time (w.r.t. $||\langle \mathcal{A}, w \rangle||$).*

From the above theorem and Example 2.5, we can immediately conclude that computing a minimum-cardinality vertex cover is feasible in polynomial time on classes of graphs having bounded treewidth whereas, on arbitrary classes of graphs, it is NP-hard.

3 Hypertree Width

The structure of a computational problem is sometimes better described by a hypergraph rather than by a graph. This is, in particular, the case if local relationships involve many elements together, such as in the case of relational structures with large arities. Therefore, various width-notions for hypergraphs have been defined and studied, and often these are more effective than simply applying the treewidth on a suitable "binarization" [36, 47].

Width-notions for hypergraphs come as generalizations of *hypergraph acyclicity*, which is recalled next.

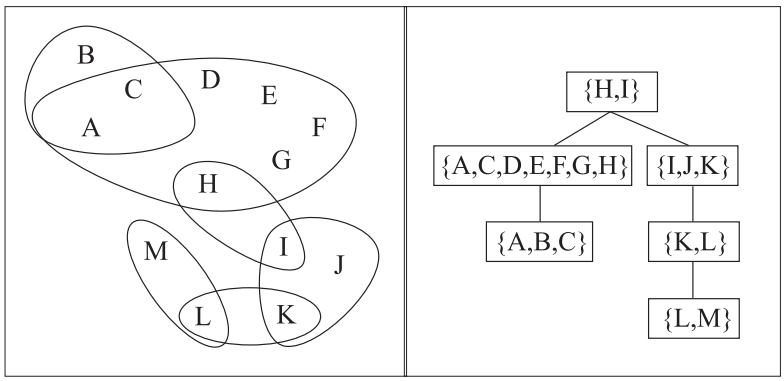

Figure 4: A hypergraph $\mathcal{H}_1$ and a join tree $\mathcal{JT}(\mathcal{H}_1)$.

A hypergraph $\mathcal{H}$ is *acyclic* iff it has a join tree [7]. A *join tree* $\mathcal{JT}(\mathcal{H})$ for a hypergraph $\mathcal{H}$ is a tree whose vertices are the hyperedges of $\mathcal{H}$ such that, whenever the same node $X \in V$ occurs in two hyperedges h_1 and h_2 of $\mathcal{H}$, then X occurs in each vertex on the unique path linking h_1 and h_2 in $\mathcal{JT}(\mathcal{H})$ (connectedness condition for X). Note that this notion of acyclicity is the most general one known in the literature, coinciding with α-acyclicity according to Fagin [27].

For example, the hypergraph $\mathcal{H}_1$ shown on the left of Figure 4 is acyclic as it is witnessed by the join tree $\mathcal{JT}(\mathcal{H}_1)$. Instead, the reader can check that the hypergraph $\mathcal{H}'_1$ shown on the left of Figure 5, obtained by adding an edge $\{B, M\}$ to $\mathcal{H}_1$, is not acyclic. Indeed, there is no way to build a join tree for it. For instance, the reader may check that every attempt to add a vertex for $\{B, M\}$ to the join tree shown in Figure 4 does not satisfy the connectedness condition for B or M. Similarly, it can be seen that the hypergraph $\mathcal{H}_0$ reported on the right of Figure 1 is not acyclic, too.

3.1 Embedding Hypergraphs in (Hyper)trees

The natural counter-part of the tree decomposition method over hypergraphs is the notion of (generalized) hypertree decomposition [39, 38]. In the following, for a hypergraph $\mathcal{H} = (V, H)$, we denote by $\mathcal{N}(\mathcal{H})$ and $\mathcal{E}(\mathcal{H})$ the sets V and H, respectively. Moreover, its associated *primal graph* is defined over the same set $\mathcal{N}(\mathcal{H})$ of nodes and contains an edge for each pair of nodes included in some hyperedge of $\mathcal{E}(\mathcal{H})$.

A *hypertree for a hypergraph* $\mathcal{H}$ is a triple $\langle T, \chi, \lambda \rangle$, where $T = (N, E)$ is a rooted tree, and χ and λ are labeling functions that associate with each vertex $p \in N$ two sets $\chi(p) \subseteq \mathcal{N}(\mathcal{H})$ and $\lambda(p) \subseteq \mathcal{E}(\mathcal{H})$. The *width* of a hypertree is the cardinality of its largest λ label, i.e., $max_{p \in N} |\lambda(p)|$.

Hypertree decompositions are similar to tree decompositions, but for the associated notion of width, which is determined by a minimum hyperedge covering of the sets of nodes in the χ labeling.

Definition 3.1. [40] A *generalized hypertree decomposition* of a hypergraph $\mathcal{H}$ is a hypertree $HD = \langle T, \chi, \lambda \rangle$ for $\mathcal{H}$, where $\langle T, \chi \rangle$ is a tree decomposition of the primal graph of $\mathcal{H}$, and λ is a function labeling the vertices of T by sets of hyperedges of $\mathcal{H}$ such that, for each $p \in vertices(T)$, $\chi(p) \subseteq \bigcup_{h \in \lambda(v)} h$. That is, all nodes in the χ labeling are covered by hyperedges in the λ labeling.

A *hypertree decomposition* is a generalized hypertree decomposition that satisfies the following additional condition, called *Descendant Condition* or also *special condition*: $\forall p \in vertices(T)$, $\forall h \in \lambda(p)$, $h \cap \chi(T_p) \subseteq \chi(p)$, where T_p denotes the subtree of T rooted at p, and $\chi(T_p)$ the set

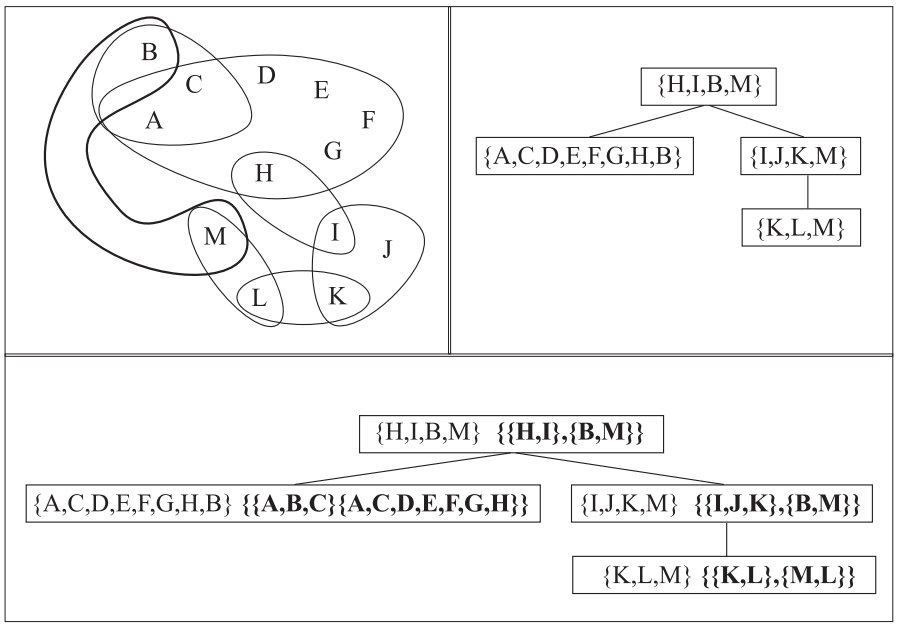

Figure 5: A hypergraph $\mathcal{H}'_1$, a tree decomposition for its primal graph, and a width-2 hypertree decomposition for $\mathcal{H}'_1$.

of all variables occurring in the χ labeling of this subtree. □

Note that the notions of hypertree width and generalized hypertree width are true generalizations of acyclicity, as the acyclic hypergraphs are precisely those hypergraphs having hypertree width and generalized hypertree width one [40].

Example 3.2. Recall the hypergraph $\mathcal{H}'_1$ shown in Figure 5. We have already observed that this hypergraph is not acyclic because it has no join tree. However, the connectedness condition for all nodes may be fulfilled if one may use additional nodes, possibly taken from multiple hyperedges, to label the desired tree. Indeed, this behavior may be observed in the tree decomposition of the primal graph of $\mathcal{H}'_1$ shown in the left part of Figure 5.

Note that the width of this decomposition is 7, because the notion of treewidth is based on the number of nodes used in each label. However, this hypergraph is evidently quasi-acyclic. And in fact the hypertree width of $\mathcal{H}'_1$ is at most 2, because all sets of nodes used in the labels may be covered by two hyperedges at most, as witnessed by the hypertree decomposition in the bottom part of Figure 5. To complete the picture, we may also conclude that the hypertree width of $\mathcal{H}'_1$ is precisely 2, because the fact that it is cyclic entails $hw(\mathcal{H}'_1) > 1$. ◁

At a first glance, a generalized hypertree decomposition may simply be viewed as a clustering of the hyperedges where the classical connectedness condition of join trees holds. However, this is not the case, as it can be seen by looking in more detail at the two labels associated with each vertex p: the set of hyperedges $\lambda(p)$, and the set of *effective* nodes $\chi(p)$, which are subject to the connectedness condition. In particular, all nodes that appear in the hyperedges of $\lambda(p)$ but that are not included in $\chi(p)$ are "ineffective" for v and do not count w.r.t. the connectedness condition.

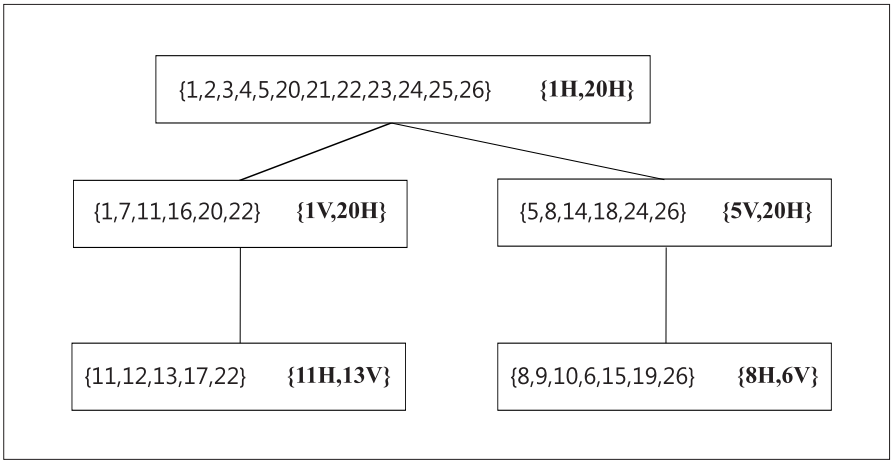

Figure 6: A width-2 hypertree decomposition for the hypergraph $\mathcal{H}_0$ reported on the right of Figure 1.

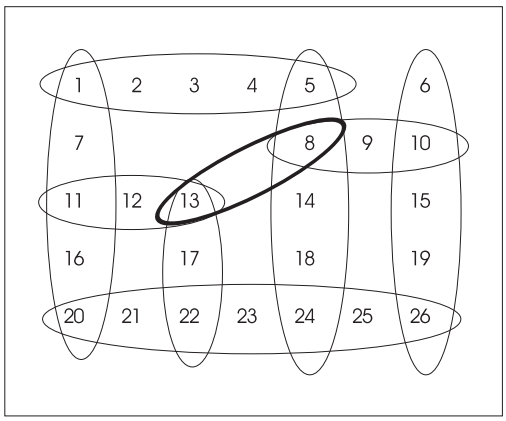

Figure 7: The hypergraph $\mathcal{H}'_0$ in Example 3.3.

Example 3.3. Reconsider the hypergraph $\mathcal{H}_0$ reported on the right of Figure 1 and associated with the crossword puzzle. The hypergraph is not acyclic, and a width-2 hypertree decomposition of it is shown in Figure 6, thus witnessing that $hw(\mathcal{H}_0) = 2$. Note, for instance, that the hyperedge $20H$ is used in 3 distinct vertices of the decomposition. In all such occurrences but for that in the root, some of the nodes in $20H$ are "ineffective".

Finally, for a further example, consider the hypergraph $\mathcal{H}'_0$ shown in Figure 7, and note that $\mathcal{H}'_0$ is obtained by adding one hyperedge to the hypergraph $\mathcal{H}_0$ shown in Figure 1. It can be checked that there exists no width-2 hypertree decomposition of $\mathcal{H}'_0$. In fact, $hw(\mathcal{H}'_0) = 3$. $\lhd$

3.2 Complexity Issues

Choosing a decomposition tree and suitable χ and λ vertex labelings in order to get a hypertree decomposition below a fixed threshold-width k is not that easy, and it is definitely more difficult than computing a simple tree decomposition.

In fact, it has been shown that generalized hypertree-width is an intractable notion, as decid-

ing whether a hypergraph has generalized hypertree width at most k is an NP-complete problem, for any fixed $k \geq 3$ [41]. It is thus very nice and somehow surprising that dealing with hypertree width is a very easy task. More precisely, for any fixed $k \geq 1$, deciding whether a given hypergraph has hypertree width at most k is in LOGCFL. For the sake of completeness, we recall here that the class LOGCFL consists of all decision problems that are logspace reducible to a context-free language, and that it contains many interesting and natural complete problems, such as evaluating (Boolean) acyclic conjunctive queries [37]. Moreover, the relationship between LOGCFL and other well-known complexity classes can be summarized in the following chain of inclusions:

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

where L denotes logspace, AC^i and NC^i are *logspace-uniform* classes based on the corresponding types of Boolean circuits, NL denotes nondeterministic logspace, and P is polynomial time—for definitions of all these classes, see [57].

Note that, since $\mathrm{LOGCFL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2$, deciding whether a given hypergraph has hypertree width at most k is a highly parallelizable problem. Correspondingly, the search problem of computing a k-bounded hypertree decomposition belongs to the functional version of LOGCFL, which is $\mathrm{L}^{\mathrm{LOGCFL}}$ [39]. However, unlike treewidth for which a linear-time algorithm exists, the problem of deciding whether, for a hypergraph $\mathcal{H}$, $hw(\mathcal{H}) \leq k$ is fixed-parameter intractable (more precisely, W[2]-hard) in the parameter k [33]. Therefore, unless some unlikely collapse occurs in fixed-parameter complexity theory, a bad exponential dependency of the form $O(f_1(n)^{f_2(k)})$ is unavoidable in the running time of sound and complete algorithms. See the chapter by Fomin and Saraubh in this collection, for an introduction to the main concepts in the theory of fixed parameter tractability.

Of course, the notion of hypertree width is less general than the notion of generalized hypertree width. However, it provides a good approximation for generalized hypertree width as, for each hypergraph $\mathcal{H}$, $ghw(\mathcal{H}) \leq hw(\mathcal{H}) \leq 3 \times ghw(\mathcal{H}) + 1$ [1].

3.3 Algorithms for Hypertree Computation

Several efforts have been spent in the last few years to define algorithms for hypertree computation. See the Hypertree Decomposition Home Page [75], for available implementations of algorithms for computing hypertree decompositions, and further links to heuristics and other papers on this subject.

Exact approaches. The first proposal in the literature appeared in [39], where an algorithm called k-`decomp` has been presented constructing a ("normal-form") hypertree decomposition of minimal width less than or equal to k (if such a decomposition exists). However, k-`decomp` "runs" on alternating Turing machines using logarithmic workspace, and hence is not designed for real-world applications. A more practical algorithm, named `opt-`k`-decomp`, has been obtained in [35] by "uprolling" k-`decomp` in a sequential bottom-up fashion. The algorithm runs in $O(m^{2k}v^2)$ time, where m and v are the number of edges and the number of nodes of the hypergraph, respectively. This algorithm has been improved subsequently in [55], where some techniques for limiting redundant computations have been discussed, which actually do not improve the asymptotic worst-case bounds of the original algorithm.

Another approach for computing hypertree decompositions has been discussed in [80]. Its basic idea is to exploit a backtracking procedure that stops as soon as it discovers a decomposition of width at most k (differently from `opt-`k`-decomp`, which implicitly builds a structure from which it is possible to enumerate easily all possible normal-form decompositions of width at most k). The (worst case) time complexity of this approach is $O(v^{3k+3})$.

Heuristics. Most recent research focuses on heuristic approaches for the construction of (generalized) hypertree decompositions of "small" but not necessarily minimal width.

For instance, generalized hypertree decompositions can be constructed starting from tree decompositions, and subsequently covering the variables at each vertex label by a small number of hyperedges. This latter condition can be straightforwardly implemented by set covering (heuristics), so that it is possible to use tree decomposition heuristics for the construction of generalized hypertree decompositions. In particular, *Bucket Elimination* [22] is used in combination with several variable ordering heuristics in [64].

Another technique has been discussed in [74], which shows how to use the *branch-decomposition* approach for ordinary graphs [17] for the heuristic construction of generalized hypertree decompositions (based on the fact that every branch decomposition of width k can be transformed into a tree decomposition of width at most $3k/2$).

Reference [60] investigates the idea of computing generalized hypertree decompositions based on the tree decompositions of the primal and of the dual graph.

The use of tabu search for computing (generalized) hypertree decompositions has been considered in [66].

More recently, [43] considered a combination of exact and heuristic approaches in the sense that the search space is restricted by a fixed upper bound k, and some heuristics are used to accelerate the search for a generalized hypertree decomposition of width at most k (but not necessarily the minimal one). The resulting algorithm `det-`k`-decomp` is based on backtracking, and can also be implemented for parallel executions.

3.4 Game-Theoretic Characterizations

Even though the formal definitions of hypertree and generalized hypertree width are quite involved, these notions (similarly to treewidth) have very natural game-theoretic characterizations. Having a game view of graph-theoretic notions not only helps grasping their meaning, but also provides a useful tool for both practical applications and formal results. For instance, game winning-strategies are often related to suitable "normal-form" decompositions. Such decompositions have been used both to speed up the computation of hypertree decompositions and to formally prove that the NP-hard notion of generalized hypertree width is in fact in NP (cf. [32]).

Hypertree Width. *The robber and marshals game* is played by one robber and a number of marshals on a hypergraph. The robber moves on nodes, while marshals move on hyperedges. At each step, any marshal controls an entire hyperedge. During a move of the marshals from the set of hyperedges E to the set of hyperedges E', the robber cannot pass through the nodes in $B = (\cup E) \cap (\cup E')$, where, for a set of hyperedges F, $\cup F$ denotes the union of all hyperedges in F. Intuitively, the vertices in B are those not released by the marshals during their move. The game is won by the marshals if they corner and capture the robber somewhere in the hypergraph, by monotonically shrinking the moving space of the robber. A hypergraph $\mathcal{H}$ has k-bounded hypertree width if, and only if, k marshals win the robber and marshals game on $\mathcal{H}$ [40].

Example 3.4. Consider the robber and marshals game played on the hypergraph $\mathcal{H}_0$ of Figure 1, and the moves illustrated in Figure 8: the robber initially stands on node 11. Then, two marshals enter in action and block hyperedges $1H$ and $20H$. During the move of the marshals, the robber is fast enough to move on any node of the hypergraph that will not be blocked by them. For instance, the robber might in principle move to node 14. In fact, the robber decides to move to 11 and, hence, (s)he is now confined to the set of nodes $\{7, 11, 12, 13, 16, 17\}$. No matter of the move of the robber, one marshal keeps blocked $20H$ while the other one occupies $1V$. Note that during this move the marshal releases all nodes in $\{2, 3, 4, 5\}$, but still blocks node 1 (which is

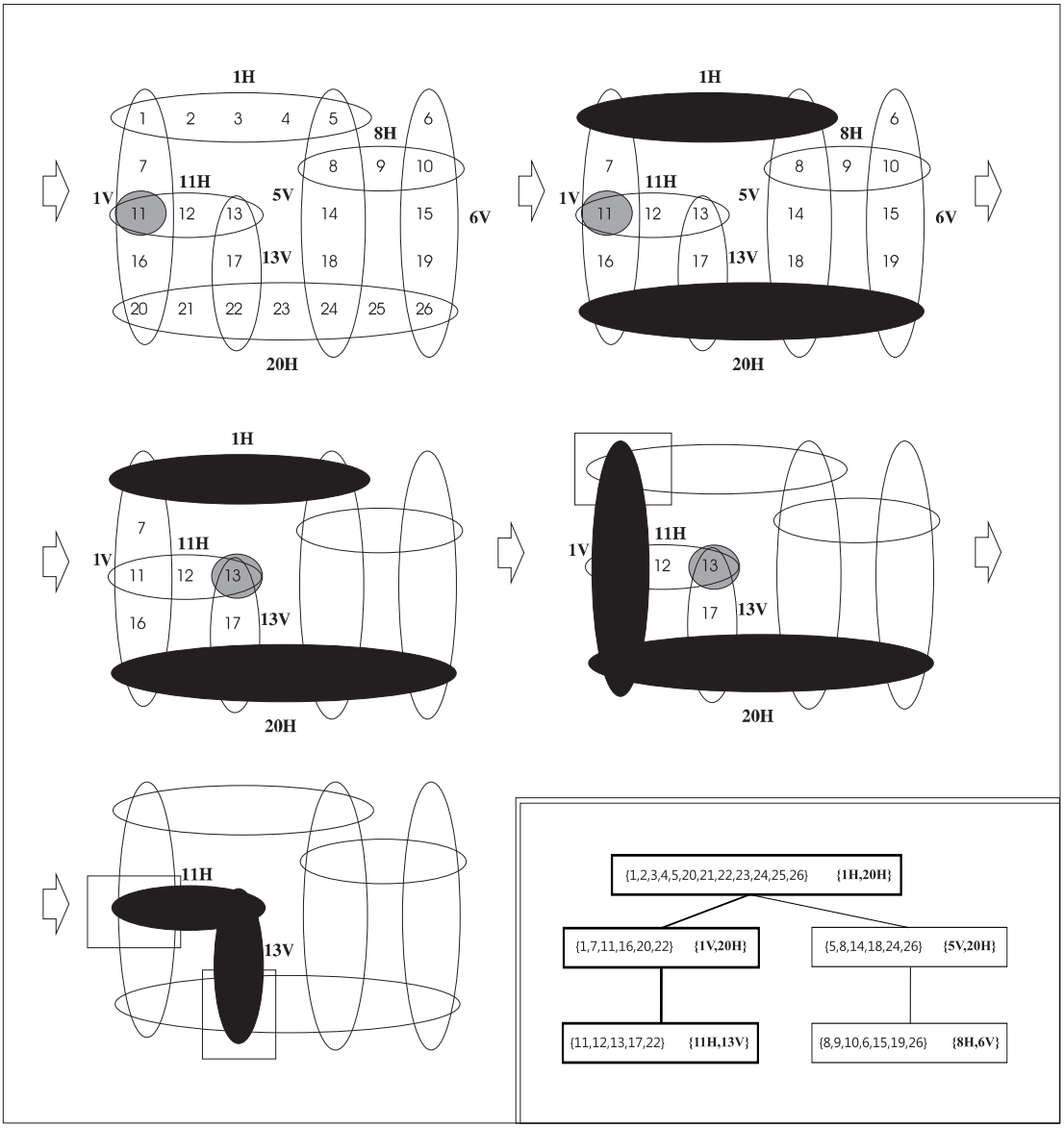

Figure 8: The robber and marshal game played on the hypergraph $\mathcal{H}_0$.

covered by both $20H$ and $1V$). It follows that the robber cannot escape via node 1, and is now confined to move over $\{12, 13, 17\}$. In fact, no matter of the next move of the robber, the two marshals can now capture the robber by moving on $11H$ and $13V$. Again, note that during this move the "escape doors" of the robber (nodes 11 and 22) remain blocked.

Note that the above described sequence of moves leading to the capture of the robber corresponds to one branch of the hypertree decomposition of $\mathcal{H}_0$ depicted in Figure 1 (and replicated in Figure 8, with such branch being evidenced). In fact, the correspondence is not by chance: the depicted width-2 hypertree decomposition can be seen as encoding a "winning strategy" for 2 marshals. $\triangleleft$

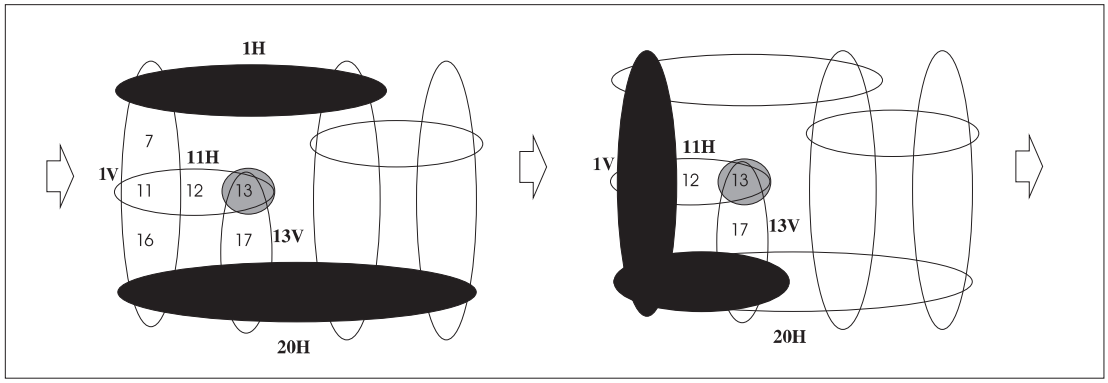

Figure 9: A move in the captain and robber game played on the hypergraph $\mathcal{H}_0$ of Figure 1.

Generalized Hypertree Width. The *captain and robber game* is played on a hypergraph, by a robber and a captain controlling some squads of cops.[1] The robber stands on a vertex and can run at great speed along the edges; however, (s)he is not permitted to run trough a vertex that is controlled by a cop. Each move of the captain involves one squad of cops, which is encoded as an edge h. The captain may ask *any subset* of cops in the squad h to run in action, as long as they occupy vertices that are currently reachable by the robber, thereby blocking an escape path for the robber. Thus, "second-lines" cops cannot be activated by the captain. The goal of the captain is to place a cop on the vertex occupied by the robber. A hypergraph $\mathcal{H}$ has k-bounded generalized hypertree width if, and only if, a captain controlling k squad of cops wins the captain and robber game on $\mathcal{H}$ [46]. Note that, in contrast with the previous game, in this case the captain is not forced to block entirely a hyperedge with a squad, and (s)he is not required to shrink monotonically the escape-space of the robber. In fact, it turns out that non-monotonic strategies give no more power to the captain [46], similarly to the game characterizing the treewidth.

Example 3.5. In order to understand the main difference between the robber and cops game and the robber and marshals game, consider again the moves depicted in Figure 8. In particular, let us focus on the first step where the two marshals occupy $1H$ and $20H$. In the captain and robber game, the same starting configuration may occur as well, because the captain may ask the two squads (associated with hyper edges) $1H$ and $20H$ to enter in action (see Figure 9). After this move, either the robber is confined to the set of nodes $\{7, 11, 12, 13, 16, 17\}$, or to the set of nodes $\{6, 8, 9, 10, 14, 15, 18, 19\}$, as in the robber and marshals game.

Consider in particular hyperedge $20H$: either nodes 23, 24, 25, 26, or nodes 20, 21, 22, and 23 are no longer reachable by the robber. Hence, they are second-lines and cannot be used by the captain. However, any subset of nodes (cops) potentially useful to actually constrain robber's moves may freely be selected to enter in action. For instance, with the former choice of the robber (say standing on node 13), in the subsequent step the captain may employ the full squad $1V$ and the two cops on nodes 20 and 21 from squad $20H$. In this case the game immediately ends with a final step where the two squads $11H$ and $13V$ enter in action, and the robber cannot move any more. $\lhd$

[1] This game is actually the specialization to generalized hypertree decompositions of the homonymous game played on pairs of hypergraphs that characterizes tree projections [46].

4 Applications of Hypertree Width

The notion of (generalized) hypertree width has been exploited profitably in the last few years to single out islands of tractability to a broad spectrum of problems in different application areas. Differently from tree decompositions, there is no theorem à la Courcelle to be used for establishing tractability results for instances having bounded hypertree width via some logic-based encodings. In this case, a useful tool to isolate tractable instances of some given problem is to establish a structure-preserving polynomial-time reduction to some problem for which islands of tractability are already known. In the following, we shall illustrate such a tool for a number of application examples, by focusing on tractable classes of the *constraint satisfaction problem* (CSP), which is able to express in a natural way many problems from different fields.

4.1 Application to Constraint Satisfaction

An instance of a *constraint satisfaction problem (CSP)* (also *constraint network*) (e.g., [23]) is a triple $I = (Var, U, \mathcal{C})$, where Var is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), where S_i is a list of variables of length m_i called the *constraint scope*, and r_i is an m_i-ary relation over U, called the *constraint relation*. (The tuples of r_i indicate the allowed combinations of simultaneous values for the variables S_i). A *solution* to a CSP instance is a substitution $\theta : Var \longrightarrow U$, such that for each $1 \leq i \leq q$, $S_i\theta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability*.

Example 4.1. A combinatorial crossword puzzle (see Figure 1) is a typical CSP [21, 69]. A set of legal words is associated to each horizontal or vertical array of white boxes delimited by black boxes. A solution to the puzzle is an assignment of a letter to each white box such that to each white array is assigned a word from its set of legal words.

This problem is represented as follows. There is a variable X_i for each white box, and a constraint C for each array D of white boxes. (For simplicity, we just write the index i for variable X_i.) The scope of C is the list of variables corresponding to the white boxes of the sequence D; the relation of C contains the legal words for D. For the example in Figure 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 $\{\langle h, o, u, s, e\rangle, \langle c, o, i, n, s\rangle, \langle b, l, o, c, k\rangle\}$. ◁

Structural Tractability. The structure of a CSP instance I can be represented by its associated hypergraph $\mathcal{H}(I) = (V, H)$, where $V = Var$ and $H = \{S \mid (S, r) \in \mathcal{C}\}$. For example, the hypergraph associated with the crossword puzzle formalized above is the one on the left of Figure 1.

Constraint satisfiability is in general NP-complete. However, bounded hypertree width of the associated hypergraph is a key for tractability. Formally, we say that a class of instances $\mathcal{C}$ has bounded hypertree width if there exists some natural number k such that $hw(\mathcal{H}(I)) \leq k$, for every $I \in \mathcal{C}$. Observe that a class $\mathcal{C}$ has bounded hypertree width if, and only if, it has bounded generalized hypertree width, after the mentioned constant-approximation relationship between the two notions [1]. We thus speak hereafter only of bounded hypertree width classes. Moreover, it is known that, for any class $\mathcal{C}$, bounded treewidth entails bounded hypertree width, while the converse does not hold in general (unless the maximum size of the constraint scopes in every

instance of $\mathcal{C}$ is bounded by a fixed constant).[2]

The following result has been discussed by different authors for different decomposition methods and (hyper)graph representations [24, 59, 14], while the more general version for bounded hypertree width (together with the completeness of the problem for the class LOGCFL) was proven in [39]. Note that the result below is purely structural, as the kind of tuples occurring in the various constraint relations do not play any role.

Theorem 4.2 ([39]). *Constraint satisfiability is feasible in polynomial time over any class $\mathcal{C}$ of bounded hypertree width.*

To establish the result, the idea is to compute a hypertree decomposition $HD = \langle T, \chi, \lambda \rangle$ of hypergraph $\mathcal{H}(I)$ of the given instance I, and then solve constraint satisfiability by traversing HD from the leaves to the root r, by means of a bottom-up procedure. Recall first that each vertex v of T is associated with a set of hyperedges $\lambda(v)$ of $\mathcal{H}(I)$ and, hence, to a set of constraints in I, and define (initially) rel_v as the set of all solutions of the CSP restricted to such constraints, possibly projected over the variables in $\chi(v)$. Note that $|rel_v| \leq C^k$, where k is the hypertree width and C denotes the cardinality of the largest constraint relation. Then, in the bottom-up procedure, at each (non-leaf) vertex v, for each child c of v in T, we filter rel_v by keeping only those substitutions θ_v that coincide with some substitution $\theta_c \in rel_c$ on the variables they have in common. At the end, the given instance I admits a solution if, and only if, rel_r is not empty. By this procedure, I can be evaluated in $O((m-1)C^k \log C)$ where m is the number of vertices of T (which is at most the number of variables, in normal-form decompositions). Note that for $k = 1$, the method above coincides with the well-known algorithm by Yannakakis [81] for the evaluation of acyclic instances.

It is natural to ask whether we can achieve fixed-parameter tractability (FPT) [25] by finding a better algorithm which would allow us to get rid of the constant k in the exponent. Unfortunately, this appears to be very unlikely. In fact, the problem can be shown to be fixed-parameter intractable (more precisely, $W[1]$-hard) in the number of constraints or the number of variables as parameters [68]. The same holds for treewidth.

Strategic Games, Databases. Several problems can be reformulated as CSPs, so that structural tractability follows via Theorem 4.2. A noticeable example comes from the theory of strategic games [67]: Pure Nash equilibria are shown to be computable in polynomial time over compactly specified games (see, e.g., [31, 58, 20]) where the players' interaction is encoded in form of a hypergraph with bounded hypertree width [31]—in general the problem if NP-hard.

For other example, we recall that constraint satisfiability is known to be equivalent to a number of problems in database theory [53, 59], e.g., to the problem of conjunctive query containment [59], or to the problem of evaluating *Boolean conjunctive queries* over a relational database [62]. Therefore, cross fertilization among these different research fields was possible and led to major achievements both in the AI and in the DB communities. However, observe that, even if in principle we are talking about equivalent problems, in practice the instances considered in the applications are very different, and thus one cannot simply take any technique from AI and apply it to DB, or vice-versa. Typical CSP instances are indeed characterized by many constraints with relatively small constraint relations, while typical query-answering tasks involve relatively small queries on large (often huge) databases.

[2] To be precise, in order to use the notion of treewidth, it should be specified how a non-binary structure is encoded as a graph. However, it has been shown that the above mentioned result holds in fact for all CSP graph-encodings (currently) described in the literature [47, 36].

4.2 Application to Enumeration Problems

We first focus on the tractability of the problem ECSP of enumerating (possibly projected) solutions. In particular, since even easy instances may have an exponential number of solutions, tractability means here having algorithms that compute solutions *with polynomial delay* (WPD): An algorithm M solves WPD a computation problem P if there is a polynomial $p(\cdot)$ such that, for every instance of P of size n, M discovers if there are no solutions in time $O(p(n))$; otherwise, it outputs all solutions in such a way that a new solution is computed within $O(p(n))$ time from the previous one.

We remark that having efficient algorithms for ECSP is important not only in the (obvious) case we are interested in computing all solutions, but also whenever we are interested in solutions having some specific properties that cannot be expressed by standard constraints. For instance, this happens whenever the desired answers are the solutions of a problem beyond NP, hence not expressible as a (standard) CSP instance. Examples are the *Strategic Companies* problem [13], or problems related with *Conformant Planning* and *Conditional Planning* [71], which are all hard for the second level of the polynomial hierarchy. Indeed, in such cases, one typically starts an enumeration (possibly, anytime) algorithm, and for each computed solution checks whether the additional properties are met or not, which may require a completely different algorithm. In the worst case, the complete enumeration of all solutions may be required. Actually, note that in these cases one is usually interested in a (minimal) subset O of variables sufficient to check the additional properties. Moreover, observe that modeling real-world applications through CSP instances typically requires the use of "auxiliary" variables, whose precise values in the solutions are not relevant for the user, and that are (usually) filtered-out from the output. Therefore, computing all combinations of their values occurring in solutions means wasting time, possibly exponential time. Of course, this is irrelevant for computing just one solution, but it is crucial for the enumeration problem.

It was shown that this problem is tractable for classes of instances having bounded tree-width [12]. Actually, such a tractability result has been extended to the more general tree-projection framework [48], which comprise all purely structural decomposition methods. We next recall a specialization to the hypertree width method.

Theorem 4.3 ([48]). *Let $\mathcal{C}$ be any class of CSPs having bounded hypertree width. Then, for every $I \in \mathcal{C}$ and every set of variables O occurring in I, all the solutions of I projected over O can be enumerated with polynomial delay.*

Analogous positive results hold for the related problem of *counting* the number of solutions. Note that, even for instances with exponentially many solutions, the number of these solutions may well be computed in polynomial time if it is not necessary to actually generate them. Indeed, it has been shown that counting the number of solutions of a CSP is feasible in polynomial time for classes of instances having bounded hypertree width and where the set of output variables O is in fact the full set of variables. The result is also tight on recursively enumerable classes of instances having bounded arity (unless FPT = $W[1]$) [19]. Moreover, in the general case of classes of instances having bounded hypertree width and with arbitrary sets O of desired variables, the problem is still tractable if either the constraint relations or the constraint scopes have a fixed maximum size [70].

Conjunctive Queries. Note that the above enumeration problem is precisely the classical query answering problem for conjunctive queries over relational databases, which are equivalent to SELECT-PROJECT-JOIN queries. Moreover, counting the number of answers is a basic function in most query languages (just think of the `count` operator in SQL). However, despite the very nice computational properties of structural decomposition methods such as treewidth

and hypertree width, they did not have a significant impact on the design of commercial DBMS optimizers, and the interest for these techniques remained only at a theoretical level, until very recently. This is mainly due to two reasons. First, decomposition methods flatten in the hypergraph all the "quantitative" aspects of data, such as selectivity and cardinality of relations, whose knowledge may dramatically speed-up the evaluation time. Second, such methods do not generally take care of the output of the queries, or of aggregate operators. Relevant steps to fill the gap between theory and practice have been made in [77], where the hypertree decomposition method has been extended in order to combine this structural decomposition method with quantitative approaches, and in [29], where methods have been discussed to deal with output variables and aggregate operators. A system prototype implementing these approaches has recently been presented in [28]. The system can be put on top of any existing database management system supporting JDBC technology, by transparently interacting/replacing its standard query optimization module.

4.3 Application to Constraint Optimization

Whenever assignments are associated with some cost because of the semantics of the underlying application domain, computing an arbitrary solution might not be enough. For instance, the crossword puzzle in Example 4.1 may admit more than one solution, and expert solvers may be asked to single out the most difficult ones, such as those solutions that minimize the total number of vowels occurring in the used words. For another example, think of database queries, which are often ranked according to user preferences.

In these cases, one is often interested in the *optimization problem* of computing the solution of minimum cost, whose modeling is accounted for in several variants of the basic CSP framework, such as fuzzy, probabilistic, weighted, lexicographic, penalty, valued, and semiring-based CSPs (see [65, 8], and the references therein). A thorough analysis of the complexity of constraint optimization under structural restrictions has been carried out in [50]. Below, we overview such recent results.

Formal Framework. Let Var be a set of variables and let U be a domain of constants. Let $\succeq$ be a total order over a domain of values $\mathbb{D}$. Then, an *evaluation function* $\mathcal{F}$ over $\mathbb{D}$ and $\succeq$ is a tuple $\langle w, \oplus \rangle$ with $w : Var \times U \mapsto \mathbb{D}$ and where $\oplus$ is a commutative, associative, and closed binary operator with an identity element over $\mathbb{D}$.

For a substitution $\theta \neq \emptyset$, $\mathcal{F}(\theta)$ is the value $\bigoplus_{X/u \in \theta} w(X, u)$; and, conventionally, $\mathcal{F}(\emptyset)$ is the identity element (w.r.t. $\oplus$). The evaluation function $\mathcal{F} = \langle w, \oplus \rangle$ is *monotone* if $\mathcal{F}(\theta) \succeq \mathcal{F}(\theta')$ implies that $\mathcal{F}(\theta) \oplus \mathcal{F}(\theta'') \succeq \mathcal{F}(\theta') \oplus \mathcal{F}(\theta'')$, for each triple of substitution $\theta, \theta', \theta''$.

Let $L = [\mathcal{F}_1, ..., \mathcal{F}_m]$ be a list of evaluation functions, where each $\mathcal{F}_i$ is defined over a domain $\mathbb{D}_i$ and a total order $\succeq_i$, $\forall i \in \{1, ..., m\}$. Then, for any substitution θ, $L(\theta)$ denotes the vector of values $(\mathcal{F}_1(\theta), ..., \mathcal{F}_m(\theta)) \in \mathbb{D}_1 \times \cdots \times \mathbb{D}_m$. To compare elements of $\mathbb{D}_1 \times \cdots \times \mathbb{D}_m$, we consider the lexicographical total order $\succeq_{lex}$, inducing a hierarchy over the preference relations in each domain. Let $\mathbf{x} = (x_1, ..., x_m)$ and $\mathbf{y} = (y_1, ..., y_m)$ be two vectors with $x_i, y_i \in \mathbb{D}_i$, for each $i \in \{1, ..., m\}$. Then, as usual, $\mathbf{x} \succeq_{lex} \mathbf{y}$, if either $\mathbf{x} = \mathbf{y}$, or there is an index $i \in \{1, ..., m\}$ such that $x_i \succ_i y_i$ and $x_j = y_j$ holds, for each $j \in \{1, ..., i-1\}$.

Let $L = [\mathcal{F}_1, ..., \mathcal{F}_m]$ be a list of evaluation functions. Then, we define $\succeq_L$ as the binary relation such that, for each pair θ_1 and θ_2 of substitutions, $\theta_1 \succeq_L \theta_2$ if, and only if, $L(\theta_1) \succeq_{lex} L(\theta_2)$. Note that $\succeq_L$ is a preorder, which might be not antisymmetric, as $L(\theta_1) = L(\theta_2)$ does not imply that $\theta_1 = \theta_2$. As the ordering might even not be a partial order at all, it is natural to exploit linearization techniques, as discussed in [2].

Let $\succeq_U$ be an arbitrary total order defined over U. Let $\ell = [X_1, ..., X_n]$ be a list including all the variables in Var, hereinafter called *linearization*. Then, we define $\succeq_L^\ell$ as the binary relation

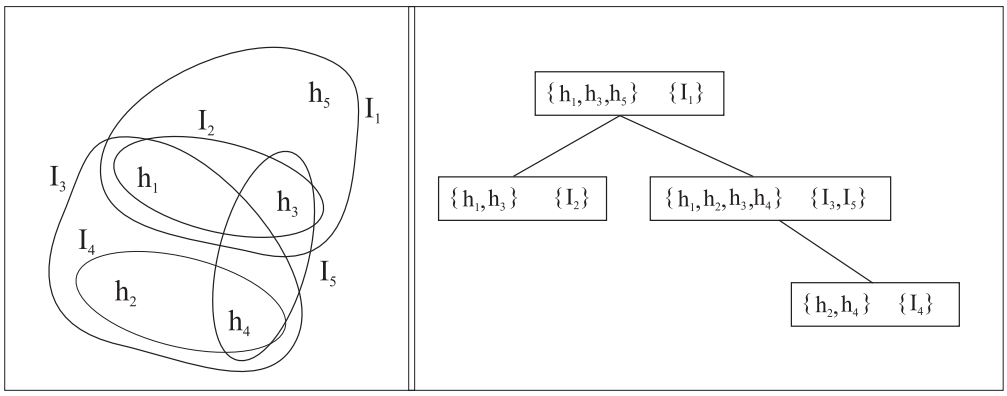

Figure 10: A Dual Hypergraph with a with-2 hypertree decomposition.

such that, for each pair θ_1 and θ_2 of substitutions, $\theta_1 \succeq_L^\ell \theta_2$ if, and only if, (i) $\theta_1 = \theta_2$, or (ii) $\theta_1 \succeq_L \theta_2$ and $\theta_2 \not\succeq_L \theta_1$, or (iii) $\theta_1 \succeq_L \theta_2$, $\theta_2 \succeq_L \theta_1$, and there is a variable X_i such that $\theta_1(X_i) \succ_U \theta_2(X_i)$, and $\theta_1(X_j) = \theta_2(X_j)$, for each $j \in \{1, ..., i-1\}$. Note that $\succeq_L^\ell$ is a total order, where ties in $\succeq_L$ are resolved according to ℓ and the total order $\succeq_U$ over U. In fact, $\succeq_L^\ell$ is a refinement of $\succeq_L$.

Structural Tractability. An instance $I = (Var, U, \mathcal{C})$ of a constraint satisfaction problem equipped with a list L of evaluation functions is called a *constraint optimization problem*, and is denoted by I_L. A problem that naturally arises with evaluation functions is the MIN(I_L, ℓ) problem of computing the solution θ to I such that there is no solution θ' with $\theta \succ_L^\ell \theta'$ (or recognizing that no solution exists at all). In case of arbitrary evaluation functions, results are bad news about the tractability of MIN, even on classes of acyclic instances.

Theorem 4.4 ([50]). MIN *is* NP*-hard for any linearization, even on classes of instances* I_L *where* $\mathcal{H}(I)$ *is acyclic.*

However, on monotone functions to be optimized, bounded hypertree width is again a key to ensure tractability. The complexity analysis follows here the usual simple approach of counting 1 each mathematical operation, hence in principle one may compute in polynomial time (operations) values whose size is exponential w.r.t. the input size. We thus explicitly care about the size of values computed during the execution of algorithms, and look for output polynomial-space algorithms.

Theorem 4.5 ([50]). *On classes of constraint optimization problems* I_L *where the hypertree width of* $\mathcal{H}(I)$ *is bounded by some fixed natural number and where* L *is a list of monotone evaluation functions,* MIN *is feasible in polynomial time and output-polynomial space (for any linearization).*

The result is established by modifying the bottom-up procedure illustrated in the section above for constraint satisfiability. The main difference is that, at each (non-leaf) vertex v, for each child c of v in T, one filters rel_v by keeping only the "best" substitutions θ_v, i.e., those on which the solution of minimum cost can be achieved for the CSP obtained by considering the constraints in the subtree rooted at v.

Combinatorial Auctions. The above solution algorithm may be viewed as a generalization to lexicographic optimization of an algorithm discussed in [30] for *combinatorial auctions*, where there is just one function to be optimized.

Combinatorial auctions are well-known mechanisms for resource and task allocation where bidders are allowed to simultaneously bid on combinations of items. Such mechanisms model

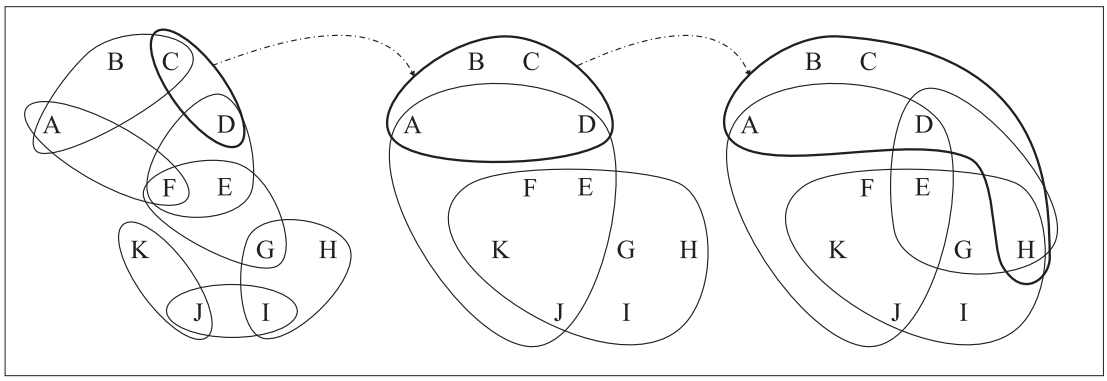

Figure 11: A tree projection for the hypergraph $\mathcal{H}_1$ in Figure 4 (on the left) w.r.t. the resource hypergraph reported on the right.

very well those situations where bidders' valuations of bundles of items are not equal to the sum of their valuations of individual items.

Formally, a *combinatorial auction* is a pair $\langle \mathcal{I}, \mathcal{B} \rangle$, where $\mathcal{I} = \{I_1, ..., I_m\}$ is the set of items the auctioneer has to sell, and $\mathcal{B} = \{B_1, ..., B_n\}$ is the set of bids from the buyers interested in the items in $\mathcal{I}$. Each bid B_i has the form $\langle item(B_i), pay(B_i) \rangle$, where $pay(B_i)$ is a rational number denoting the price a buyer offers for the items in $item(B_i) \subseteq \mathcal{I}$. An outcome for $\langle \mathcal{I}, \mathcal{B} \rangle$ is a subset $\mathbf{b}$ of $\mathcal{B}$ such that $item(B_i) \cap item(B_j) = \emptyset$, for each pair B_i and B_j of bids in $\mathbf{b}$ with $i \neq j$. Bidder interaction in a combinatorial auction $\langle \mathcal{I}, \mathcal{B} \rangle$ can be represented via its associated *dual hypergraph* [30] $\overline{\mathcal{H}}(\langle \mathcal{I}, \mathcal{B} \rangle)$, whose nodes are the various bids in the auction and hyperedges represent items, that is, each item $\mathrm{I} \in \mathcal{I}$ is associated with a hyperedge consisting of the set of bids that contain I.

For example, the hypergraph in Figure 10, on the left, encodes an auction over items $\{\mathrm{I}_1, ..., \mathrm{I}_5\}$ and where bids are on the following bundles of items: $h_1 : \{\mathrm{I}_1\}$, $h_2 : \{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3\}$, $h_3 : \{\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_5\}$, $h_4 : \{\mathrm{I}_3, \mathrm{I}_4\}$, and $h_5 : \{\mathrm{I}_3, \mathrm{I}_4, \mathrm{I}_5\}$. Note that this hypergraph is not acyclic and its hypertree width 2, as it is witnessed by the decomposition reported in Figure 10.

A crucial problem for combinatorial auctions is the *winner determination problem* of determining the outcome $\mathbf{b}^*$ that maximizes the sum of the accepted bid prices (i.e., $\sum_{B_i \in \mathbf{b}^*} pay(B_i)$) over all the possible outcomes. The problem is in general NP-hard [73], while it is tractable on classes of auctions with bounded hypertree-width dual hypergraphs [30].

5 Beyond (Hyper)tree Decompositions

All the known so-called purely-structural decomposition methods, where decompositions are only based on the (hyper)graph structure $\mathcal{H}$ of the given instance, are in fact specializations of the general and abstract framework of *tree projections* [54]. In this view, the goal is to cover the hypergraph $\mathcal{H}$ via an acyclic hypergraph (the tree projection), in a way that each hyperedge is contained in some hyperedge of another given "resource" hypergraph $\mathcal{H}'$ (also known as the acyclic hypergraph sandwich problem). See Figure 11, for an example of tree projection.

The existence of a tree projection is often a key to establish tractability results for decision problems [54] and for enumeration ones [48]. Moreover, it guarantees that interesting consistency properties [49] and game-theoretic characterizations [46] hold. According to this unifying view, differences among the various (purely) structural decomposition methods just come in the way the resource hypergraph $\mathcal{H}'$ is defined. In particular, for a fixed natural number k, the

treewidth method is obtained by considering as available hyperedges in the resource hypergraph $\mathcal{H}'$ all combinations of k nodes of $\mathcal{H}$; while the generalized hypertree-width method is obtained by considering all combinations of k hyperedges of $\mathcal{H}$. However, note that the notion of tree projection is more general then both treewidth and hypertree width, because the hyperedges of the "resource" hypergraph $\mathcal{H}'$ may model arbitrary subproblems of the given instance whose solutions are easy to compute, or already available from previous computations (for instance, materialized views while answering database queries).

As an example of a different set of useful "resources" (subproblems), we mention a powerful extension of hypertree decompositions, called *fractional* hypertree decompositions [52]. According to this notion, the resources are the width-k fractional covers of hypergraph vertices, instead of the integral covers that characterize hypertree decompositions (where each hyperedge counts 1). It turns out that fractional hypertree-width is strictly more general than the hypertree width, as there exist classes of hypergraphs having bounded fractional hypertree width and unbounded (generalized) hypertree width.

For completeness, we recall the most powerful method known at this time, whose width is determined by suitably covering tree-decomposition bags by using submodular functions. The resulting notion, called submodular width [63], is in its turn strictly more general than fractional hypertree width. However, unlike the previously mentioned methods that guarantee polynomial-time tractability for bounded-width classes, this technique only guarantees that classes of instances having bounded submodular-width are fixed-parameter tractable (and in [63] a tight result is proved for this kind of tractability, see below).

6 Tractability Frontiers (for CSPs)

Constraint satisfaction is often formalized as a homomorphism problem that takes as input two finite relational structures $\mathcal{A}$ (modeling variables and scopes of the constraints) and $\mathcal{B}$ (modeling the relations associated with constraints), and asks whether there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$ [59]. We consider problems where $\mathcal{A}$ must be taken from some suitably defined class $\mathbf{A}$ of structures, while $\mathcal{B}$ is any arbitrary structure from the class "$-$" of all finite structures, shortly denoted as $\mathrm{CSP}(\mathbf{A}, -)$. Note that we face the so-called *uniform* homomorphism problem, where both structures are part of the input, i.e., nothing is fixed. For completeness we recall that, instead, in the typical *non-uniform* problem $\mathrm{CSP}(-, \mathcal{B})$, $\mathcal{B}$ is a fixed structure (thus, any instance of such a problem just consists of some left-hand structure $\mathcal{A}$).

6.1 Decision Problem

Several decomposition methods have been proposed in the last few years, with the aim of discovering further islands of tractability and, ultimately, of charting the tractability frontier for constraint satisfaction. In the bounded-arity case, research has already achieved this ambitious goal.

Theorem 6.1 ([51])**.** *Assume* FPT $\neq W[1]$. *Then, for any recursively enumerable class of bounded-arity structures* $\mathbf{A}$*, CSP*$(\mathbf{A}, -)$ *is solvable in polynomial time* if, and only if, *the cores of the structures in* $\mathbf{A}$ *have bounded (hyper)treewidth.*[3]

Note that the latter condition may be equivalently stated as follows: for every $\mathcal{A} \in \mathbf{A}$ there is some $\mathcal{A}'$ homomorphically equivalent to $\mathcal{A}$ and such that its treewidth is below the required fixed

[3]Recall that, for classes of structures having bounded arity, bounded hypertree width entails bounded treewidth (the converse is always true), so that the two notions are interchangeable for these classes.

threshold. For short, we say that such a class $\mathbf{A}$ has bounded treewidth modulo homomorphic equivalence.

Things with unbounded-arity classes are, instead, not that clear. Generalized hypertree-width seems the natural counterpart of the tree decomposition method over unbounded-arity classes, and in fact $\mathrm{CSP}(\mathbf{A}, -)$ is solvable in polynomial time *if* $\mathbf{A}$ has bounded hypertree-width modulo homomorphic equivalence [15]. However, it is known that generalized hypertree-width does not characterize all classes of structures where $\mathrm{CSP}(\mathbf{A}, -)$ is solvable in polynomial time. Indeed, there are classes of structures having unbounded hypertree width that are tractable, because they have bounded fractional hypertree-width [52]. It seems that tighter results may be obtained by moving from polynomial-time tractability to fixed-parameter tractability (FPT). For any class of hypergraphs $\mathcal{H}$, let $\mathrm{CSP}(\mathcal{H})$ denote the class of all homomorphism problem instances $(\mathcal{A}, \mathcal{B})$ where the hypergraph associated with $\mathcal{A}$ belongs to $\mathcal{H}$. Then, under some reasonable technical assumptions, $\mathrm{CSP}(\mathcal{H})$ is FPT *if, and only if,* hypergraphs in $\mathcal{H}$ have bounded *submodular* width [63].

It is worthwhile noting that the above mentioned tractability results for classes of instances defined modulo homomorphically equivalence are actually tractability results for the *promise* version of the problem. In fact, unless P = NP, there is no polynomial-time algorithm that may check whether a given instance $\mathcal{A}$ actually belongs to such a class $\mathbf{A}$. In particular, it has been observed by different authors [76, 12] that there are classes of instances having bounded treewidth modulo homomorphically equivalence for which answers computable in polynomial time cannot be trusted. That is, unless P = NP, there is no efficient way to distinguish whether a "yes" answer means that there exists a solution of the problem, or that $\mathcal{A} \notin \mathbf{A}$.

In fact, the tractability frontier for the so-called *search problem* of computing just one solution (whose correctness may be easily checked) is an interesting open problem, which is somehow related to the frontier of the problem of enumerating homomorphisms, discussed next (for more on the relationships between these two problems, see [12]). For completeness, we recall that things are different for the case of non-uniform CSPs, where instead the tractability of the decision problem always entails the tractability of the search problem [16].

6.2 Enumeration Problems

Define formally an ECSP instance to be a triple $(\mathcal{A}, \mathcal{B}, O)$, for which we have to compute all solutions (homomorphisms) projected to a set of desired output variables O, denoted by $\mathcal{A}^{\mathcal{B}}[O]$.

In order to identify the (structural) tractability frontier, there are two choices to model the presence of output variables. The first possibility is that output variables are part of the (left-hand) structure. For instance, in [51] an additional "virtual" constraint covering together all possible output variables is added to the input structure. For bounded-arity recursively-enumerable classes of this form, it turns out that the enumeration problem is tractable for a class of instances if, and only if, the class has bounded treewidth modulo homomorphic equivalence. Note that the only possibility to meet this bounded treewidth requirement is having a fixed number of output variables in the class, otherwise the additional virtual constraint will have unbounded arity. Therefore, according to this approach, only instances with a polynomial number of (projected) solutions may be dealt with.

This limitation does not occur in a more recent modeling choice of output-aware structures, where possible output variables are described as those variables X having a domain constraint $dom(X)$, that is, a distinguished unary constraint specifying the domain of this variable. Such variables are said domain restricted. In fact, this choice reflects the classical approach in constraint satisfaction systems, where variables are typically associated with domains, which are heavily exploited by constraint propagation algorithms. Interestingly, for bounded-arity

recursively-enumerable classes we get the same kind of statement as above, but now allowing an unbounded number of output variables, and hence the computation of exponentially many solutions, in general.

Theorem 6.2 ([48]). *Let* $\mathbf{A}$ *be a class of (left-hand) structures having bounded hypertree-width modulo homomorphic equivalence. Then,* $\mathbf{A}$ *has a tractable enumeration problem, i.e., for every* $\mathcal{A} \in \mathbf{A}$*, for every right-hand structure* $\mathcal{B}$*, and for every set* O *of domain restricted variables, the* ECSP *instance* $(\mathcal{A}, \mathcal{B}, O)$ *is solvable WPD.*

Moreover, let $\mathbf{A}$ *be any recursively-enumerable class of structures of bounded arity having a tractable enumeration problem. Then,* $\mathbf{A}$ *has bounded (hyper)tree-width modulo homomorphic equivalence (unless* FPT $= W[1]$*).*

The second possibility to deal with output variables is to leave them completely arbitrary, hence looking for a stronger form of tractability that should hold for any required set O of desired output variables. Intuitively, in this case it is not possible to have tractable instances with intractable substructures. Observe that this latter natural property does not hold if we do not consider projected solutions but we rather look for the enumeration of all (full) solutions, as shown in [12].

The tractability frontier for this case is still an open problem (see, e.g., [51] for a statement of this problem in the context of conjunctive database queries). However, there is a partial answer for the case of bounded-arity classes of structures closed under taking minors. It turns out that, for these classes, the enumeration problem is tractable if, and only if, the whole structures (and not just their cores) have bounded (hyper)treewidth.

Theorem 6.3 ([48]). *Let* $\mathbf{A}$ *be any recursively-enumerable class of (left-hand) structures of bounded arity closed under taking minors. Then, for every* $\mathcal{A} \in \mathbf{A}$*, for every right-hand structure* $\mathcal{B}$*, and for every set* O *of variables, the* ECSP *instance* $(\mathcal{A}, \mathcal{B}, O)$ *is solvable WPD if, and only if,* $\mathbf{A}$ *has bounded (hyper)tree-width (unless* FPT $= W[1]$*).*

A final important observation is that the former results based on homomorphic equivalence, which generalize the results on decision problems to the enumeration problem, suffer of the same practical problem of giving as their output solutions that cannot be trusted, in general (because of the promise). Instead, for the above classes of structures having a bounded-width hypertree decomposition, we are able to compute *certified solutions* with polynomial delay [48], which is what we need in practical applications (think, e.g., of database query answering).

6.3 Optimization Problems

The same kind of tight tractability results, based on the existence of a hypertree decomposition of the whole structure, have been found for constraint optimization problems. In particular, besides Min, we recall the problem of enumerating the best K solutions (Top-K), and the problem of computing the next solution following one that is given at hand (Next). For these problems, good news have been found not only for *monotone* functions, but also for those (possibly) *non-monotone* functions, called *smooth evaluation functions*, which manipulate "small" (in fact, polynomially-bounded) values.

Monotone functions have already been discussed in Section 4.3, where we have noticed that the winner determination problem in combinatorial auction can be formulated as a constraint optimization problem over this kind of functions.

For a simple example of a smooth evaluation function, consider a function which counts the number of variables that are mapped to some domain values. This is useful whenever we would like to minimize the variables mapped to some "undesirable" values. Observe that the possible

Problem	**No Restriction**	**Monotone**	**Smooth**
MIN	NP-hard	in P	in P
NEXT	NP-hard	NP-hard	in P
TOP-K	NP-hard	WPD	WPD

Figure 12: Summary of complexity results for bounded hypertree-width instances.

output values of this function are polynomially bounded in the size of the input, because they range from 0 to the number of variables. In fact, it is a smooth *and* monotone function, as it is often the case for functions based on counting.

Finally, as an example of non-monotone smooth evaluation-function, assume for the sake of simplicity to have one "undesirable" value 'a', and consider the following function: take the product of weights associated with variable assignments, where every variable mapped to 'a' is weighed -1, and all the others get 1. That is, we prefer those solutions with an odd number of variables mapped to the "undesirable" value. In general, functions involving multiplications are non-monotone, yet they are smooth if their output values are polynomially bounded (w.r.t. the input size).

Figure 12 provides a picture of the complexity of such optimization problems over classes of instances having bounded hypertree width—there recall from Section 4.2 that WPD is the class of all problems that can be solved with polynomial delay. All tractability results are tight on recursively enumerable classes of instances having bounded arity (unless FPT = $W[1]$) [50].

7 Conclusion

This chapter describes techniques where graphical structures are profitably exploited to solve efficiently a large number of problem instances. We mentioned only a few possible applications, but we believe the list being not exhaustive at all, because such basic (hyper)graph theoretic notions may be actually useful in many different fields, most often outside authors' areas of research. Just to give a rough idea of the wide spectrum of these fields, we recall the connections between the treewidth and the resolution width in propositional proof systems [4] or the applications to natural sciences (e.g., some solutions to structure-sequence alignment problems in bioinformatics exploit the fact that many involved structure graphs, such as almost all existing RNA pseudoknots, have small treewidth [79, 56]).

In fact, it is very common to observe that many instances of real-life problems exhibit a kind of sparse tree-like structure in the large, but have a very dense highly-cyclic structure in local connections. Therefore, it is natural to imagine that in such applications the ability to combine different techniques, possibly based on different notions of "good" structure, will become very important in the near future. For instance, by using the tree projection framework described in Section 5, one may arrange in a tree-like structure a number of *dense* subproblems whose solutions are instead computed easily because of completely different properties.

References

[1] I. Adler, G. Gottlob, and M. Grohe. Hypertree-Width and Related Hypergraph Invariants. *Journal of Combinatorics*, 28(8), pp. 2167–2181, 2007.

[2] R.I. Brafman, F. Rossi, D. Salvagnin, K.B. Venable, and T. Walsh. Finding the Next

Solution in Constraint- and Preference-Based Knowledge Representation Formalisms. In *Proc. of KR'10*, pp. 425–433, 2010.

[3] A. Atserias, A.A. Bulatov, and V. Dalmau. On the Power of k-Consistency, In *Proc. of ICALP'07*, pp. 279–290, 2007.

[4] A. Atserias and V. Dalmau. A combinatorial characterization of resolution width, *Journal of Computer and System Sciences*, 74(3), pp. 323–334, 2008.

[5] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2), pp. 308-340, 1991.

[6] E. Ben-Sasson and A. Wigderson. Short proofs are narrow - resolution made simple. *Journal of the ACM*, 48(2), pp. 149–169, 2001.

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

[8] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier. Semiring-Based CSPs and Valued CSPs: Frameworks, Properties,and Comparison. *Constraints*, 4(3), pp. 199–240, 1999.

[9] H.L. Bodlaender and F.V. Fomin. A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. SIAM Journal on Computing, 25(6), pp. 1305–1317, 1996.

[10] H.L. Bodlaender, F.V. Fomin, A.M.C.A. Koster, D. Kratsch, and D.M. Thilikos. On exact algorithms for treewidth. In *Proc. of ESA'06*, pp. 672-683, 2006.

[11] H.L. Bodlaender and A.M.C.A. Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3), pp. 259-275, 2010.

[12] A. Bulatov, V. Dalmau, M. Grohe, and D. Marx. Enumerating Homomorphism. *Journal of Computer and System Sciences*, 78(2), pp. 638–650, 2012.

[13] M. Cadoli, T. Eiter, and G. Gottlob. Default logic as a query language. *IEEE Transactions on Knowledge and Data Engineering*, 9(3), pp. 448–463, 1997.

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

[15] H. Chen and V. Dalmau. Beyond Hypertree Width: Decomposition Methods Without Decompositions. In *Proc. of CP'05*, pp. 167–181, 2005.

[16] D.A. Cohen. Tractable Decision for a Constraint Language Implies Tractable Search. *Constraints*, 9(3), 219–229, 2004.

[17] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3), pp. 233-248, 2003.

[18] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, The MIT Press, Cambridge, MA, USA, pp. 193242, 1990.

[19] V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3), pp. 315–323, 2004.

[20] C. Daskalakis and C.H. Papadimitriou. Computing pure nash equilibria in graphical games via markov random fields. In *Proc. of ACM EC'06*, pp. 91–99, 2006.

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

[22] R. Dechter. Bucket Elimination: A Unifying Framework for Processing Hard and Soft Constraints. In *Proc. of UAI'96*, pp. 211–219, 1996.

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

[24] R. Dechter and J. Pearl. Tree-clustering schemes for constraint-processing. In *Proc. of AAAI*, pp. 150-154, 1988.

[25] R. Downey and M. Fellows. *Parameterized Complexity*. Springer, 1999.

[26] M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proc. of FOCS'10*, pp. 143-152, 2010.

[27] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of ACM*, 30(3), pp. 514–550, 1983.

[28] L. Ghionna, G. Greco, and F. Scarcello. H-DB: A Hybrid Quantitative-Structural SQL Optimizer. In *Proc. of CIKM'11*, 2011.

[29] L. Ghionna, L.Granata, G. Greco, and F. Scarcello. Hypertree Decompositions for Query Optimization. In *Proc. of ICDE '07*, pp. 36–45, 2007.

[30] G. Gottlob and G. Greco. On the complexity of combinatorial auctions: structured item graphs and hypertree decomposition. In *Proc. EC'07*, pp. 152–161, 2007.

[31] G. Gottlob, G. Greco, and F. Scarcello. Pure Nash Equilibria: Hard and Easy Games. *Journal of Artificial Intelligence Research*, 24, pp. 357–406, 2005.

[32] G. Gottlob, G. Greco, Z. Miklós, F. Scarcello, T. Schwentick. Tree Projections: Game Characterization and Computational Aspects. *Graph Theory, Computational Intelligence and Thought*, pp. 217–226, 2009.

[33] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. Hypertree Decompositions: Structure, Algorithms, and Applications. In *Proc. of WG'05*, 2005.

[34] G. Gottlob and S.T. Lee. A logical approach to multicut problems. *Information Processing Letters*, 103(4), pp. 136–141, 2007.

[35] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *Proc. of DEXA'99*, pp. 1–15, 1999.

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

[37] G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries. *Journal of the ACM*, 43(3), pp. 431–494, 2001.

[38] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions: A Survey. In *Proc. of MFCS'01*, pp. 37–57, 2001.

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

[40] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. of Computer and System Sciences*, 66(4), pp. 775–808, 2003.

[41] G. Gottlob, Z. Miklós, and T. Schwentick. Generalized hypertree decompositions: np-hardness and tractable variants. *Journal of the ACM*, 56(6), 2009.

[42] G. Gottlob, R. Pichler, and F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artificial Intelligence*, 174(1), pp. 105–132, 2010.

[43] G. Gottlob and M. Samer. A Backtracking-Based Algorithm for Computing Hypertree-Decompositions. *Journal of Experimental Algorithmics*, 13, Article 1, 2009.

[44] E. Grädel, P.G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M.Y. Vardi, Y. Venema, S. Weinstein. *Finite Model Theory and Its Applications*. Springer, 2007.

[45] G. Greco, E. Malizia, F. Scarcello, and L. Palopoli. On the complexity of core, kernel, and bargaining set. *Artificial Intelligence*, 175(12-13), pp. 1877–1910, 2011.

[46] G. Greco and F. Scarcello. Tree Projections: Hypergraph Games and Minimality. In *Proc. of ICALP'08*, pp. 736–747, 2008.

[47] G. Greco and F. Scarcello. On the power of structural decompositions of graph-based representations of constraint problems. *Artificial Intelligence*, 174(5-6), pp. 382–409, 2010.

[48] G. Greco and F. Scarcello. Structural Tractability of Enumerating CSP Solutions. *Constraints* 18(1), pp. 38–74, 2013.

[49] G. Greco and F. Scarcello. The Power of Tree Projections: Local Consistency, Greedy Algorithms, and Larger Islands of Tractability. In *Proc. of PODS'10*, pp. 327–338, 2010. A full version is currently available at CoRR abs/1212.2314 (2012).

[50] G. Greco and F. Scarcello. Structural Tractability of Constraint Optimization. In *Proc. of CP'11*, pp. 340–355, 2011.

[51] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1), 2007.

[52] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proc. of SODA '06*, pp. 289–298, 2006.

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

[54] N. Goodman and O. Shmueli. The tree projection theorem and relational query processing. *J. of Computer and System Sciences*, 29(3), pp. 767–786, 1984.

[55] P. Harvey and A. Ghose. Reducing Redundancy in the Hypertree Decomposition Scheme. In *Proc. of ICTAI'03*, pp. 474–481, 2003.

[56] Z. Huang, Y. Wu, J. Robertson, L. Feng, R.L. Malmberg, and L. Cai. Fast and accurate search for non-coding RNA pseudoknot structures in genomes. *Bioinformatics*, 24(20), pp. 2281–2287, 2008.

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

[58] M. Kearns, M. Littman, and S. Singh, S. Graphical models for game theory. In *Proc. of UAI'01*, pp. 253–260, 2001.

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

[60] T. Korimort. Constraint Satisfaction Problems – Heuristic Decomposition. PhD thesis, Vienna University of Technology, April 2003.

[61] D. Lehmann, R. Müller, and T. Sandholm. The Winner Determination Problem. In P. Cramton, Y. Shoham, and R. Steinberg, editors, *Combinatorial Auctions*. MIT Press, 2006.

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

[63] D. Marx. Tractable Hypergraph Properties for Constraint Satisfaction and Conjunctive Queries. In *Proc. of STOC'10*, pp. 735–744, 2010.

[64] B. McMahan. Bucket elimination and hypertree decompositions. Implementation report, Institute of Information Systems (DBAI), TU Vienna, 2004.

[65] P. Meseguer, F. Rossi and T. Schiex. Soft Constraints. *Handbook of Constraint Programming*, Elsevier, 2006.

[66] N. Musliu. Tabu Search for Generalized Hypertree Decompositions. In *Proc. of MIC'07*, 2007.

[67] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2), pp. 286–295, 1951.

[68] C.H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and Systems Sciences*, 58(3), pp. 407427, 1999.

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

[70] R. Pichler and S. Skritek. Tractable Counting of the Answers to Conjunctive Queries. *Journal of Computer and System Sciences*, 79(6), pp. 984–1001, 2013.

[71] J. Rintanen. Planning and SAT. In *Handbook of Satisfiability*, Frontiers in Artificial Intelligence and Applications, 185, pp. 483–504, IOS Press, 2009.

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

[73] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44:1131–1147, 1998.

[74] M. Samer. Hypertree-decomposition via Branch-decomposition. In *Proc. of IJCAI'05*, pp. 1535–1536, 2005.

[75] F. Scarcello. The Hypertree Decompositions HomePage, since 2002. `http://www.dimes.unical.it/scarcello/Hypertrees/` See also the WEB page `http://www.dbai.tuwien.ac.at/proj/hypertree/` mantained by N. Musliu.

[76] F. Scarcello, G. Gottlob, and G. Greco. Uniform Constraint Satisfaction Problems and Database Theory. In *Complexity of Constraints*, LNCS 5250, pp. 156–195, Springer-Verlag, 2008.

[77] F. Scarcello, G. Greco, and N. Leone. Weighted Hypertree Decompositions and Optimal Query Plans. *Journal of Computer and System Sciences*, pp. 475–506, 2007.

[78] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58, pp. 22-33, 1993.

[79] Y. Song, C. Liu, X. Huang, R.L. Malmberg, Y. Xu, and L. Cai. Efficient Parameterized Algorithms for Biopolymer Structure-Sequence Alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4), pp. 423–432, 2006.

[80] S. Subbarayan and H. Reif Andersen. Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs. In *Proc. of IJCAI'07*, pp. 180–185, 2007.

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