PART 1

GRAPHICAL STRUCTURE

1

Treewidth and Hypertree Width

Georg Gottlob, Gianluigi Greco, Francesco Scarcello

This 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.

Many NP-hard problems in different areas such as AI [42], Database Systems [6, 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

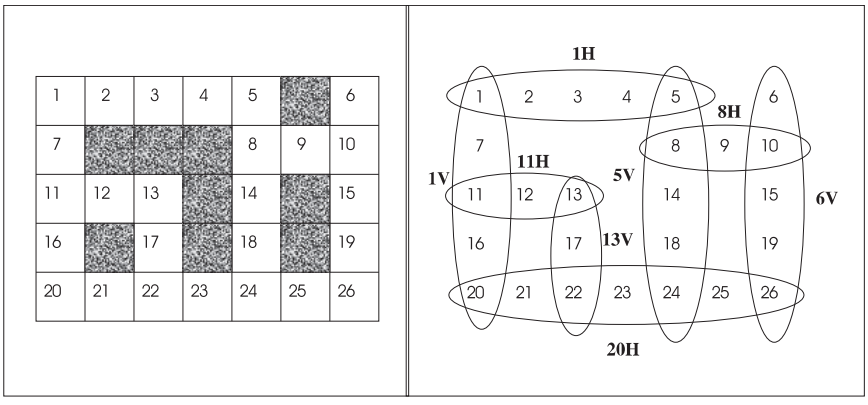

Figure 1.1 A crossword puzzle and its representation as a hypergraph $\mathcal{H}_0$.

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 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 1.1, 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 1.1. 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 1.3. 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 1.4.

The rest of this chapter is organized as follows. In Section 1.1 we review the notion of *treewidth*, and in Section 1.2 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 applications of such decomposition methods are illustrated. In particular, Section 1.3 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 1.5.

1.1 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 1.1 ([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 $p \in V$

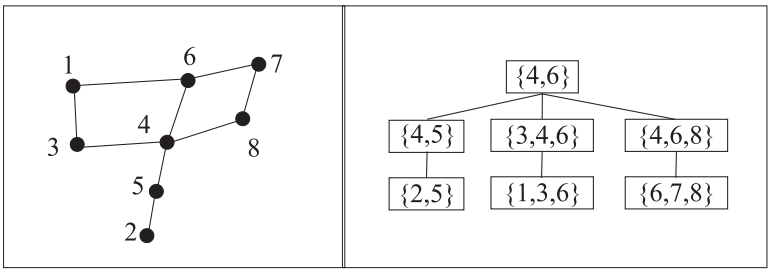

Figure 1.2 A graph G_0 and a tree decomposition for it.

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 1.2 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 [8], and was recently shown to be achievable in logarithmic space [26]. Note that the multiplicative constant factor of Bodlaender's linear algorithm [8] 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, [9, 10] and the references therein.

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

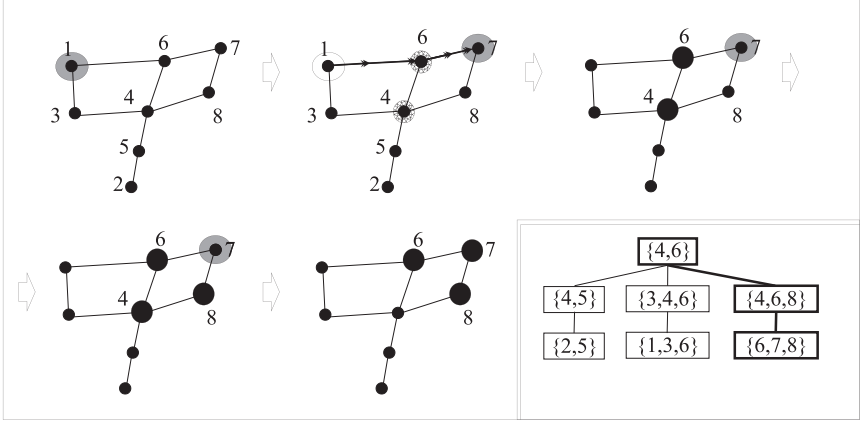

Figure 1.3 The robber and cop game played on the graph G_0 of Figure 1.2.

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 1.2 Consider the robber and cops game played on the graph G_0 of Figure 1.2, and the moves illustrated in Figure 1.3: 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 1.2 (and replicated in Figure 1.3, 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. ◁

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 simply and elegantly 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*.

1.1.1 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 1.3 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. ◁

The next theorem relates treewidth to MSO.

Theorem 1.4 *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] *(with respect to $||\mathcal{A}||$).*

From the above theorem and Example 1.3, 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.

1.1.2 Applications to Optimization Problems

An important generalization of MSO formulae to *optimization* problems was presented by [2].

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 1.5 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. ◁

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 1.6 (simplified from [2]) *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 (with respect to $||\langle \mathcal{A}, w \rangle||$).*

From the above theorem and Example 1.5, 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.

1.2 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