Width Parameters Beyond Tree-width and Their Applications

Petr Hliněný*
Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Rep.
E-mail: hlineny@fi.muni.cz
and
VŠB – Technical University of Ostrava, Czech Rep.

Sang-il Oum
School of Mathematics, Georgia Institute of Technology,
Atlanta GA 30332-0160, USA.
E-mail: sangil@math.gatech.edu

Detlef Seese
Institute AIFB, University Karlsruhe (TH),
D-76128 Karlsruhe, Germany.
E-mail: seese@aifb.uni-karlsruhe.de

Georg Gottlob
Oxford University Computing Laboratory,
Parks Road, Oxford OX1 3QD, England.
E-mail: ggottlob@gmail.com
and
Vienna University of Technology, Austria.

May 18, 2006

Abstract

Besides the successful concept of tree-width (see [H. Bodlaender, A. Koster: *Combinatorial optimisation on graphs of bounded treewidth*, ****** this survey volume ******, 14 p.]) in the past years, many concepts and parameters measuring a similarity of structures to trees, or how a structure distinguishes from a tree, have been born and studied. These concepts and parameters proved to be useful tools for many applications, especially in the design of efficient algorithms. We present a novel view of contemporary developments of these "width" parameters in combinatorial structures that, besides traditional tree-width and derived dynamic programming schemes, leads to other usable parameters like branch-width,

*Corresponding author, hlineny@fi.muni.cz.

rank-width (clique-width), or hypertree-width. Our article demonstrates how "width" parameters of graphs and generalized structures (like matroids or hypergraphs), on an abstract level, can be used to improve the design of parameterized algorithms and the structural analysis in other applications.

1 Introduction and Overview

We begin with few words about organization of our text: After this introduction covering motivation, key examples and bit of notation, follows Chapter 2 on some basic tools from logic which are the key to some later applications. Then follow three relatively independent chapters surveying interesting and currently active areas of structural width parameters in computer science. Chapter 3 is devoted to the notion of branch-width and several of its generalizations. Here we discuss connectivity functions, branch-width concepts for graphs and matroids and parameterized algorithms. Chapter 4 is devoted to rank-width and clique-width and many related concepts. Hypergraphs are then discussed in Chapter 5 on hypertree-width and related concepts, which are of special interest for databases. (A typical problem handled there is the Constraint Satisfaction Problem.) The last Chapter 6 adds some concluding remarks and directions for further study and research. All the concepts in this article are strong generalizations of the tree-width concept which is covered in [BK06].

To make this paper accessible to majority of computer scientists, each Chapter gradually moves from (quite) informal description of the content and applications to more formal (and mathematically involved) concepts. Except assuming basic knowledge of graphs and parametrized complexity topics which are handled in [***** this survey volume *****] (see also [DF99, FG06]), we try our best to make the chapters of the text self-contained, so that an interested reader can pick up his favorite topic at an accessible level without need for a long study.

Since graphs are the basic mathematical structures on which we demonstrate our idea, we briefly introduce their terms and notation. Our graphs are (mostly) finite, and undirected unless stated otherwise. A graph G has the vertex set $V(G)$ and the edge set $E(G)$. Graph edges can be simply viewed as pairs of vertices, but we sometimes consider also multigraphs (which may have parallel edges or loops). An edge with ends u and v is shortly written as uv. A complete graph on n vertices is denoted by K_n, a cycle of length n by C_n, and a path of length n by P_{n+1}. A graph is *connected* if every pair of its vertices is connected by a path. A connected graph is a *tree* if it contains no cycles.

For missing basic notation and terminology on graphs and algorithms we refer the readers to classical text books, as [CLRS01] and [Die05].

1.1 Tree-shaped structures: Motivation

Many algorithmic problems of practical or theoretical interest are NP-hard and have not found till now efficient solutions in polynomial time. Complexity counts since algorithmic problems waiting for solutions become larger and larger, caused by globalization of business: The complexity of products and production, the necessity to manage modern technologies (as VLSI circuit production, nano technology or modern robotics), the necessity to solve computational prob-

lems in biology or medicine, and the necessity to handle flood of information in globally growing networks, they all contribute to such rapid growth.

A basic observation is that complexity of computational problems often depends on structural parameters of the ingredients, i.e. the participating objects, of the problem. To develop tools to enable efficient solutions of problems of practical interest it is necessary to study possible parameters and structural properties of the participating objects and their influence to the complexity of the regarded problems.

Trees and structures. With respect to the previous observations, the most prominent classes of structures are those closely related to trees. In the history of graph theory, trees were regarded all the time as the most simple structures. Trees – connected graphs without a cycle (see Figure 1), have simple structure and so many results can be proved quite easily for them.

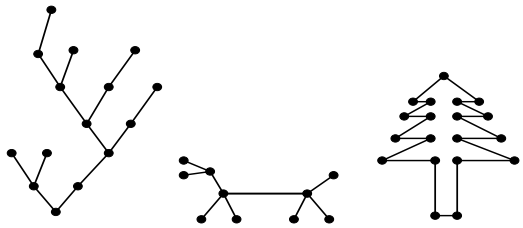


Figure 1: Examples of trees (left and middle), and a cycle (right).

On the algorithmic side, trees are also one of the best-known data structures. Many algorithmic problems can be solved efficiently by arranging or organizing the regarded data in a tree-shaped manner. Examples of algorithms making essential use of properties of trees can be found in any algorithm textbooks.

The similarity of a structure to a tree is often measured by a parameter, such as tree-width, path-width, branch-width, clique-width, rank-width. Such a "width"-parameter measures how much an object differs from a tree, or in other words how "thick" its "tree-like" structure is. These parameters can be used to analyze the structure of input objects in algorithmic problems, and this structural analysis can serve at the same time as the right tool to design an efficient algorithm to solve the regarded problem. It is the basic goal of this paper to present the core ideas of recent developments in this promising area and basic techniques to use this toolbox in applications.

Partial k-trees. Those and related good properties of trees were the reason to search for generalizations of trees in many areas of application and to look for classes of structures with similar properties. An easy way to generalize trees is to notice that a tree can be defined inductively from a vertex by adding new vertices one after another, each next adjacent to just one previous.

This idea led to the definition of *k-trees* (already in 1968, see [BP68, BP69] and [Ros74]), where one starts with a clique with k vertices, which is defined to be a k-tree, and then proceeds via induction: If T is a k-tree, then one can

enlarge T by selecting a clique H in T and choosing a new vertex b (which is not in T), and then connecting b to each vertex in H. (See Figure 2, where this principle is demonstrated in the case $k = 2$.)

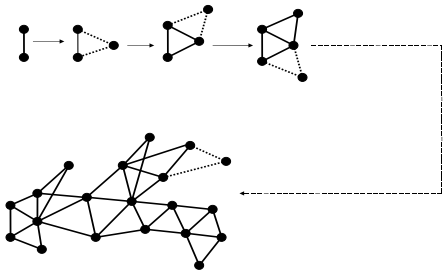


Figure 2: Construction of a 2-tree.

Here k-trees can be viewed as a straight generalization of trees, which are 1-trees for the parameter $k = 1$. Many algorithmic problems for graphs have been proved to be solvable in polynomial time or even in linear time for k-trees and for *partial k-trees*, i.e. subgraphs of k-trees, if the parameter k is fixed (see [AP89, AP86b, AP86a]). The elimination order, the inverse of the order in which the vertices are added when constructing a k-tree, can be used to find efficient algorithms for many problems using a "dynamic programming" scheme, which is demonstrated for instance in Section 1.3, Algorithm 1.2. (Briefly explaining, such an algorithm keeps certain information based on a strictly local check in the structure, and processes it along a decomposition defined by the elimination order.)

Tree-width. Partial k-trees can be used to define the first widely-known width parameter, the tree-width:

Definition. The *tree-width* $\mathrm{twd}(G)$ of a graph G is the least parameter k such that G is a partial k-tree.

There is an equivalent definition given by Robertson and Seymour, originating in 1983 [RS83, RS86] in connection with their Graph Minor Project [RS85] (also Section 1.4): First, a *tree-decomposition* of a graph G is a pair $(T, \mathcal{X})$ where T is a tree and $\mathcal{X}$ is a family of vertex sets $X_t \subseteq V(G)$, called the *bags*, indexed by the vertices t of T, such that the following holds: (i) for each edge e from G there is a vertex t of T such that both end-vertices of e belong to X_t; (ii) for all vertices v of G the subtree of T induced by $\{t : v \in X_t\}$ is connected (the interpolation property); (iii) the union of all X_t in $\mathcal{X}$ equals the vertex set of G. The width of a tree-decomposition $(T, \mathcal{X})$ is the maximum of $|X_t| - 1$ over all $t \in V(T)$. The tree-width of a graph G is then the minimum k such that G has a tree-decomposition of width k.

Equivalence of the both definitions is quite straightforward. Other possible ways of defining tree-width include a definition indirectly via an elimination ordering of graph vertices, or a new "vertex-free" definition in Section 6.1. In general we refer interested readers to a current survey of tree-width and of principles to design efficient algorithms for structures of bounded tree-width by Bodlaender [BK06]. An example of a tree-decomposition is given in Figure 3.

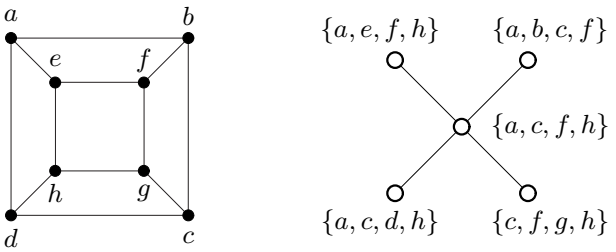


Figure 3: An example of a tree-decomposition of the cube graph of width 3, where the vertex bags are listed at the tree nodes.

Besides the notion of tree-width, other closely related notions have been found and studied in this context, such as path-width (the definition of path-width differs only by substituting the underlying tree by a path, see [RS83, Pro84]), or strong tree-with (here the sets of vertices of the tree-decomposition have to be pairwise disjoint and the edges of the graph have either both their end-vertices inside such a set or there are two neighboring sets such that the endvertices belong to the neighbors, see [See85b, See85a, See86] where such graphs are denoted as tree-partite graphs). The latter parts of our survey are devoted to new interesting additions in the area.

1.2 On the borderline of complexity: trees, grids, and information flow

In almost all areas of algorithmic applications one finds huge amount of problems which seem not to be solvable efficiently for their high complexity. (In most cases heuristic algorithms are then used to approach these problems of practical interest.) The only indication that there is likely no efficient solution possible, is often a proof that the considered problems are NP-hard. Since these problems *have to be solved* for their practical interest, one often tends to investigate whether the problems remain difficult when the class of input structures is restricted to eventually more simple structures. Hence in the literature there are many hundreds of articles proving that specially selected NP-hard problems can be solved in polynomial time for specially selected classes of input objects. It seems more beneficial trying to describe large classes of problems in a logical or algebraic calculus, and showing that all such problems can be solved more efficiently for structures with a special property (see the monographs [BLS99, DF99]).

Unification of approaches. It is one of the goals of this section to make an attempt to find an idea for a unified structural classification approach. We propose a heuristic criterion which could give a first hint to classify some of the known approaches and which can serve as a guideline to interesting open problems in this area. To feel how a structure influences the complexity of a problem, we look at computational complexity of (some) decision problems for graphs, i.e. problems where one has just to decide, whether a given graph has a property or not. Examples of such properties are:

Planarity

Instance: A graph G.

Question: Does G have an embedding without edge-crossings in the Euclidean plane?

HAMILTONIAN CYCLE

Instance: A graph G.

Question: Does G contain a Hamiltonian cycle, i.e. a subgraph of G which is a cycle containing each vertex of G?

While the first problem can be solved in linear time for all graphs, the latter one is a standard example of an NP-complete problem for which no polynomial time solution has been found until now, and no one is expected to exist.

Various problems are investigated with respect to their complexity in [GJ79, DF99, ACG+99, BLS99], and many of them are interesting for real world engineering applications. There are different attempts to make a complex decision problem tractable by restricting the class of problem instances to graphs with a special structure, say, to planar graphs or trees instead of the class of all graphs. It is a surprising observation that a lot of NP-complete problems remain NP-complete for almost all structural restrictions of the input, with the exception of structures closely related to trees. See a scheme in Figure 4.

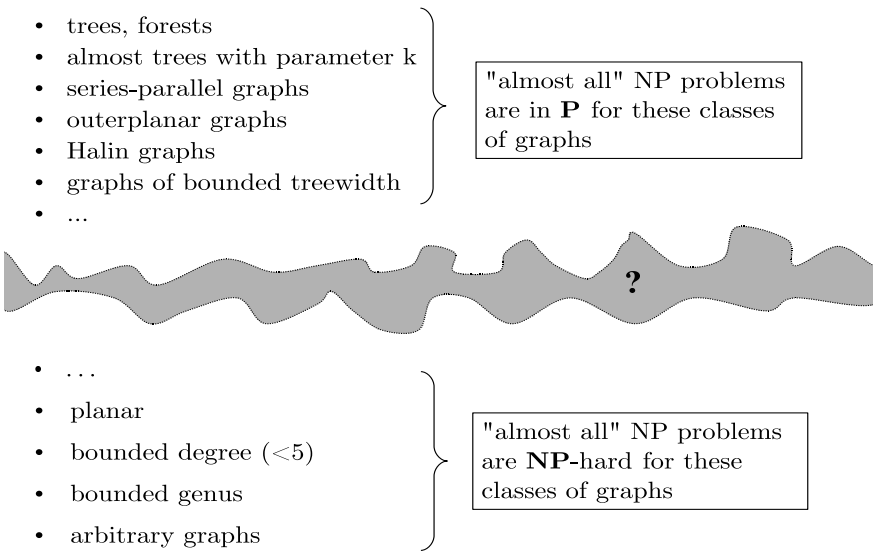


Figure 4: The borderline between P and NP

For trees and graphs with a structure "close to trees", most practical algorithmic problems are solvable in polynomial time or even in linear time. So the natural question is to search for a characterization of the borderline between these different kinds of behavior by finding a structural reason for high and low complexity. To shape the idea of such a criterion, it is useful to look at the basic idea to prove that a given problem, say $\mathcal{P}$, is NP-hard. The usual way to prove this is to choose a known NP-hard problem $\mathcal{P}'$, and to show that $\mathcal{P}'$ has a polynomial time reduction to the given problem $\mathcal{P}$.

Tiling problem. For our purposes we introduce a special "master" reduction problem which has already proved to be useful in different applications.

TILING

Instance: $D = t_0, \ldots, t_k$, a set of square tile types together with two relations $H, V \subseteq D \times D$ (the horizontal and vertical compatibility relations, respectively), and a natural number n.

Question: Is there an $n \times n$ tiling, i.e. a function $f : \{1, \ldots, n\} \times \{1, \ldots, n\} \to D$ such that (a) $f(1,1) = t_0$, and (b) for all i, j : $(f(i,j), f(i+1,j)) \in H$, and $(f(i,j), f(i,j+1)) \in V$?

One can imagine the horizontal and vertical compatibility relations as "colors" given to the tile edges, and these colors are required to match on neighboring tiles. There are many variants and applications of the tiling problem in complexity, decidability, picture recognition and physics (see for instance [AD96, Pap94]). It is not difficult to generalize tilings to covering and colouring problems for arbitrary structures with local conditions. For our applications the following result is important.

Theorem 1.1 ([Pap94, Ber66, Han74, AD96])**.** Tiling *is NP-complete if n is given in unary representation; it becomes NEXP-complete if n is given in binary representation; the problem becomes undecidable if asking for existence of an $n \times n$-tiling for all $n > 0$, even when the* origin constraint*, condition (a), is omitted.*

Assume now that we are regarding a decision problem $\mathcal{P}$ for a class K of input structures for which we are not able to find a polynomial time solution. In this case it is often conjectured that the problem is NP-hard. If we need a proof for this conjecture the only thing to do is to find a polynomial time reduction of the Tiling problem to the original problem, i.e. we have to find an algorithm F which transforms each tiling problem (D, V, H, n) in polynomial time into an input structure $F((D, V, H, n)) = G \in \mathcal{K}$ such that: There exists an $n \times n$-tiling of (D, V, H) if and only if $F((D, V, H, n))$ has the property $\mathcal{P}$.

Usually, this is accomplished by showing that

(A) some structures $G \in \mathcal{K}$ contain (in a definable way) a large grid-structure, representing the positions of the tiles in the $n \times n$-square,

(B) the local structure of these G permits coding of the tiles, and

(C) the local structure of G allows (via the problem definition) "flow of information" along the grid edges in such a way that it can be verified whether two neighboring tiles match together (horizontally and vertically).

A simple illustration of this idea is shown in Figure 5. Detailed examples can be found in [vEB83, Har83].

Reducing problem complexity. An analysis of the above idea leads us to three canonical possibilities to reduce the complexity of a problem $\mathcal{P}$ on a class $\mathcal{K}$ of structures; either by (A) avoiding the possibility to find or define large grids inside the input structures, or by (B) preventing the possibility to code the tiles, or by (C) restricting the flow of information between parts of the structure coding different tiles.

This can be achieved by

- explicitly disallowing (A) input structures which "contain" large grids (in a definable way), or by

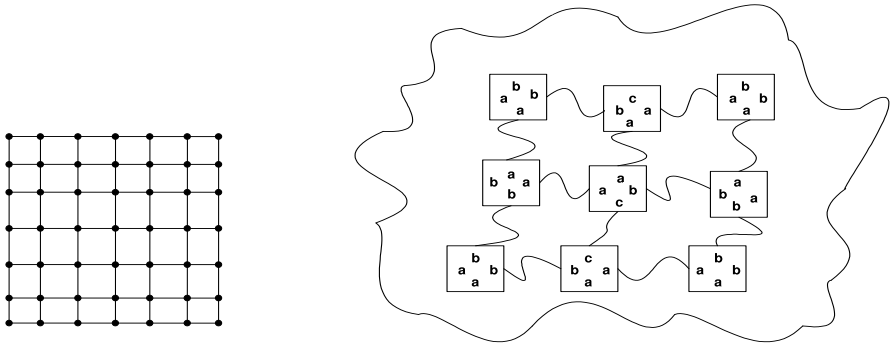


Figure 5: A picture of the 7-grid graph Q_7 on the left, and an illustration how to define a tiling inside a problem $\mathcal{P}$ (using grid-structure) on the right.

- restricting (B) the local structure of the input in such a way that it looks locally homogeneous (not allowing to code distinct tiles), or by
- simply limiting (C) the flow of information, say by limiting the "expressive power" of the problem $\mathcal{P}$ on our structure.

This criterion is of course not precise. However, even the imprecise formulation of this approach (see also [SS02]) can serve as a good guideline to search for more clearly stated approaches to reduce complexity, by giving the imprecise notions a clear mathematical meaning.

For instance stating the condition "disallow input structures with large grids" in a more precise way as "there is a fixed integer k such that the input graph does not have a k-grid Q_k (Figure 5) as a minor" leads to the class of graphs of bounded tree-width, see [RS91] and Theorem 1.4. This theorem, and also its newer generalizations to matroids [GGW03] (Section 3.4) or to vertex-minors [Oum05c] (Section 4.2), provide the structural basis to explain why the exclusion of a containment of large grids leads to problems with "tree-shaped" input structures such as those of bounded tree-width.

Furthermore, ad (B), a "homogeneous local structure" of input instances can be formally achieved by considering, for instance, graphs of bounded clique-width as discussed in Chapter 4. Another possibility, ad (B), are input structures defined by hierarchical expressions, for which several algorithmic problems can be solved efficiently, see [HLW92, Len82, Len89, Len87, LW92, LW88, LW87, Wag84]. These results have interesting applications in the area of VLSI circuits [Len86, Len90].

Ad (C), "limiting the flow of information", here large grids and inhomogeneous structures are allowed, but the flow of information ("communication") between different parts of the structure is restricted. One example of problems forbidding inherently such a comunication are the problems which can be described by first order logic. Such problems are solvable in polynomial time over all finite structures and can be solve in linar time for structures of finite degree [See96, Lib04] and for structures of locally bounded tree-width [FG01, FG04].

Further strong general result, in this context, can be obtained in combination with (A). Say, allowing only graphs of bounded tree-width as the input, and limiting the considered algorithmic problems to those definable in simple

extensions of monadic second-order logic lead to the ideas in Chapter 2 and [ALS91, BPT92, Cou92a], which explain why the majority of practical algorithmic problems can be solved in polynomial time or even linear time for structures close to trees.

Related to our (A,B,C)-approach there are many open problems. It is definitely only a heuristic idea having not reached the state of full maturity yet. However, already now its rough ideas can guide the reader through our topic of "width" parameters and an efficient algorithm design quite naturally. Some of these ideas, especially ad (A), will be refined by discussing other width parameters and their applications in the subsequent parts of this paper.

1.3 Example: Dynamic programming approach

The natural question a computer practicioner would likely ask at this point is; how can one practically use a "tree-shaped" structure (or bounded "width") of, say a graph in designing more efficient algorithms? To give an illustrating answer, we present two really simple examples of applying the dynamic programming technique in such situations, for solving problems which are otherwise very hard (NP-complete).

Independent set for bounded tree-width. A set of vertices in a graph is *independent* if no two of them are adjacent. We also refer to the definition of tree-width (Section 1.1 or [BK06]). Imagine now a given tree-decomposition $(T, \mathcal{X})$ of a graph G; and choose an arbitrary root of T. A natural dynamic programming scheme processes the bags $\mathcal{X}$ of vertices of G in the decomposition from leaves to the root, and takes advantage of the fact that only restricted information about vertices in the current (fixed-size) bag has to be kept for further processing up in the tree. Note that the size of T is linear in the number of vertices of G.

Algorithm 1.2. *Finding the size of the largest independent set in a graph G of tree-width at most fixed k, assuming a rooted tree-decomposition is given, in time $O(2^k \cdot n)$.*

We define functions $\mathcal{I}_X : 2^X \to \mathbb{N}_0$ for each bag $X \in \mathcal{X}$ of the decomposition as follows: Let $Y \subseteq X$, and let $W \subseteq V(G)$ denote the union of X and all the bags below X in the decomposition T. Then $\mathcal{I}_X(Y)$ equals the maximum cardinality of an independent set $S \subseteq W$ in G such that $S \cap X = Y$. (Note that $\mathcal{I}_X(Y) = 0$ if Y itself is not independent.)

- For each leaf bag X of the decomposition we can compute $\mathcal{I}_X$ in time $O(2^k)$ by brute force.
- Suppose that s is a node of T, having children $t_1, \ldots, t_m$. Let X_i be the bag at t_i in T, X_s be the bag at s, and $W_i \subseteq V(G)$ be defined as above. By the interpolation property of a tree-decomposition, $W_i \cap W_j \subseteq X_s$ for $i \neq j$, and moreover there are no edges in G between $W_i \setminus X_s$ and $W_j \setminus X_s$. Hence it is enough to combine the information of $\mathcal{I}_{X_i}$, $i = 1, \ldots, m$ to construct $\mathcal{I}_{X_s}$ in time $O(2^k \cdot m)$.
- Finally, we extract the size of the largest independent set $\max\{\mathcal{I}_{X_r}(Y) : Y \subseteq X_r\}$ from the root bag X_r.

Chromatic number for cographs. A *k-coloring* of a graph G is an assignment of colors numbered $1, 2, \ldots, k$ to the vertices of G in such a way that no two adjacent vertices get the same color. A graph is a *cograph* if it can be composed from single vertices by means of the following recursive definition (a composition scheme): A disjoint union of two cographs is a cograph again, and so is a disjoint union with added all edges between the two graphs. See also Section 4.3. Information restriction in this case is achieved by observing that, whenever two parts of a cograph are merged together at a node of the composition scheme, their vertices become mutually equivalent in all further operations.

Algorithm 1.3. *Finding the minimum number of colors k needed to k-color a cograph G with a given composition scheme in time $O(n)$.*

We simply proceed along the composition scheme from singletons up to G.

- A single vertex requires one color.
- If a subgraph H is composed as a disjoint union of H_1 and H_2 which require c_1 and c_2 colors, respectively, then H needs exactly $\max(c_1, c_2)$ colors.
- If, similarly, a subgraph H is composed as a disjoint union of H_1 and H_2 with adding all edges between, then H needs exactly $c_1 + c_2$ colors.

Bounded width or a grid. Finally, we mention that a dynamic approach with restricted decomposition can be sometimes successfully used in general cases. Consider for instance the following advanced idea: We are asking whether it suffices to remove k vertices from a graph G to make it acyclic (the *feedback vertex set* problem). If the tree-width of G is large enough (in k), then G contains a large grid (see Figure 5 and further Theorem 1.4) which itself defines more than k vertex-disjoint cycles in G, and so the answer is NO. Otherwise, we can construct a tree-decomposition of G of bounded width, and then use a dynamic programming scheme on the decomposition to find the right answer.

1.4 Graph minors in a shortcut

The aforementioned *Graph Minor Project* of Robertson and Seymour [RS85, RS83, RS84, RS86, RS91, RS95, RS04] undoubtedly presents a milestone in modern structural graph theory. (See [`http://www1.cs.columbia.edu/~sanders-/graphtheory/research/05C83.html`].) Because of its strong points of concurrence with our subject we think it deserves a closer sketch here. The grand nature of the whole project is probably best illustrated by the fact that the first paper of the series (of 22 currently out) has been published in 1983 while the most recent ones are being published only these days. Each of the papers delivers a number of deep mathematical results. However, those are the (sometimes really unexpected) deep and vast algorithmic applications of the project which are most important for our audience. We refer also a recent survey [Lov06] of Lovász on this topic.

Minors in graphs. The Graph Minor Project is centered around solving an old conjecture of Wagner, that finite graphs are "well-quasi-ordered" by the minor inclusion. A graph H is a *minor* of a graph G if H is obtained from G by

deleting vertices and by deleting or contracting edges. *Contracting* an edge e in a graph means to identify the ends of e into one vertex (with all their incident edges) and removing e, see Figure 6. Notice that if H or a subdivision of H are subgraphs of G, then they are also minors of G, but the converse is not always true.

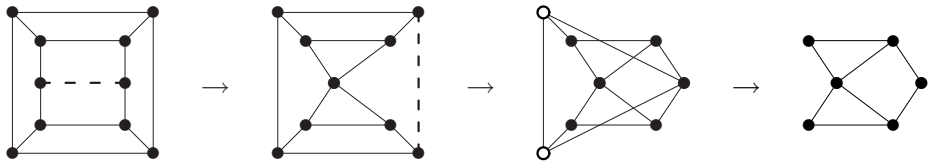


Figure 6: What results by contracting the dashed edges, and then deleting the hollow vertices, in a graph.

It turns out that various graph properties are inherited by all minors of a graph, like being acyclic, being series-parallel, or being planar. Such properties are called *minor-closed*. Importantly, the "width" parameters we study are usually minor-closed, and in relation with the material of Section 1.2 it holds:

Theorem 1.4 (Robertson and Seymour [RS91])**.** *A class $\mathcal{K}$ of graphs has universally bounded tree-width if and only if there is k such that the grid Q_k is not isomorphic to a minor of any graph in $\mathcal{K}$.*

Moreover, many graph properties can be described by forbidding graphs from a certain list as minors, these graphs being called *forbidden minors*. Such as, the Kuratowski theorem states that a graph G is planar if and only if G has no K_5- or $K_{3,3}$-minor, or a graph G is series-parallel if and only if G has no K_4-minor. The celebrated result of [RS04] can be stated as follows:

Theorem 1.5 (Robertson and Seymour [RS04])**.** *Every minor-closed class of finite graphs can be characterized by a finite list of forbidden minors.*

Minors and efficient algorithms. There is yet another very deep outcome of the Graph Minor Project—that testing presence of a minor in a graph is fixed-parameter tractable. (Obviously, testing whether G contains an H-minor with both G, H on the input is NP-hard since one can test, say, the existence of a Hamiltonian cycle in G in this way.)

Theorem 1.6 (Robertson and Seymour [RS95])**.** *For each fixed H, there is an algorithm testing in time $O(n^3)$ whether an n-vertex graph G has a minor isomorphic to H. Hence, from Theorem 1.5, every minor-closed property of graphs can be decided efficiently in cubic time.*

The last conclusion is incredibly strong, but somehow impractical. First, the lists of forbidden minors are usually very long even for simple properties, and there is no algorithmic way how to construct them in general. Second, even if the forbidden minors are eventually found, the algorithm for minor testing contains such a large "hidden constant" that it is not usable for practical implementation. Hence the importance of Theorem 1.6 is mainly theoretical. (Imagine, for instance, that someone would manage to prove NP-hardness of some minor-closed graph property!)

Yet there are many algorithmic ideas in the whole project that are practically usable, for instance those associated with graphs drawn on surfaces, or with graphs of bounded ???-width (such as tree-width). We shall present some of the latter ideas throughout our survey.

2 Role of Logic in Width Decompositions

Mathematical logic can be considered as one of the obstetricians of modern computer science, for developing the propositional and predicate calculus as ways to formally describe and handle problems in mathematics and also in a digital world. Moreover logic has created the notion of an algorithm which is the fundamental notion of modern programming. Of special importance for our problems on bounded "width" instances is a calculus denoted as monadic second-order (MSO) logic. It is the goal of this chapter to give a thorough introduction into this calculus and to present some of the key algorithmic results in this area which are also used as tools in subsequent chapters.

2.1 Second-order logic

Describing problems in a formal calculus and trying to find an efficient method to solve the problems described in this calculus sets contradicting goals. On one hand the calculus should have a large expressive power to be able to express as many problems as possible, and on the other hand its expressive power should not be too large so that the calculus has a nice model theory and one is able to find efficient solutions for all the problems in it. It is commonly accepted that the complexity class NP (containing those problems for which a positive answer can be verified efficiently) contains vast majority of problems of practical interest in their decision versions. So to identify the right language the following result is of interest.

Theorem 2.1 (Fagin [Fag74], see also [Lib04, Imm98])**.** *$NP \subseteq \exists SO$. That means each problem $\mathcal{Q}$ in NP can be described in the following way: G has property $\mathcal{Q}$ if and only if $G \models \exists X_1 \ldots \exists X_n \varphi$, for a formula φ of the usual first order predicate calculus (FO). Here $X_1, \ldots, X_n$ are variables for arbitrary finite-ary relations.*

We assume the reader is familiar with the most basic notions of syntax and semantics of first-order or predicate logic, as it can be found in any elementary textbook of logic, for instance [EFT94, Lib04, Bar77].

This result states that all problems in NP can be expressed by asking for the global existence of certain relations $X_1, \ldots, X_n$, such that these relations have a certain property φ over the structure G which can be expressed in first order predicate logic FO. This result is one of the key results of descriptive complexity theory (see [Imm87, Imm98]) which tries to capture complexity classes, as P or NP, by the expressive power of special logical languages. Here we say that a decision problem $\mathcal{Q}$ is *L-definable* over the class $\mathcal{K}$ of structures if there is an L-formula $\varphi_{\mathcal{Q}}$ such that, for all structures $G \in K$, it is that G has the property $\mathcal{Q}$ if and only if $G \models \varphi_{\mathcal{Q}}$.

With respect to our master problem to study the trade-off between structural, descriptional and computational complexity, we have to impose some

restrictions on the classes of structures and on the languages. (Since the master problem is unsolvable in a general setting.) Considering the topic of our paper, the first decision — the structural one — has already been made. For a width parameter w (which will be specified later as, say, the tree-width), a class $\mathcal{K}$ of structures is of *(universally) bounded w-width* if there is a positive integer k such that each structure $G \in \mathcal{K}$ has w-width $\leq k$.

At the next step we have to select a logical calculus. We know by Theorem 2.1, that $\exists SO$ is too strong for our reason, since there exist NP-hard problems for very simple trees (for example, the band-width problem for caterpillars with a hair length ≤ 3 [Mon86]). Hence we need a language with a more restricted expressive power. The main problem with $\exists SO$ is the possibility to quantify over arbitrary relations. Hence we shall restrict our attention to unary or monadic relations, or equivalently to sets. This leads to the calculus which is suitable for large classes of applications and which is the basis for many generalizations — the *monadic second-order logic* (MSO logic) [EFT94, EF99, Gur85]:

Definition. MSO logic is an extension of the usual first order logic (FO) by variables (usually capital letters $X_1, X_2, \dots$) running over subsets of the domain, and by the relational symbol $\in$ for membership. The usual definition of the semantics for FO-formulas is extended by the clauses

- $G \models \exists X \varphi(X) \longleftrightarrow$ there exists a subset A of the domain of G such that $G \models \varphi[A]$,
- $G \models \forall X \varphi(X) \longleftrightarrow$ for all subsets A of the domain of G it holds $G \models \varphi[A]$.

Here G is an arbitrary structure for φ, and $\varphi[A]$ results from $\varphi(X)$ by substituting the variable X by a constant which is interpreted as the set A.

A decision problem $\mathcal{Q}$ is an *MSO-decision problem* (or *MSO-problem* for short) over a class $\mathcal{K}$ of structures if there is an MSO-formula $\varphi_{\mathcal{Q}}$ such that G has the property $\mathcal{Q}$ if and only if $G \models \varphi_{\mathcal{Q}}$ for all structures $G \in \mathcal{K}$.

2.2 MSO logic on graphs

An example of a simple MSO-property is the property that a graph can be properly colored with two colors (no two adjacent vertices receiving the same color). An MSO-formula expressing this property reads

$$\exists X_1 \exists X_2 \big[\forall x \, (x \in X_1 \vee x \in X_2) \wedge \forall x \forall y \, \big(edge(x,y) \rightarrow \neg (x, y \in X_1 \vee x, y \in X_2) \big) \big] .$$

Colors are represented here as sets X_1, X_2, hence the existential quantifier block at the beginning states the existence of a coloring. The first part of the formula says that each vertex is colored, while the second part expresses that for each pair x, y of adjacent vertices their color has to be different. One can readily extend the formula to express the property that a graph can be colored with three colors, which is an NP-complete problem on all graphs: $\exists X_1 \exists X_2 \exists X_3 \big[\forall x \, (x \in X_1 \vee x \in X_2 \vee x \in X_3) \wedge \forall x \forall y \, \big(edge(x,y) \rightarrow \neg (x, y \in X_1 \vee x, y \in X_2 \vee x, y \in X_3) \big) \big]$.

Another MSO-property is the connectivity of a graph. A graph G is connected if and only if the following MSO-formula is true on G: $\forall X \forall Y \big[\exists x \in X \wedge \exists x \in Y \wedge \big(\forall x (x \in X \vee x \in Y) \wedge \neg \exists x (x \in X \wedge x \in Y) \big) \rightarrow \exists x \exists y \big(x \in X \wedge y \in Y \wedge edge(x,y) \big) \big]$. This formula says that for every nonempty partition of the

vertices of G into X, Y, there is an edge connecting a vertex from X with a vertex from Y. Connectivity is an MSO-property which is provably not expressible in first-order logic, i.e. it is not an FO-property.

Two kinds of graphs. In all these examples we assumed that graphs are coded as *adjacency structures*, i.e. with a binary relation *edge* coding the adjacency between the vertices. There is another way to code graphs, using two-sorted structures (V, E, I), where V represents the vertices, E represents the edges and I represents the incidence relation between vertices and edges. Also for this kind of structures one can build a suitable monadic second-order logic, which allows besides the usual variables for vertices and sets of vertices also variables for edges and sets of edges, and contains a symbol *inc* for the incidence relation. All the other logical parts, as connectives and quantifiers for all kinds of variables and the corresponding semantics are unchanged. To distinguish both kinds of graph logic, we denote the first as MS_1 and the latter as MS_2. (Of course, the real difference is not in logic, the difference lies in the considered classes of structures.)

It turns out that MS_2 has a higher expressive power than MS_1. For instance in MS_2 one can speak about arbitrary subgraphs H of a graph G, by selecting the vertices and the edges of H as subsets, which is not possible in MS_1. (Somehow, MS_1 can speak only about induced subgraphs; for instance, any MS_1 expression on complete graphs is quite trivial, since only complete substructures are considered, whereas MS_2 can select an arbitrary subset of edges there.) Using this idea it is easy to express in MS_2 the existence of the edge set of a Hamiltonian cycle C in a graph G; such that each vertex of C is of degree two, C is connected, and C covers all vertices of G. Courcelle has shown [Cou90, Cou94b] that this property is not expressible in MS_1.

Particularly, for a fixed graph H the property that H is isomorphic to a minor of an arbitrarily given graph G is MS_2-expressible by a formula φ_H: φ_H just has to state that for each vertex $v \in V(G)$ there is a connected subgraph X_v of G, such that all these subgraphs are pairwise disjoint, and for each edge $e = uv \in E(H)$ there is an edge e' in G such that e' connects X_u with X_v. (See [Die05].) Consequently each property definable by excluding a finite number of minors is expressible in MS_2, and hence each class of graphs closed under taking minors can be defined as an MS_2-property via the structural result of Robertson and Seymour in Theorem 1.5. This fact was used in [APS91] to show that for MSO-definable classes of graphs of bounded tree-width their minimal forbidden minors can be computed in linear time.

Besides these, many other NP-hard decision problems for graphs and networks are expressible as MS_1 or MS_2 properties, see [ALS91].

2.3 Solving MSO properties efficiently

So far we have demonstrated that the monadic second-order calculus has a sufficiently high expressive power, and now we have to think about how to solve the problems in such a calculus in an efficient way. The key tool we shall use is the so-called *interpretability method* (Figure 7). This method is basically a translation of formulas φ of one language L into formulas φ^I of another language L' , which is combined with an efficient transformation of the structures corresponding to these languages. The translation starts with

the atomic formulas of the first language L, which are substituted by formulas of the second language L'. Then one proceeds via induction on the structure of formulas (see [Rab65, ALS88, ALS91, See92] for details). The L'-formulas, which are used in the interpretation to define the atomic formulas of L, can be used in a canonical way to define in an arbitrary L'-structure H an L-structure H^I.

$$\begin{array}{ccc} \varphi \in L & \xrightarrow{\quad I \quad} & \varphi^I \in L' \\ G \in \mathcal{K} & & H \in \mathcal{K}' \\ \\ H^I \simeq G & \xleftarrow{\quad I \quad} & H \end{array}$$

Figure 7: An illustration of the concept of an interpretation I.

Such a translation is called an $O(f(n))$-*interpretation* of a class $\mathcal{K}$ of L-structures into a class $\mathcal{K}'$ of L'-structures if there is an algorithm A transforming each structure $G \in \mathcal{K}$ in time $O(f(|G|))$ into a structure $A(G) \in \mathcal{K}'$ in such a way that $G \cong A(G)^I$. This last condition guarantees that the switch from one language to the other via interpretability is truth preserving. In case that $f(n) = n$, we speak about *linear-time interpretability*.

Lemma 2.2 (Rabin [Rab65])**.** *For each sentence φ of L and each structure $G \in \mathcal{K}'$ it holds: $G \models \varphi^I$ if and only if $G^I \models \varphi$.*

We cannot describe this method formally here due to space restrictions, but we refer interested readers to the references. The interpretability method was originally developed to prove the decidability or undecidability of theories in mathematical logic by Rabin [Rab65, Rab77]). Compton and Henson [CH87] used it to deduce lower bounds for the complexity of theories. Later it was adapted to linear and polynomial time computable decision and optimization problems in [ALS88, ALS91, See92]. See also [CM93, Cou94a, Cou97] where a variant of it is called a *transduction*. A key result for interpretability is the following:

Theorem 2.3 ([ALS88, ALS91])**.** *Let L, L', $\mathcal{K}$ and $\mathcal{K}'$ be as above. If there exists an $O(f(n))$-time interpretation of $\mathcal{K}$ into $\mathcal{K}'$ with respect to L and L', and if each L'-definable problem for $\mathcal{K}'$ can be solved in time $O(f(n))$, then each L-definable problem can be solved in time $O(f(n))$.*

The reason for usefulness of this method is that one does not have to look for a special algorithm for each special problem $\mathcal{P}$ defined by a formula φ in a language L. Instead, one just shows that the regarded class $\mathcal{K}$ of structures is efficiently interpretable into another class $\mathcal{K}'$ of structures with respect to L and a corresponding language L', for which efficient algorithms are already known.

We build on the following result.

Theorem 2.4 (Courcelle [Cou92a], also [ALS88, ALS91], implicitly [TW68])**.** *Each MSO-problem can be solved in linear time for the class of binary trees.*

Here a slightly modified variant of trees, called *binary trees*, is used: The *full binary tree* is the structure $(\{0,1\}^*, sc_0, sc_1)$, where $\{0,1\}^*$ is the set of all

finite words over the alphabet $\{0,1\}$, and the successor functions sc_0 and sc_1 are defined as $sc_0(w) = w0$ and $sc_1(w) = w1$ for each word $w \in \{0,1\}^*$. A *binary tree* is then a restriction of the full binary tree to an arbitrary set of words $\Sigma \subseteq \{0,1\}^*$, which has to be closed under initial segments (see Figure 8).

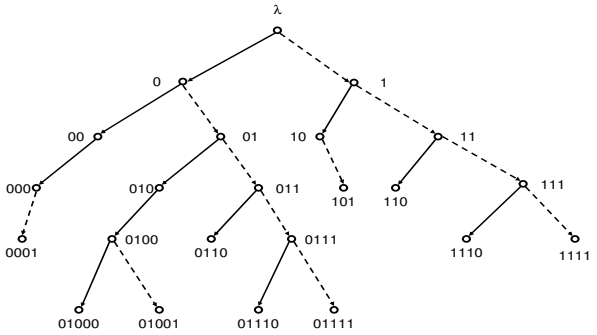


Figure 8: An example of a binary tree with root λ.

The basic idea to prove Theorem 2.4 is to use the result of [TW68]; for each MSO-formula φ there is a constructible tree-automaton A_φ such that for each binary tree T, $T \models \varphi$ if and only if A_φ accepts T. Informally speaking, a tree-automaton is a finite automaton working on a tree in such a way that it starts from the leaves of the tree, and computes on its way up to the root an evaluation of the vertices of the tree with elements from its finite state set. This evaluation is called a *run* of the automaton. If the state computed for the root is in the set of final or accepting states, then the automaton accepts the tree, otherwise it does not accept it. This process follows in a way the scheme of dynamic programming, and a run can easily be emulated in linear time.

Now we combine Theorem 2.4 with a suitable linear-time interpretation, Theorem 2.3. This is nothing else than a translation of the description of a tree-decomposition in MSO logic (see [ALS88, ALS91]), with a construction of the tree-decomposition for an arbitrary graph of bounded tree-width in linear time (see Bodlaender [BK06]). All these ideas together lead to a proof of the following theorem, explaining why all MSO-problems (MS_1, to be accurate) can be solved in linear time via a dynamic programming scheme in these settings.

Theorem 2.5 (Courcelle [Cou92a], also [ALS88, ALS91])**.** *Each MSO-problem can be solved in linear time for an arbitrary graph class of universally bounded tree-width.*

Extensions of MSO properties. This result can easily be generalized to other extensions of monadic second-order logic, to EMSO-problems which are more adapted to optimization problems for evaluated graphs and networks, and to counting monadic second-order logic. We describe the *EMSO-problems* first, which have been introduced in [ALS88, ALS91]. Note that we now consider graphs as two-sorted incidence structures (i.e. we refer to the stronger language MS_2).

Let $\mathcal{K}$ be a class of structures and assume that the regarded structures are provided additionally with m functions $f_1^G, \ldots, f_m^G$, evaluating vertices or edges with rational numbers. This evaluations have canonical extensions to subsets

A of the domain (in case of graphs to sets of vertices or edges) by defining $f_i^G(A) = \sum_{a \in A} f_i^G(a)$.

From these functions arithmetic expressions can be built by using the operations $+, -$ and $\times$. Linear expressions are built by using $+$ and $-$ only. These expressions are denoted as *(linear) evaluation terms*. A *(linear) evaluation relation* results by comparing such (linear) evaluation terms with rational constants via $=, \leq$ or $<$.

Definition. A *linear EMSO-problem over* $\mathcal{K}$ is a decision problem, definable for all structures $G \in \mathcal{K}$, in the form "there are subsets $A_1 \ldots A_k$ of the domain of G such that $G \models \varphi[A_1, \ldots, A_k]$ and $\psi(A_1, \ldots, A_k)$ holds", where φ is an arbitrary MSO-formula for $\mathcal{K}$ and ψ is a Boolean expression built from linear evaluation relations.

These problems are called *LinEMSO-problems* over $\mathcal{K}$. *EMSO-problems* over $\mathcal{K}$ are defined in the same way by simply dropping the word linear.

Definition. A problem is a *(linear) EMSO-optimization problem over* $\mathcal{K}$ ((Lin)EMSOopt for short) if it can be stated as "$\max \{\psi_1(A_1, \ldots, A_k) : G \models \varphi[A_1, \ldots, A_k]$ and $\psi_2(A_1, \ldots, A_k)$ holds, where $A_1, \ldots, A_k$ are subsets of the domain of $G\}$". Here ψ_1, ψ_2 are (linear) evaluation terms and φ is an MSO formula for the language corresponding to G.

The operator min can also be allowed in such problems, since it is expressible in terms of maximization and the negation which are allowed in evaluation terms. The LinEMSOL optimization problems over $\mathcal{K}$, introduced in [CMR98, CO00], follow essentially the definition of linear EMSO optimization problems from [ALS88, ALS91], but they restrict the constants to integers and do not allow arbitrary evaluation relations.

A problem to compute the cardinality $\left|\{(A_1, \ldots, A_l) : G \models \varphi\,[A_1, \ldots, A_l]\}\right|$ for an MSO formula $\varphi(X_1, \ldots, X_l)$ and a given structure $G \in \mathcal{K}$, is called an *MSO enumeration problem* over $\mathcal{K}$ (*MSOenum*). Furthermore, *CMSO-properties* are the properties formulated in counting monadic second-order logic (CMSO logic), which was introduced in [Cou92a] and which results as an extension of MSO logic by allowing counting cardinality of sets (of vertices or edges) modulo k for positive integers k. We will write LinECMSOopt, ECMSOopt, ECMSO, CMSOenum, if CMSO logic is used instead of MSO.

The following theorem is the key result for a general reduction of algorithmic problems on graphs whose structure is closely related to trees.

Theorem 2.6 (Arnborg, Lagergren, and Seese [ALS88, ALS91])**.** *For each natural number m, every class $\mathcal{K}$ of graphs of tree-width bounded by m is evaluation-preserving linear-time interpretable into the class of binary trees. Hence each MSO-, LinEMSO-, LinEMSOopt-, MSOenum-, LinECMSO-, LinECMSOopt-, and CMSOenum-definable problem can be solved in linear time for graphs in $\mathcal{K}$.*

The advantage of this approach is that to solve a problem it is sufficient to describe it in a suitable language. The algorithm for the solution then comes out automatically by translating the formulas describing the problem into equivalent formulas describing an equivalent problem (via interpretation) for binary trees, and solving the problem via dynamic programming. This result was proved originally in [ALS88, ALS91] for graphs using a slightly different terminology. But literally the same proof can be used to prove Theorem 2.6. The only additional remark is to add for the CMSO cases. Cour-

celle stated in [Cou92a] that CMSO logic is provably a strict extension of MSO logic, since it is not definable in pure MSO for arbitrary structures. But nevertheless CMSO-problems for structures of bounded tree-width can be reduced to MSO-problems for binary trees (see also [See96]), since CMSO logic is definable in MSO logic for binary trees. For graphs of bounded tree-width it was proved in [Cou92a] that each MSO- and CMSO-problems can be solved in linear time. This approach basically generalizes many approaches from [TNS82, SS89, Wim87, BLW87, Bod88, FL89, HR89, HR90].

Additional remarks. The results of this chapter give a theoretical explanation why so many practical problems for input structures without large grids (cf. Section 1.2) can be solved efficiently; since the graphs without large grids have bounded tree-width (Theorem 1.4), and hence we may usually apply Theorem 2.6 for finding their solutions. Moreover, the outlined proof ideas (a relation to tree automata) explain that a dynamic programming scheme on a tree-decomposition is the best choice for their practical implementation.

It is surprising that these ideas work even for other classes of structures which "do not contain large grids" in different meaning. For instance the usual grid minors (in the meaning of Robertson and Seymour) are generally not MS_1-definable, but there is another recent notion of a containment (a vertex-minor) which is CMS_1-definable, and it corresponds to bounding "clique-width / rank-width" of a graph. This is further discussed in Chapter 4.

3 Graph Branch-Width and its Generalizations

One should surely note that the Graph Minor Project [RS84, RS85] (see Section 1.4) of Robertson and Seymour, which first formally defined the graph tree-width, brought also a formal definition of a so-called branch-width [RS91]. These two width notions on graphs are closely related to each other (Theorem 3.1), and they both have their advantages. Undoubtedly, it is the tree-width that got much more attention among computer scientists over the past two decades, which we consider a pity. The main purpose of this chapter is to show the beauty, advantages, and extensions of the branch-width parameter.

3.1 Connectivity and branch-width

We gradually introduce the notion of branch-width from a simple view on graphs to a general abstract definition. We briefly recall that the notion of tree-width is related with an "ordinary view" of graph connectivity considering (the size of) *vertex cuts* separating components: The bag at a tree-decomposition node forms a vertex cut separating vertices appearing in different branches of the node. (This is the crucial property that allows us to design efficient dynamic algorithmic schemes along tree-decompositions.)

Considering the branch-width, it is helpful to show the readers another, very similar though not exactly the same, view of connectivity in graphs. Now instead of asking for a vertex cut separating two parts of a graph from each other, we look for a partition of the graph edges (called an edge *separation*) such that the two parts share small number of vertices in common (called the *guts* of the separation). Mathematically, let G be a graph with the edge set E,

and consider a separation $(F, E \setminus F)$ of the edges. (Of course, the separation is symmetric and it is determined by one of its sides, say F.) The *connectivity value* $\lambda_G(F)$ of the separation is defined as the number of vertices of G that are incident both with an edge in F and with an edge in $E \setminus F$.

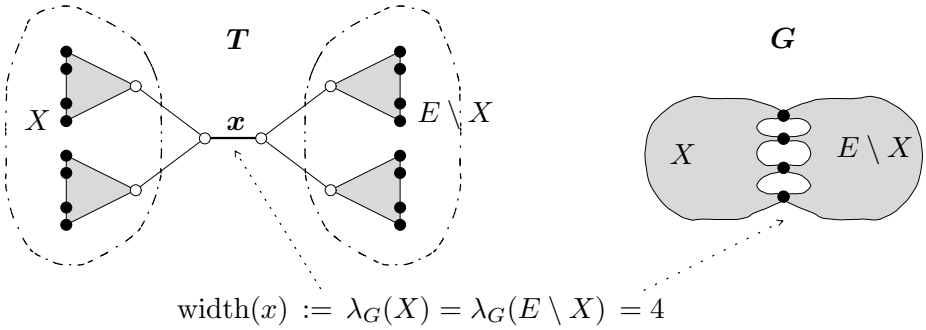


Figure 9: An illustration to the definition of a branch-decomposition.

Branch-decompositions. A tree is *sub-cubic* if every node has degree 1 or 3. Imagine that the edges of a graph G are "decomposed" into the leaves of a sub-cubic tree; precisely each leaf of the sub-cubic tree T holds one edge of G. Such is called a *branch-decomposition* T of G. Then an edge x of T divides T into two components, and hence x defines a separation $(X, E \setminus X)$ of the edges of G where X are the edges mapped to one component of $T - x$ and $E \setminus X$ to the other one. We say that the *width* of the edge x in T is the connectivity value $\lambda_G(X) = \lambda_G(E \setminus X)$. See Figure 9. The width of whole T is the maximum over widths of its edges.

The *branch-width* $\mathrm{bwd}(G)$ of a graph G is the minimum width over all branch-decompositions of G. Some properties of branch-width are summarized here [RS91]:

- Branch-width of G is 0 if and only if G has no component with more than one edge.
- Branch-width of G is at most 1 if and only if G is a forest of stars.
- Branch-width of G is at most 2 if and only if G is a "series-parallel" graph. Series-parallel graphs can be characterized as graphs with no minor isomorphic to K_4, and they are exactly graphs of tree-width at most 2.
- The same definition of branch-width can be immediately applied to hypergraphs (i.e. structures in which an edge may have more than two end-vertices), and many of the properties remain true.

Moreover, the basic result relating branch-width to tree-width is proved in [RS91, Theorem 5.1]. We present a simplified formulation:

Theorem 3.1 (Robertson and Seymour [RS91])**.** *Let G be a graph of tree-width t and branch-width $b > 1$. Then*

$$b \leq t + 1 \leq \left\lfloor \frac{3}{2} b \right\rfloor .$$

Sketch of proof. Let us have a branch-decomposition of G of width b. For a node u of the decomposition, denote by $W_u^1, W_u^2, W_u^3 \subseteq V(G)$ the guts of the separations displayed by the three incident edges to u. It is easily seen that a vertex of G occuring in one of W_u^1, W_u^2, W_u^3 has to appear also in another one of those, and hence the cardinality of $B_u = W_u^1 \cup W_u^2 \cup W_u^3$ is at most $\lfloor 3b/2 \rfloor$. So the tree-decomposition formed by the same tree with the bags B_u has width at most $\lfloor 3b/2 \rfloor - 1$. In the other direction, start with a tree-decomposition of G of width t. The decomposition can be "split" by duplicating bags to get a sub-cubic underlying tree, with one leaf for each edge of G. That, in turn, provides a branch-decomposition of G of width at most $t+1$ by the interpolation property. □

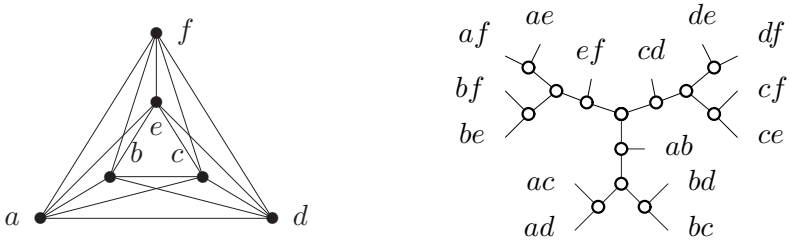


Figure 10: A branch-decomposition of width 4 of the complete graph K_6.

It is good to understand why the both bounds are best possible in Theorem 3.1: For the right-hand inequality, the complete graph K_{3r} has tree-width $3r-1$ and branch-width only $2r$ (using a decomposition sketched in Figure 10). On the other hand, the complete bipartite graph $K_{s,s}$ has branch-width s and tree-width also s. A little modification — removing the edges of a perfect matching from $K_{s,s}$ results in a graph of tree-width $s-1$ and branch-width still s. So also the left-hand inequality is best possible.

Branch-width of connectivity functions. One of the attractive properties of branch-width is that it can be readily extended to all combinatorial structures possessing a reasonable measure of connectivity. We make this abstract idea mathematically precise now.

Let E be a finite set and λ be an integer-valued function defined on the subsets of E. Then, following Robertson and Seymour [RS91, Section 3], we say that λ is a *connectivity function* if, for all $X, Y \subseteq E$,

1. $\lambda(X) = \lambda(E \setminus X)$ (symmetric), and
2. $\lambda(X) + \lambda(Y) \geq \lambda(X \cap Y) + \lambda(X \cup Y)$ (submodular).

Note that a graph G has a natural connectivity function λ_G defined on edge separations above. It is easily seen that λ_G is a connectivity function in the sense defined here.

Definition. Let λ be a connectivity function on a ground set E. A *branch-decomposition* of λ is a pair (T, ω) where T is a sub-cubic tree, and ω is a bijection of E onto the leaves of T. For an edge x of the tree T, we denote by T_x one of the connected components of $T - x$ and by $X = \omega^{-1}\big(V(T_x)\big)$, i.e. X are those points of E that are mapped to the leaves of T_x. We say the x

displays the partition $(X, E \setminus X)$, and define the *width* of x as $\lambda(X)$. (Figure 9.) The width of the decomposition (T, ω) is the largest width over all edges of T, and the smallest width over all branch-decompositions of λ is the *branch-width* $\mathrm{bwd}(\lambda)$ of the function λ. If $|E| \leq 1$, then we define branch-width of λ as $\lambda(\emptyset)$. (Notice that $\lambda(\emptyset)$ may be nonzero.)

3.2 Properties and advantages

Recall (Section 1.4) that a graph H is a *minor* of a graph G if H is obtained from a subgraph of G by contracting edges. Likewise tree-width, branch-width is also a minor-monotone property as we sketch for an illustration:

Proposition 3.2. $\mathrm{bwd}(H) \leq \mathrm{bwd}(G)$ *for all minors* H *of a graph* G*.*

Sketch of proof. It is clearly enough to prove the claim in the cases when H results by deleting, or by contracting, one edge e in G. We use the tree of the decomposition of G, but with the leaf of e removed (and its neighbor changed to an edge). One may easily verify that no separation displayed by this branch-decomposition of H contains more vertices in its guts than the corresponding displayed separation in G. □

Hence, in particular, there is a finite list of obstructions for a graph having branch-width at most k for each k by Theorem 1.5. It is an easy exercise to find the obstructions for $k = 0, 1$.

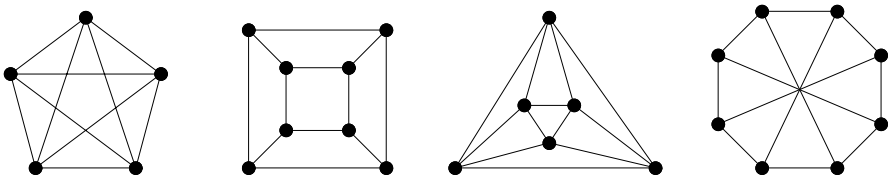


Figure 11: The four forbidden minors for the graphs of branch-width at most 3.

Graphs of small branch-width. The graphs of branch-width 0, 1 or 2 are quite easy to describe, as we have seen in Section 3.1. Thus the smallest non-trivial value for branch-width of a graph is 3. Quite a lot is known nowadays about structure of the branch-width-3 graphs. First, they can be characterized by just four simple obstacles — the forbidden minors in Figure 11. The obstacles appear published first in [BT99], but they have been known already to Dharmatilake, Chopra, Johnson, and Robertson [Dha94]. Second, [BT99, Theorem 6] a graph has branch-width at most 3 if and only if it has tree-width at most 3 and contains no minor isomorphic to the cube Q_3. Hence we may regard the cube as the "minimal" graph showing a nontrivial structural difference between the notions of tree-width and branch-width.

Third, Bodlaender and Thilikos have given [BT99, Theorem 9] a small set of reduction rules (Figure 12) that fully characterizes all the graphs of branch-width at most 3, and one can construct a branch-decomposition for such graphs in linear time. Naturally, one would like to extend such nice structural characterizations to higher values of branch-width. Unfortunately, only weaker partial

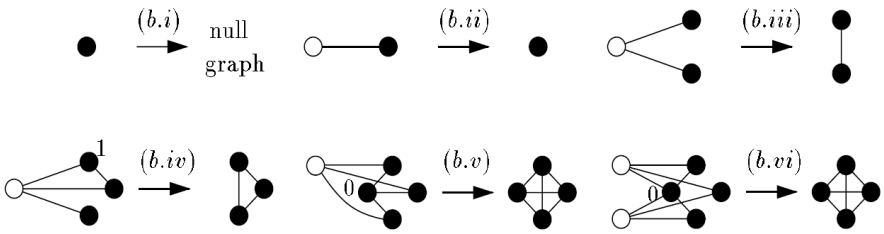


Figure 12: [BT99] Reduction rules for the graphs of branch-width at most 3. (Hollow vertices have all neighbors shown, and they are removed in the reduction.)

results in the case of branch-width 4 are known so far, for instance, see [Rig01] for a set of reduction rules and some planar obstructions.

General algorithms. Testing if branch-width of a graph is at most k is NP-complete for k on the input [ST94, Theorem 8.10]. On the other hand, graph branch-width is a fixed-parameter tractable property [BT97], but the algorithm is not easy and likely not practically usable:

Theorem 3.3 (Bodlaender and Thilikos [BT97])**.** *For each fixed k, there is a constructible algorithm that in linear time checks whether a given graph has branch-width at most k, and, if so, outputs an optimal branch-decomposition.*

Miraculously, for planar graphs there is an algorithm [ST94, Algorithm 7.3,9.1] for computing the branch-width in fully polynomial time.

Theorem 3.4 (Seymour and Thomas [ST94])**.** *There is an algorithm that, given a planar graph G and an integer k, decides in time $O(m^2)$ whether G has branch-width at most k (where $m = |V(G)| + |E(G)|$). If so, then an optimal branch-decomposition of G is computed in time $O(m^4)$.*

It is worth to mention that, unlike many other involved algorithms associated with the Graph Minor Project, this algorithm is really implementable and practical. Note some recent branch-width implementation and computing experiments by Hicks [Hic05a, Hic05b]. Theorem 3.4 gives us one clear reason why to use branch-width instead of tree-width; since determining the tree-width stays NP-complete even on planar graphs. This fact, and some nice structural properties of branch-width for planar graphs, are used for instance in [FT03] to obtain better subexponential exact algorithms for hard problems on planar graphs.

Another reason why to use branch-width instead of tree-width may come out in some dynamic programming schemes, taking an advantage of the fact that a branch-decomposition does not explicitly refer to graph vertices which makes processing simpler. For instance, Noble [Nob98] computes the (notoriously hard) Tutte polynomial of a graph of bounded tree-width using ideas that are actually much better suited for branch-decompositions, as one can see also in [Hli06d].

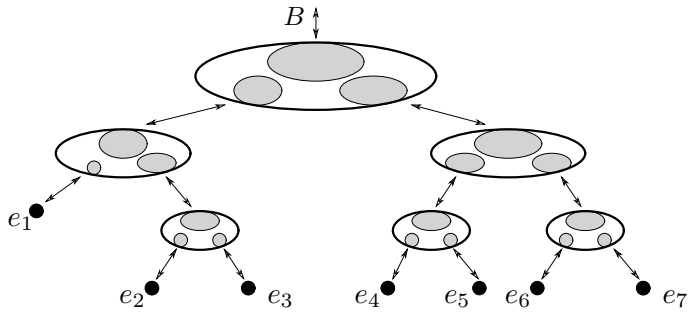


Figure 13: A simple illustration of a (boundaried) parse tree of a branch-decomposition. The guts (boundaries) are shaded.

3.3 Dynamic programming and parse trees

In this section we return to the topic of Section 1.3 — use of a dynamic programming scheme to solve problems more efficiently — from a formal perspective of branch-width and of graph "parse trees". This is a useful (and reasonably simple) mathematical formalism for handling tree-like decomposition schemes. For a nice thorough introduction to graph parse trees with relation to tree-width we refer to [DF99, Chapter 6]. We remark that this topic is also very close to so-called "graph grammars" which we are not able to cover here due to space restrictions.

Imagine a graph G with designated (labeled) subset B of k ordered vertices; hence $B \subset V(G)$ is actually a sequence of length k. (The size of whole G is arbitrary with respect to k.) We call such G a *k-boundaried* graph, and B is the *boundary* of G. We write (G, B) for a reference. Then we can naturally introduce *(k-boundaried) composition operators* over boundaried graphs, that specify which pairs of boundary vertices (labels) are identified in a composition.

Under this simple formal view, we easily transform an (arbitrarily rooted) branch-decomposition of a graph G into a rooted *parse tree* T, whose leaves keep the edges of G and internal nodes specify composition operators used to "glue" the edges together to form G (Figure 13). If the width of the decomposition is k, that means if the guts of each separation of G displayed by an edge of T contains at most k vertices, then k-boundaried composition operators are clearly enough to define the parse tree T, and vice versa.

Example: Enumeration of matchings. A set of edges M in a graph is a *matching* if no two edges of M share a common vertex. Although one can find the size of a maximum matching in a graph efficiently, the task to compute the total number of matchings in a given graph is one of prominent #P-complete problems. (This task plays quite important role, say, in statistical physics.) For a brief explanation, the class #P [Val79] is the enumerative counterpart of the class NP. Hence an efficient algorithm for computing the number of matchings is desirable even in special cases.

For a boundaried graph (G, B), we denote by $m_G(A)$ the number of such matchings in G that hit the boundary B in a subset $A \subseteq B$. Then we formally define a multivariete polynomial $P_{(G,B)}(x_A : A \subseteq B) = \sum_{A \subseteq B} m_G(A) \cdot x_A$. The number of matchings in G is evaluated as $P(1, \ldots, 1)$. We apply the following

easy dynamic procedure:

Algorithm 3.5. *Computing the above polynomial $P_{(G,B)}$ (and so the number of perfect matchings) in a graph G of branch-width at most fixed k, assuming the corresponding parse tree T is given, in $O(2^{2k} \cdot n)$ steps.*

- For each leaf of the parse tree T, holding an edge $e = uv$, the subgraph formed by e has boundary $\{u, v\}$, and $P_e = x_{\{u,v\}} + x_\emptyset$ since there are precisely those two matchings in a one-edge graph.
- Let, at an internal node of the parse tree T, the left subtree define a boundaried subgraph (H_1, D_1) and the right one a boundaried subgraph (H_2, D_2) of G, and let the composition at this node result in a boundaried subgraph (H, D). We simply multiply the (recursively obtained) polynomials $P_{(H_1,D_1)}$ and $P_{(H_2,D_2)}$

$$Q = P_{(H_1,D_1)} \cdot P_{(H_2,D_2)}\,,$$

and we define the polynomial $P_{(H,D)}$ by substitutions ($A_1 \subseteq D_1$, $A_2 \subseteq D_2$)

$$x_{A_1} \cdot x_{A_2} = \begin{cases} 0 & \text{if } A_1 \cap A_2 \neq \emptyset \\ x'_A & \text{where } A = (A_1 \cup A_2) \cap D, \text{ otherwise.} \end{cases}$$

We leave verification of this straightforward algorithm to the readers. (Notice how much simpler is this formal setting than if one considered a tree-decomposition of the graph G in the dynamic scheme.)

3.4 Extending branch-width to matroids

Matroids present a wide combinatorial generalization of graphs towards their "geometric essence". Such generalization is not purposeless since, in some situations, a "matroidal view" of a problem can bring new ideas or solutions even to ordinary graph problems. That is also the case of the branch-width parameter which has an immediate natural meaning on matroids. Since matroids (besides the greedy algorithm) are not much known among computer scientists, we include all necessary matroid definitions here.

Matroids. We refer to Oxley [Oxl92] for matroid terminology. A *matroid* is a pair $M = (E, \mathcal{B})$ where $E = E(M)$ is the finite ground set of M (elements of M), and $\mathcal{B} \subseteq 2^E$ is a nonempty collection of *bases* of M, no two of which are in an inclusion. Moreover, matroid bases satisfy the "exchange axiom"; if $B_1, B_2 \in \mathcal{B}$ and $x \in B_1 \setminus B_2$, then there is $y \in B_2 \setminus B_1$ such that $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. Subsets of bases are called *independent sets*, and the remaining sets are *dependent*. Minimal dependent sets are called *circuits*. The *rank function* $\mathrm{rank}_M(X)$ in M assigns to each $X \subseteq E(M)$ the maximum cardinality of an independent subset of X.

If G is a graph, then its *cycle matroid* on the ground set $E(G)$ is denoted by $M(G)$. The independent sets of $M(G)$ are the forests of G, and the circuits of $M(G)$ are the cycles of G. In fact, a lot of matroid terminology is inherited from graphs. Another typical example of a matroid is a finite set of vectors with usual linear dependency. The two examples are jointly illustrated in Figure 14.

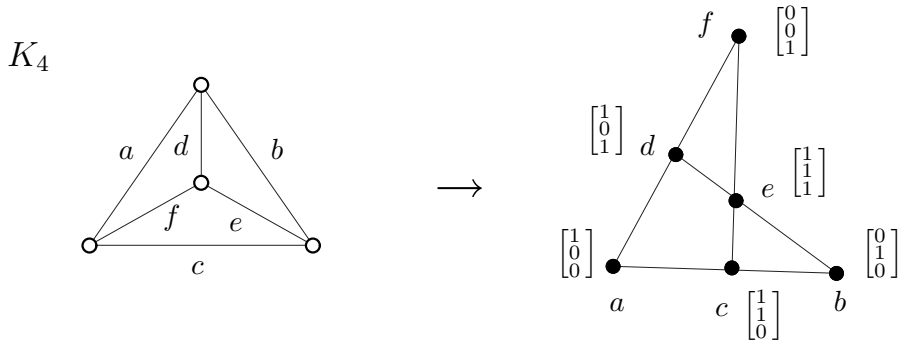


Figure 14: An example of a vector representation of the cycle matroid $M(K_4)$. The matroid elements (edges of K_4) are depicted by dots on the right, and their linear dependency is shown by lines.

The *dual* matroid M^* of M has the same ground set E, and the bases of M^* are the set-complements of the bases of M. (This corresponds to usual topological duality of planar graphs.) An element e of M is called a *loop* (a *coloop*), if $\{e\}$ is dependent in M (in M^*). The matroid $M\backslash e$ obtained by *deleting* a non-coloop element e is defined as $(E \setminus \{e\}, \mathcal{B}^-)$ where $\mathcal{B}^- = \{B : B \in \mathcal{B},\, e \notin B\}$. The matroid M/e obtained by *contracting* a non-loop element e is defined using duality $M/e = (M^*\backslash e)^*$. (This corresponds to contracting an edge in a graph.) A *minor* of a matroid is obtained by a sequence of deletions and contractions of elements.

Matroid connectivity. The connectivity function λ_M of a matroid M is defined for all $A \subseteq E$ by $\lambda_M(A) = \mathrm{rank}_M(A) + \mathrm{rank}_M(E \setminus A) - \mathrm{rank}_M(E) + 1$.

Geometrically, the affine closures of the two sides of a separation $(A, E \setminus A)$ intersect in a subspace of rank $\lambda_M(A) - 1 = \lambda_M(E \setminus A) - 1$ (such as, in a line if $\lambda = 3$). It is remarkable that the matroid connectivity function is preserved under duality, $\lambda_M \equiv \lambda_{M^*}$. Recalling the graph connectivity function λ_G from Section 3.1, we note that these two function are equal $\lambda_G(A) = \lambda_{M(G)}(A)$ over a graph G only if the edges of both sides A and $E \setminus A$ form connected subgraphs in G.

Definition. The *branch-width* of a matroid M is the branch-width of its connectivity function λ_M.

As one can see, the matroid branch-width definition corresponds to graph branch-width, but it is not an exact generalization due to possible differences between graph and its matroid connectivity functions. The concept is illustrated in Figure 15.

Properties of branch-width. We now summarize some folklore simple properties of matroid branch-width that are closely related to graph branch-width.

- Branch-width of a matroid M is always at least $\lambda_M(\emptyset) = 1$.
- Branch-width of M is 1 if and only if M has no dependent set or, dually, if M has rank 0.

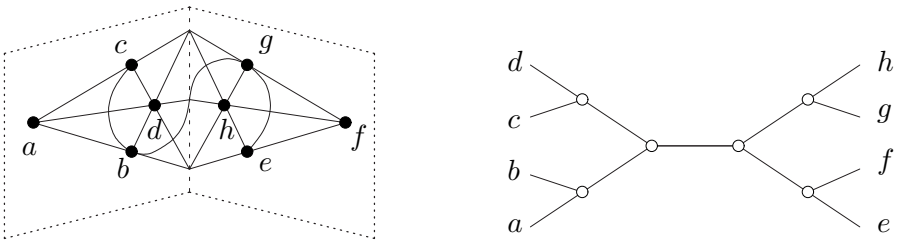


Figure 15: An example of a branch-decomposition of width 3 of a rank-4 matroid.

- Branch-width of M is at most 2 if and only if M is the cycle matroid of a graph of branch-width at most 2.
- Branch-width of M equals the branch-width of its dual M^*.
- Branch-width of M is minor-monotone, i.e. $\mathrm{bwd}(N) \leq \mathrm{bwd}(M)$ for all minors N of M.

One can see here a small difference between the branch-width notions of a graph and of its cycle matroid. For instance, the branch-width of a tree that is not a star is 2, while the cycle matroid of a tree has branch-width 1. Surprisingly, a general relation between graph and matroid branch-widths has been open for long time. The (expected) answer has been claimed to be proved only recently, by Hicks, McMurray [HMJ05], and by Mazoit, Thomassé [MT05] independently.

Theorem 3.6 ([HMJ05], [MT05])**.** *Branch-width of a bridgeless graph (every edge contained in a cycle) is equal to the branch-width of its cycle matroid.*

In particular, Theorem 3.6 provides a simple reason why the branch-width of a planar graph containing a cycle equals the branch-width of its topological dual. (This nontrivial corollary follows already from [RS91] or [ST94], although it is not explicitly stated there.) Note that the tree-width of a planar graph may differ from the tree-width of its dual, as an example of the cube and the octahedron shows.

Forbidden minors. Analogously to Section 3.2, we may ask about the forbidden minors for matroids of branch-width $\leq k$. The answer, however, gets complicated already for $k = 3$. For binary matroids (those representable over $GF(2)$) only, a list of 10 forbidden minors for branch-width ≤ 3 has been found by Dharmatilake [Dha94], and proved to be complete by Hliněný [Hli02]. But, over thousand more non-binary forbidden minors for the matroids of branch-width ≤ 3 have been constructed [Hli04] subsequently. That leaves little hope for finding a complete answer in general.

Although matroid branch-width is minor-monotone, we remark that no analogue of Theorem 1.5 is true over all matroids [GGW02]. Hence it is a question whether the list of forbidden minors for the matroids of branch-width $\leq k$ is finite after all. The affirmative answer for all k's is given by Geelen, Gerards, Robertson, and Whittle [GGRW03], showing an explicit size bound of at most $(6^{k+1} - 1)/5$ elements for such forbidden minors. So it is a matter of a finite, although currently infeasible, computer search to determine the lists for all k's.

Regarding matroidal generalizations of Theorem 1.5, Geelen, Gerards, and Whittle [GGW02] have proved that each minor-closed class of matroids of bounded branch-width which are representable over a fixed finite field has a finite list of forbidden minors. The more general case of minor-closed classes among all matroids representable over a finite field is, still, an important open question in structural matroid theory. (Recall that the cycle matroids of graphs are representable over every field.)

3.5 Parametrized algorithms on matroids

Thinking about computational complexity of matroid algorithms, a computer scientist soon recognizes a complication: In order to give a complete information about an n-element matroid M, it is necessary to say something about each subset of the ground set and the amount of information is exponential in n. Since accepting that as the input size would ruin usual complexity measures, one has to find another resolution. For example, it is possible to give just the ground set E on the input, together with an *oracle* (black box) answering queries about independence or rank of subsets of E in M.

Another possibility is to give a suitable, polynomially-sized *representation* of the matroid M. The most common method is to input a matrix as a representation of M: A matrix $\boldsymbol{A}$ over a field $\mathbb{F}$ is a *vector representation* of a matroid M if the elements of M are the column vectors of $\boldsymbol{A}$ and independent sets of M are the linearly independent sets of vectors. See again Figure 14. We remark that $GF(q)$ denotes the finite field with q elements where $q = p^r$ is a prime power, and p is the characteristic of the field. With a vector representation, the elements of a matroid become "real points" in the projective geometry over the field $\mathbb{F}$. Note, however, that not all matroids can be represented by vectors / matrices. (The situation with representable matroids somehow resembles graphs embeddable on surfaces.)

Computing matroid branch-width. As with graphs, the first nontrivial value of branch-width of matroids is 3. There is a simple and practical algorithm [Hli02] which tests whether a matroid (given by an oracle) has branch-width at most 3, and provides the decomposition if so. For k on the input, it follows from the graph case and Theorem 3.6 that testing whether the branch-width of a matroid is at most k, is NP-hard. At this moment we, moreover, do not know about any FPT algorithm for exact matroid branch-width. Currently the best known general result in this direction is:

Theorem 3.7 (Oum and Seymour [OS05])**.** *Given an n-element matroid M via an independence oracle, it is possible to test whether* $\mathrm{bwd}(M) \leq k$ *in polynomial time for each fixed k (exponent depending on k).*

It is worth to mention that this theorem holds for any connectivity function.

In the special case of matroids that are representable by vectors over finite fields, an efficient parametrized algorithm exists.

Theorem 3.8 (Hliněný [Hli05])**.** *Let $\mathbb{F}$ be a finite field, and an n-element matroid M be given via a vector representation over $\mathbb{F}$. There is an FPT algorithm testing in time $O(n^3)$ whether the branch-width of M is at most k for fixed k.*

If so, then the algorithm also outputs a branch-decomposition of M of width at most $3k$.

The algorithm is mainly of theoretical interest due to its complexity and hidden constants, but it has a remarkable application to computing graph rank-width in Chapter 4.

MSO properties of matroids. Let us consider a matroid as a structure formed by the elements and a relation for independent subsets. Applying the language of MSO logic (Section 2.1) to such matroid structures gives the *monadic second-order logic MS_M of matroids*:

The syntax of MS_M includes variables for matroid elements and element sets, the quantifiers $\forall, \exists$ applicable to those, the logical connectives $\wedge, \vee, \neg$, and the predicates:

1. $=$, the equality for elements and their sets,
2. $e \in F$, where e is an element and F is an element set variables,
3. $\mathit{indep}(F)$, where F is an element set variable, and the predicate is true if and only if F is an independent set in the matroid.

To give readers a better feeling for the expressive power of the MS_M language, we show a few additional basic matroid predicates.

- We write $\mathit{basis}(B) \equiv \mathit{indep}(B) \wedge \forall D\big(B \not\subseteq D \vee B = D \vee \neg\, \mathit{indep}(D)\big)$ to express the fact that a basis is a maximal independent set.
- Similarly, we write $\mathit{circuit}(C) \equiv \neg\, \mathit{indep}(C) \wedge \forall D\big(D \not\subseteq C \vee D = C \vee \mathit{indep}(D)\big)$, saying that C is dependent, but all proper subsets of C are independent.

It is, of course, possible to define an MSO theory of matroids using any one of $\mathit{indep}, \mathit{basis}, \mathit{circuit}$ as the atomic predicate, and to express the other two predicates similarly as above. We remark that, concerning its expressive power, the language of MS_M is related to MS_2 language of graphs.

The main algorithmic result in [Hli06a] reads (cf. Theorem 2.3):

Theorem 3.9 (Hliněný [Hli06a])**.** *Let $\mathbb{F}$ be a finite field, and ϕ be a sentence in MS_M. Suppose an n-element matroid M is given via a vector representation over $\mathbb{F}$ together with a branch-decomposition (parse tree) of bounded width. Then there is an algorithm testing in linear FPT time whether $M \models \phi$.*

Using also Theorem 3.8, we get a general tool for efficient parametrized testing of all MSO-definable properties of matroids of bounded branch-width which are represented over finite fields. Since the minor relation is expressible in MS_M [Hli03], this result and [GGW02], in particular, imply FPT solvability of all minor-closed properties on such matroids.

Some hardness results. Altogether, there are many nice algorithmic properties of graph branch-width that generalize to matroids, at least to those representable over finite fields. It appears, however, that the branch-width is not as universally applicable parameter on matroids as it is on graphs. There are several interesting problems which are easy on graphs of bounded branch-width,

but they are NP-hard on matroids of branch-width 3 represented by vectors over the rational numbers: The Tutte polynomial, some MSO-definable properties [Hli06c], or minor testing [Hli06b]. A more involved discussion of this topic can be found in [HS06, Section 8].

4 Rank-Width and Clique-Width

Clique-width is a graph "width" parameter, defined by Courcelle and Olariu [CO00]. Informally speaking, clique-width measures the complexity of describing cutsets when a graph is obtained by gluing its subgraphs recursively. Clique-width is defined by so-called k-expressions, which are algebraic expressions on labeled graphs with four operations. The k-expressions express a graph by describing how to glue two induced subgraphs. Clique-width has been attracting researchers mainly because it provides another unified framework to solve generally NP-hard graph problems in polynomial time when restricting inputs.

How hard is it to determine clique-width of graphs? Fellows, Rosamond, Rotics, and Szeider [FRRS05b] reported that determining clique-width is NP-complete. It is, however, open whether, for fixed $k \geq 4$, there is a polynomial-time algorithm to decide that clique-width of graphs is at most k. Motivated by clique-width, Oum and Seymour [OS06] defined rank-width as branch-width of the cut-rank function. They showed that a graph class has bounded clique-width if and only if it has bounded rank-width. Due to virtues of branch-width described in the previous chapter, rank-width has a few advantages over clique-width. For instance, it is in P to decide that a graph has rank-width at most k for fixed k, as shown by Oum and Seymour [OS05].

Most papers on clique-width mainly focus the following research directions.

(i) Classify graph classes having bounded or unbounded clique-width.

(ii) Identify graph problems that can be answered in polynomial time if the input graph has clique-width at most k for fixed k. Like tree-width, many graph problems have been shown to be solvable in polynomial time if the input graph has bounded clique-width.

In the following sections, we will first review definitions of clique-width and rank-width and their properties, discuss (i) and (ii), and then discuss relations to other parameters. In this chapter, all graphs are assumed to have no loops, no parallel edges, and at least one vertex, unless noted otherwise.

4.1 Definitions

Clique-width. Clique-width is defined for both directed and undirected graphs [CO00], but we will mainly focus on undirected graphs. Let k be a positive integer. We call (G, lab) a *k-graph* if G is a graph and $\mathrm{lab} : V(G) \to \{1, 2, \ldots, k\}$ is a mapping. We call $\mathrm{lab}(v)$ be the *label* of a vertex v.

(1) For $i \in \{1, \ldots, k\}$, let $\bullet_i$ denote an isolated vertex labeled by i.

(2) For $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j$, we define a unary operator $\eta_{i,j}$ such that

$$\eta_{i,j}(G, \mathrm{lab}) = (G', \mathrm{lab})$$

where $V(G') = V(G)$, and $E(G') = E(G) \cup \{vw : v, w \in V, \mathrm{lab}(v) = i, \mathrm{lab}(w) = j\}$. This adds all edges between vertices of label i and vertices of label j.

(3) We let $\rho_{i\to j}$ be the unary operator such that

$$\rho_{i\to j}(G, \mathrm{lab}) = (G, \mathrm{lab}')$$

where

$$\mathrm{lab}'(v) = \begin{cases} j & \text{if } \mathrm{lab}(v) = i, \\ \mathrm{lab}(v) & \text{otherwise.} \end{cases}$$

This mapping relabels every vertex labeled by i into j.

(4) Finally, $\oplus$ is a binary operation that makes the disjoint union. Note that $G \oplus G \neq G$.

A *k-expression* is a well-formed expression t written with these symbols. The k-graph produced by performing these operations in order therefore has vertex set the set of occurrences of the constant symbols in t; and this k-graph (and any k-graph isomorphic to it) is called the *value* $\mathrm{val}(t)$ of t. If a k-expression t has value (G, lab), we say that t is a *k-expression of G*. The *clique-width* of a graph G, denoted by $\mathrm{cwd}(G)$, is the minimum k such that there is a k-expression of G. For directed graphs, we simply replace, in the definition of a k-expression, $\eta_{i,j}$ by $\eta_{i\to j}$ that creates directed edges from vertices of label i to vertices of label j.

For instance, $\eta_{1,2}(\rho_{1\to 2}(\eta_{1,2}(\rho_{1\to 2}(\eta_{1,2}(\bullet_1 \oplus \bullet_2)) \oplus \bullet_2)) \oplus \bullet_2)$ is a 2-expression of K_4, the complete graph with four vertices. It is clear that the clique-width of K_n is 2 if $n \geq 2$.

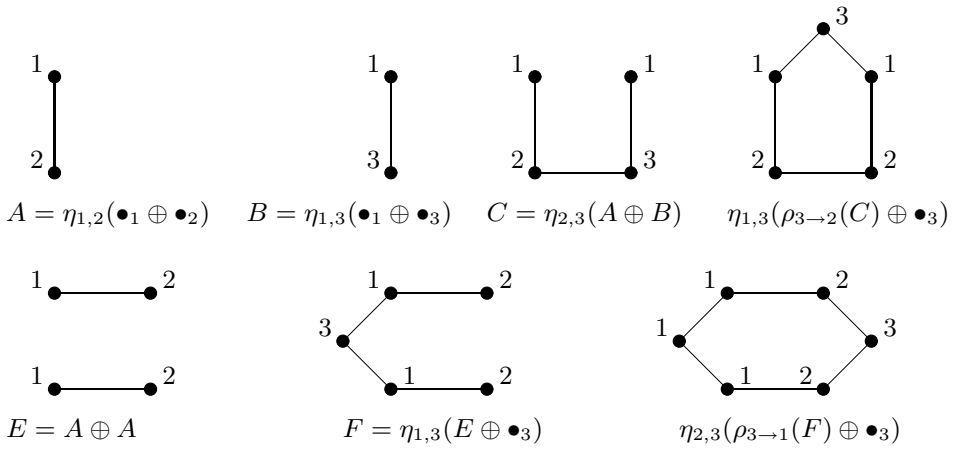


Figure 16: 3-expressions of C_5 and C_6

Rank-width. Oum and Seymour [OS06] defined rank-width of undirected graphs. We recall that branch-width is defined for arbitrary symmetric submodular functions (Section 3.1). We will define the *cut-rank* function of graphs, which is symmetric submodular, and then use it to define rank-width as the branch-width of the cut-rank function.

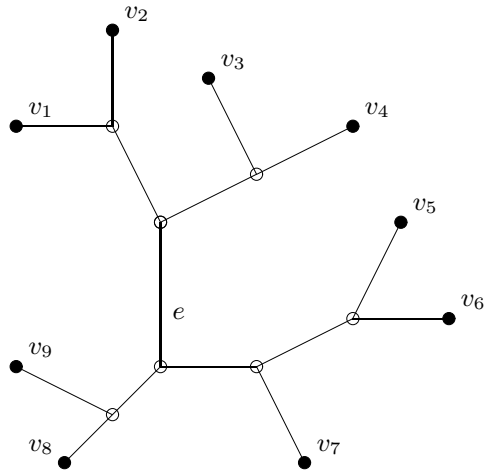
To describe the cut-rank function, we need a few notations. Let us denote $A(G)$ for the adjacency matrix of a graph G, that is a 0-1 $V(G) \times V(G)$ matrix where an entry is 1 if the column vertex is adjacent to the row vertex. We assume that the underlying field of $A(G)$ is GF(2), the field with just two elements, 0 and 1. For an $R \times C$ matrix $M = (m_{ij})_{i \in R, j \in C}$ and subsets $X \subseteq R$, $Y \subseteq C$, we denote by $M[X, Y]$ the $X \times Y$ submatrix $(m_{ij})_{i \in X, j \in Y}$.

The cut-rank function of a graph G is defined as a function $\rho_G : 2^{V(G)} \to \mathbb{Z}$ such that

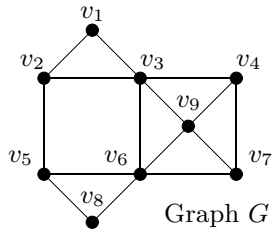
$$\rho_G(X) = \text{rank}(A(G)[X, V(G) \setminus X]),$$

where rank() is the linear rank function of matrices over GF(2). *Rank-decomposition* and *rank-width* of a graph G are defined as branch-decomposition and branch-width of the cut-rank function ρ_G of G, respectively. We denote $\text{rwd}(G)$ for the rank-width of a graph G. Rank-width is specifically designed to be strongly related to clique-width as follows [OS06]:

Theorem 4.1 (Oum and Seymour [OS06])**.** $\text{rwd}(G) \leq \text{cwd}(G) \leq 2^{1+\text{rwd}(G)} - 1$.



Rank-decomposition $(T, \mathcal{L})$ of G



Graph G

Width of e =

$$= \rho_G(\{v_5, v_6, v_7, v_8, v_9\}) =$$

$$= \text{rank} \begin{array}{c} \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array} = 3$$

Figure 17: Illustration of Rank-decompositions

Remark: other related parameters. Wanke [Wan94] defined an *NLC-width* of graphs. NLC-width is defined in a similar way as clique-width; it uses one joining operation instead of η and $\oplus$ in the k-expressions. A nice property of NLC-width is that a graph G and its complement $\overline{G}$ have the same NLC-width. Johansson [Joh98, Joh01] showed that NLC-width of a graph G is at least $\text{cwd}(G)$ but at most $2\,\text{cwd}(G)$.

Courcelle [Cou04] defined *symmetric clique-width*. For a subset X of vertices and $v, w \in X$, we define an equivalence relation $\sim_X$ such that $v \sim_X w$ if every vertex out of X is either adjacent to both v and w or to none of v and w. Let $i_G(X)$ be the number of equivalence classes of $\sim_X$. In the definition of rank-width, we replace the cut-rank functions by $\max(i_G(X), i_G(Y))$ to obtain the definition of symmetric clique-width. Courcelle [Cou04] showed that if clique-width is k, then symmetric clique-width is at most 2^k and at least $\frac{1}{2}k$.

Sequential clique-width [FRRS05a] (or *linear clique-width* [GW05b]) is analogous to path-width in relation with tree-width; in sequential clique-width, we only consider k-expressions of which each $\oplus$ operation has an operand with exactly one vertex. Similarly *linear NLC-width* and *linear symmetric clique-width* are discussed in [GW05b] and [Cou04], respectively.

4.2 Properties

Graph operations. There are many graph operations that do not change clique-width or rank-width much. We cover some of them.

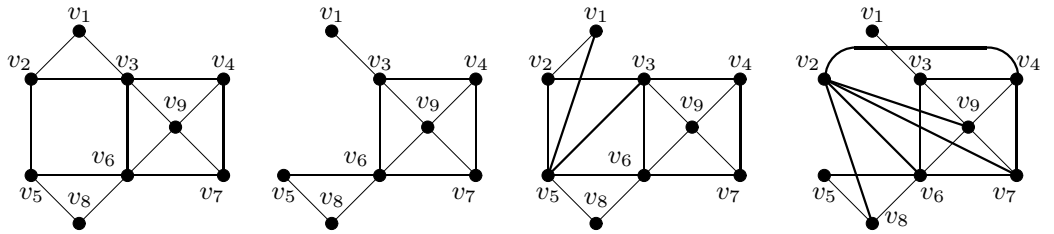


Figure 18: Deletion, local complementation, and Seidel switching

Deleting a vertex v of a graph $G = (V, E)$ is an operation to obtain a graph $G \setminus v = (V \setminus \{v\}, E')$ simply by removing v and all its incident edges from G. It is easy to show that $\mathrm{cwd}(G \setminus v) \leq \mathrm{cwd}(G)$ and $\mathrm{rwd}(G \setminus v) \leq \mathrm{rwd}(G)$. Moreover $\mathrm{rwd}(G \setminus v) \geq \mathrm{rwd}(G) - 1$. By the similar argument, one can easily show that adding or deleting k edges may increase or decrease rank-width by at most k.

The *complement* of a graph $G = (V, E)$ is a graph $\overline{G} = (V, E')$ such that two vertices in $\overline{G}$ are adjacent if they are not adjacent in G. Courcelle and Olariu [CO00] showed that $\mathrm{cwd}(\overline{G}) \leq 2\,\mathrm{cwd}(G)$. It is easy to observe that $\mathrm{rwd}(G) - 1 \leq \mathrm{rwd}(\overline{G}) \leq \mathrm{rwd}(G) + 1$, because for $\overline{G}$ we can take the rank-decomposition of G and then the width may increase by at most 1.

The *local complementation* at a vertex v of a graph G is the operation on G to add an edge xy or remove it if it already exists for every pair x, y of neighbors of the vertex v. We write $G * v$ for the graph obtained by local complementation at v to G. It is easy to observe that local complementation preserves cut-rank functions [Bou89] and therefore $\mathrm{rwd}(G * v) = \mathrm{rwd}(G)$ [Oum05c]. For an edge uv of G, the graph obtained by *pivoting* uv at G is $G \wedge uv = G * v * u * v$. A pivoting is well-defined because $G * u * v * u = G * v * u * v$ if u and v are adjacent in G. Pivoting does not change rank-width at all.

For a subset X of a vertices of a graph $G = (V, E)$, the *Seidel switching* of G by X is the operation to construct a graph $G^X = (V, E')$ from G by removing all edges between X and $V \setminus X$ and adding edges vw for all $v \in X$, $w \in V \setminus X$ if v and w is not adjacent in G. This operation was introduced by van Lint and Seidel [vLS66]. Note that $(G^X)^Y = G^{(X \setminus Y) \cup (Y \setminus X)}$ and $(G^X)^X = G$. It is straightforward to prove that $\mathrm{rwd}(G) - 1 \leq \mathrm{rwd}(G^X) \leq \mathrm{rwd}(G) + 1$.

Graph relations. An *induced subgraph* of a graph G is a graph obtained from G by deleting some vertices. As we have seen already, deleting a vertex does not

increase clique-width and rank-width and therefore for every induced subgraph H of G, we have $\mathrm{cwd}(H) \leq \mathrm{cwd}(G)$ [CO00] and $\mathrm{rwd}(H) \leq \mathrm{rwd}(G)$.

We say that H is a *vertex-minor* of G if H is obtained by applying a sequence of local complementations and vertex deletions. Similarly H is a *pivot-minor* of G if H is obtained by applying a sequence of pivoting and vertex-deletions. It is straightforward to see that if H is a vertex-minor or a pivot-minor of G, then the rank-width of H is at most the rank-width of G.

Compositions. Let G and H be a graph. From the definition of clique-width, the disjoint union $G \oplus H$ of two graphs G and H has clique-width $\max(\mathrm{cwd}(G), \mathrm{cwd}(H))$. Similarly $\mathrm{rwd}(G \oplus H) = \max(\mathrm{rwd}(G), \mathrm{rwd}(H))$. Therefore, clique-width and rank-width of a graph G is equal to the maximum of clique-width and rank-width, respectively, of its connected components.

For $v \in V(G)$ and $w \in V(H)$, the 1-*sum* of G and H at v and w is the operation to create a graph $G \oplus_{v=w} H$ from $G \oplus H$ by identifying v and w. Then

$$\mathrm{rwd}(G \oplus_{v=w} H) = \max(\mathrm{rwd}(G), \mathrm{rwd}(H)). \tag{1}$$

In particular, the rank-width of a graph is equal to the maximum of rank-width of its blocks (maximal connected subgraphs without a cutvertex). To prove (1), we do 1-sum of two trees in rank-decompositions of G and H at leaves representing v and w, and then create a new leaf at the identified vertex in the tree representing the identified vertex of $G \oplus_{v=w} H$. Notice that $\max(\mathrm{rwd}(G), \mathrm{rwd}(H)) \leq \mathrm{rwd}(G \oplus_{v=w} H)$ because both G and H are induced subgraphs of $G \oplus_{v=w} H$.

For $v \in V(G)$, the *substitution* of v in G by H is an operation to create a graph $G[H \setminus v]$ by substituting the vertex v by H. This means that in $G[H \setminus v]$, every vertex from H is adjacent to its neighbors in H as well as neighbors of v in G. Courcelle and Olariu [CO00] showed that

$$\mathrm{cwd}(G[H \setminus v]) = \max(\mathrm{cwd}(G), \mathrm{cwd}(H)). \tag{2}$$

By a similar argument to the 1-sum, one can show that $\max(\mathrm{rwd}(G), \mathrm{rwd}(H)) \leq \mathrm{rwd}(G[H \setminus v]) \leq \max(\mathrm{rwd}(G), \mathrm{rwd}(H) + 1)$. A *modular decomposition* is a tree representation of obtaining a graph recursively by substitution. A *prime* graph with respect to modular decomposition is a graph that can not be obtained by nontrivial substitution. McConnell and Spinrad [MS99] gave a linear-time algorithm for a modular decomposition. From (2), we deduce the following:

Theorem 4.2 (Courcelle and Olariu [CO00])**.** *The clique-width of a graph G is equal to the maximum of the clique-width of all prime (with respect to modular decomposition) induced subgraphs.*

In fact, it is enough to consider prime subgraphs appearing in the modular decomposition. In particular, in order to show that a certain set of graphs has bounded clique-width, it is enough to prove that all prime graphs in the set have bounded clique-width. This technique has been used in several papers, for instance [BLM05].

For $v \in V(G)$ and $w \in V(H)$, the 1-*join* of G and H with v and w is the operation to create a graph from $G \oplus H$ by adding all edges from the neighbors

of v to the neighbors of w and then deleting v and w. It is an easy exercise to show that

$$\mathrm{rwd}(1\text{-join}_{v,w}(G,H)) = \max(\mathrm{rwd}(G), \mathrm{rwd}(H)). \tag{3}$$

(The proof is similar to that of 1-sum.) The *split decomposition* [Cun82] is a tree representation of obtaining a graph recursively by 1-joins. A graph that can not be obtained by a nontrivial 1-join is called a *prime* graph with respect to split decomposition. (Be warned that primeness may refer to several decompositions.) Ma and Spinrad [MS94] showed an $O(n^2)$-time algorithm to find a split decomposition of an n-vertex graph. From (3), we deduce the following.

Theorem 4.3. *The rank-width of a graph is the maximum of the rank-width of all prime induced subgraphs with respect to split decomposition.*

Two special cases of 1-sum and substitution are interesting. The 1-sum of G and K_2 creates a *pendant* vertex to G. The substitution of a vertex in G by K_2 or P_2 creates a *twin* to G. Creating pendant vertices or twins does not change rank-width of G if G has at least one edge [Oum05c].

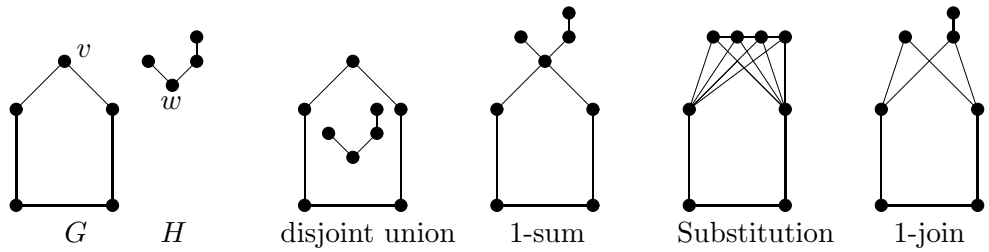


Figure 19: Compositions

Upper bounds. It is easy to see that $\mathrm{cwd}(G) \leq |V(G)|$ and $\mathrm{rwd}(G) \leq \lceil |V(G)|/3 \rceil$. Johansson [Joh98] showed that if G is an n-vertex graph and $2^k + k \leq n - k$ then $\mathrm{cwd}(G) \leq n - k$.

Clique-width and interpretability in binary trees. Courcelle [Cou92b] showed the following. ($\mathrm{C_2MS_1}$ logic is an extension of $\mathrm{MS_1}$, allowing the set predicate $\mathrm{Even}(X)$ expressing that the set variable X is finite and has even cardinality. It is a restriction of CMSO logic from Section 2.3.)

Theorem 4.4 (Courcelle [Cou92b])**.** *A set of graphs has bounded clique-width if and only if it is MS_1-interpretable into a class of binary trees.*

This theorem is a quick and powerful tool to prove that a set of graphs has bounded clique-width. For instance, if we want to show that another set B of graphs has bounded clique-width, then we find a set A of graphs having bounded clique-width and show that B is $\mathrm{MS_1}$-interpretable into A. (In the terminology of Courcelle, B is the image of a monadic second order transduction (MS transduction) of A.) [CO00, Cou06, CO06] are some examples of applying this theorem.

Connections to other parameters. Graphs of bounded tree-width have bounded clique-width but not vice versa. This was first shown by Courcelle and Olariu [CO00] and the following refinement was shown by Corneil and Rotics [CR05]. (Let $\mathrm{twd}(G)$ be the tree-width of G.)

Theorem 4.5 (Corneil and Rotics [CR05])**.** $\mathrm{cwd}(G) \leq 3(2^{\mathrm{twd}(G)-1})$.

They also showed that for any k, there is a graph G with tree-width k such that $\mathrm{cwd}(G) \geq 2^{\lfloor k/2 \rfloor - 1}$. We remark that no inequalities of the form $\mathrm{twd}(G) \leq f(\mathrm{cwd}(G))$ is possible, because $\mathrm{cwd}(K_n) = 2$ and $\mathrm{twd}(K_n) = n - 1$. However, Gurski and Wanke [GW00] showed the following. (The implicit existence of such an inequality was shown before by Courcelle and Olariu [CO00].)

Theorem 4.6 (Gurski and Wanke [GW00])**.** *If a graph G has no $K_{t,t}$ subgraph for some $t > 1$, then* $\mathrm{twd}(G) \leq 3(t-1)\,\mathrm{cwd}(G) - 1$.

We write $L(G)$ for the line graph of G, whose vertices are the edges of G, and two vertices of $L(G)$ are adjacent if they are adjacent in G. Gurski and Wanke [GW04, GW05a] showed that

$$\frac{1}{4}(\mathrm{twd}(G) + 1) \leq \mathrm{cwd}(L(G)) \leq 2\,\mathrm{twd}(G) + 2.$$

Branch-width of matroids is closely related to rank-width. We will first define the *fundamental graph* of a binary matroid $\mathcal{M}$. Let B be a base of $\mathcal{M}$. The *fundamental graph* of $\mathcal{M}$ with respect to B, denoted by $F(\mathcal{M}, B)$, is a bipartite graph on vertices $E(\mathcal{M})$ where two vertices $e, f \in E(\mathcal{M})$ are adjacent if $e \in B$, $f \in E(\mathcal{M}) \setminus B$, and $(B \setminus \{e\}) \cup \{f\}$ is independent in $\mathcal{M}$. Oum [Oum05c] showed that, for every binary matroid $\mathcal{M}$ and its base B, we have

$$\mathrm{bwd}(\mathcal{M}) = \mathrm{rwd}(F(\mathcal{M}, B)) + 1,$$

which means that the branch-width $\mathrm{bwd}(\mathcal{M})$ of a binary matroid $\mathcal{M}$ is exactly one more than the rank-width of a fundamental graph of $\mathcal{M}$.

4.3 Clique-width of graph classes

For a given set of graphs, it is interesting to know whether there is a finite upper bound on the clique-width of a graph in the set; if there is such an upper bound, then many graph problems are solvable quickly, which we will discuss in the next section.

Basic graph classes. The most basic classes of graphs are graphs of clique-width at most 2 and graphs of rank-width at most 1. Note that clique-width is 1 or rank-width is 0 if and only if the graph has no edges.

A *cograph* (or a *complement reducible graph*) is defined recursively as follows: the graph with a single vertex, the disjoint union of two cographs, and the complement of a cograph are cographs. Equivalently, cographs are *P_4-free graphs*, graphs with no induced P_4 (a path of four vertices) [CLB81]. It is easy to observe, from the definition, that cographs are exactly graphs of clique-width at most 2.

A graph G is called *distance-hereditary* if, in every connected induced subgraph of G, the distance between every pair of vertices in the subgraph is equal

to the distance in G. For example, C_5 is not distance-hereditary. Bandelt and Mulder [BM86] characterized distance-hereditary graphs; their characterization implies that distance-hereditary graphs are exactly graphs of rank-width at most 1 [Oum05c]. By Theorem 4.1, we deduce that clique-width of a distance-hereditary graph is at most 3, which was shown first by Golumbic and Rotics [GR00].

For the cycle graph C_n of n vertices,

$$\mathrm{cwd}(C_n) = \begin{cases} 2 & \text{if } n = 3, 4, \\ 3 & \text{if } n = 5, 6, \\ 4 & \text{if } n \geq 7, \end{cases} \qquad \mathrm{rwd}(C_n) = \begin{cases} 1 & \text{if } n = 3, 4, \\ 2 & \text{if } n \geq 5. \end{cases}$$

Golumbic and Rotics [GR00] showed that clique-width of the $n \times n$ grid (having n^2 vertices) is exactly $n + 1$ if $n \geq 3$. Rank-width of the $n \times n$ grid is at most $n - 1$ (Oum [unpublished]), but the exact value is not yet known.

Unbounded clique-width. To verify that a certain class of graphs has unbounded clique-width, grids are sometimes useful because if the class has unbounded clique-width if it contains infinitely many grids. For instance, the set of bipartite C_6-free graph has unbounded clique-width because it contains infinitely many grids. Moreover, any sequence $H_1, H_2, \ldots$ of graphs such that $\lim_{n\to\infty} \mathrm{cwd}(H_n) = \infty$ will help us to verify that a certain graph class has unbounded clique-width. Here is a list of some papers [MR99, GR00, BL02, BELL06] having such sequences of graphs.

Boliac and Lozin [BL02] counted the number of n-vertex graphs of bounded clique-width:

Theorem 4.7 (Boliac and Lozin [BL02])**.** *For fixed $k > 1$, the number of graphs having n vertices and clique-width at most k is $2^{\Theta(n \log n)}$.*

Consequently, bipartite graphs, co-bipartite graphs (complement of bipartite graphs), and split graphs (graphs that can be partitioned into an independent set and a clique) have unbounded clique-width, because the number of n-vertex graphs in each of them is at least $2^{n^2/4}$.

Clique-width and forbidden induced subgraphs. Cographs can be defined as P_4-free graphs and as we discussed earlier, their clique-width is at most two. Distance-hereditary graphs can be also defined by forbidding four small graphs [BM86] and have rank-width at most one and clique-width at most three.

What else can we say if we exclude other small graphs? Brandstädt, Engelfriet, Le, and Lozin [BELL06] answered this question completely up to four vertices, characterizing all graph classes having bounded clique-width defined by forbidding small graphs up to four vertices as an induced subgraph. Brandstädt, Dragan, Le, and Mosca [BDLM05] answered the question when excluding 5-vertex graphs extending P_4.

There are variations of cographs, such as P_4-reducible graphs [JO89], P_4-sparse graphs [Hoà85], P_4-tidy [GRT97], partner-limited graphs [RRT99], and (q, t) graphs [BO95]. Upper bounds of clique-width for some of those graph classes have been obtained in [CMR00, MR99, Van04]

There are miscellaneous graph classes defined by forbidding small induced subgraphs that are shown to have either bounded or unbounded clique-width. We refer readers to the Information System on Graph Class Inclusions (ISGCI) of University of Rostock at `http://wwwteo.informatik.uni-rostock.de/isgci/` which has references on clique-width for most interesting graph classes.

Rank-width and forbidden vertex-minors. Graphs of rank-width at most 1 are exactly distance-hereditary graphs [Oum05c], and they are the graphs with no vertex-minors isomorphic to C_5 [Bou87, Bou88]. Since vertex-minors of a graph of rank-width k have rank-width at most k, it is possible to describe the set of graphs of rank-width at most k by a list of forbidden vertex-minors, such that a graph has rank-width at most k if and only if none of its vertex-minors is isomorphic to one of the forbidden vertex-minors. What is interesting is that this list is always finite. Oum [Oum05c] has shown the following:

Theorem 4.8 (Oum [Oum05c])**.** *For fixed $k \geq 1$, the set of graphs having rank-width at most k is characterized by forbidden vertex-minors with at most $(6^{k+1} - 1)/5$ vertices.*

In another paper [Oum05d], he has proved a more general theorem; graphs of bounded clique-width are *well-quasi-ordered* by pivot-minors which means the following:

Theorem 4.9 (Oum [Oum05d])**.** *For every infinite sequence $G_1, G_2, \ldots$ of graphs having bounded clique-width, there are $i < j$ such that G_i is isomorphic to a pivot-minor of G_j.*

This theorem implies that the list of forbidden vertex-minors characterizing graphs of rank-width at most k is finite for each k. As an example, let us consider cycles having at least t vertices. Clearly C_t is isomorphic to a vertex-minor of C_{t+1} and therefore cycles C_t, C_{t+1}, C_{t+2}, ... are well-quasi-ordered by vertex-minors. We can deduce the same conclusion from the above theorem since they have rank-width at most 2. Cycles are, however, not well-quasi-ordered by the induced subgraph relation; no proper induced subgraph of a cycle is a cycle.

4.4 Algorithms on graphs of bounded clique-width

Major interest in clique-width is due to the fact that many NP-hard problems can be solved in polynomial time if the input is restricted to graphs of bounded clique-width. Not only there is an irregular list of solvable problems on graphs of bounded clique-width, but also there is a general theorem stating that all graph problems expressible in monadic second-order logic are solvable. Tree-width surely has an analogous property, but every set of graphs of bounded tree-width has bounded clique-width and in this sense clique-width is more powerful. The trade-off here is that the set of problems for which we know an efficient solution on graphs of bounded tree-width is (at present) significantly larger than that for bounded clique-width.

To solve those graph problems, most algorithms based on bounded clique-width use dynamic programming with the k-expression of the input graph. To provide the k-expression of the input graph to the main routine using dynamic programming, a necessary step is to generate a k-expression from the input graph given in the form of the adjacency list.

Solvable problems on graphs of bounded clique-width. Courcelle, Makowsky, and Rotics [CMR00] showed the following theorem that can be obtained from Theorem 4.4 and Theorem 2.3. (Compare this with Theorems 2.5 and 2.6 which have dealt with MS_2 and tree-width.)

Theorem 4.10 (Courcelle, Makowsky, and Rotics [CMR00])**.** *Every MS_1 decision problem (a graph problem expressible in MS_1 logic) is solvable in linear time for graphs having clique-width at most k for fixed k if the k-expression defining the graph is given with the input.*

We will describe how to find a k-expression later. In their next paper [CMR01], they showed that MS_1 counting problems, enumerating the truth assignments of a fixed MS_1 logic formula, are solvable in linear time as well under the same condition.

Not only MS_1 problems, but also many other graph problems are solvable. Here is a partial list of papers [Wan94, EGW01, KR03, Tod03, Joh03]. Except [Joh03], all known algorithms use dynamic programming with the given k-expression and thus require the k-expression of the input graph to be given as an input. (Johnson [Joh03] used recursive programs.) For instance, testing whether a graph is Hamiltonian is solvable in polynomial time for graphs of clique-width at most k [Wan94]. It is also possible to output the chromatic number of the input graph in polynomial time if the input graph has clique-width at most k for fixed k [KR03]. However, these are not fixed-parameter tractable algorithms.

Not all graph problems can be solved easily on graphs of bounded clique-width. Gurski and Wanke [GW04] found a graph problem that is solvable in linear time on graphs of clique-width at most 2 but NP-complete on graphs of clique-width at most 7.

Determining clique-width and rank-width. Fellows, Rosamond, Rotics, and Szeider [FRRS05b] reported that determining clique-width is NP-complete: they showed that if a graph G and an integer k are given as an input, it is NP-complete to decide whether the clique-width of G is at most k. In addition, they proved the following: unless $P = NP$, for each $0 < \epsilon < 1$, there is no polynomial-time algorithm that, with an n-vertex graph, outputs a $(\mathrm{cwd}(G)+n^{\epsilon})$-expression of the input graph.

The situation is different if we fix k and wish to know whether clique-width or rank-width is at most k. We summarize known results in Table 1.

k	clique-width$\leq k$	rank-width $\leq k$
1	Graphs with no edges. Trivial.	Distance-hereditary graphs [Oum05c]. Linear [HM90, DHP01]
2	Cographs. Linear [CPS85]	[Oum05a, CO06, Oum05b] $O(n^3)$
3	[CHL$^+$00] $O(n^2m)$	
≥ 4	Open	

Table 1: Fixed parameter algorithms to test clique-width and rank-width

Even if we are able to determine $\mathrm{cwd}(G) \leq k$ or $\mathrm{rwd}(G) \leq k$, we do not necessarily get the k-expression or rank-decomposition of width at most k. In particular, the algorithm in [CO06] does not provide rank-decomposition of

width at most k even if it confirms that $\mathrm{rwd}(G) \leq k$, because it tries to find a forbidden vertex-minor to test rank-width. The result of Oum and Seymour [OS05] (see Theorem 3.7) implies that there is a polynomial-time algorithm to construct a rank-decomposition of width at most k if $\mathrm{rwd}(G) \leq k$. Their algorithm is, however, not a fixed parameter algorithm.

If we assume that an input graph has bounded tree-width, then the task becomes easy. Espelage, Gurski, and Wanke [EGW03] showed a linear-time algorithm for deciding whether a graph of bounded tree-width has clique-width at most k for some fixed k. Since $\mathrm{cwd}(G) \leq 3 \cdot 2^{\mathrm{twd}(G)-1}$, we can determine the clique-width in polynomial time if the input graph has bounded tree-width.

5 Hypertree Width and other Hypergraph Invariants

The structure of the instances of many NP-hard problems is best described by graphs. This is either because the problem description itself contains a graph as a main element, or because a graph is easy to be associated with each instance. As already noted, many of such problems are tractable on instances whose associated graphs are acyclic or have bounded tree-width. A *hypergraph* $H = (V, E)$ consists of a finite set V of vertices and a set E of hyperedges, where a hyperedge is a subset of V. There is a number of important problems whose structure is better described by hypergraphs than by graphs. For these problems, we can obtain tractable classes by imposing restrictions on the instance hypergraphs.

5.1 CSP and other hypergraph-structured problems

A typical problem that falls into this category is the *Constraint Satisfaction Problem* (*CSP*) (cf. [DP87, Dec92, GLS01]). A CSP instance I consists of a set Var of variables, a domain D of possible values for these variables, and a set $C = \{c_1, \ldots, c_m\}$ of so-called *constraints*. Each constraint $c_i = \langle S_i, R_i \rangle$ consists of a list S_i of a_i variables from Var, called the *scope of* c_i, and a relation $R_i \subseteq Var^{a_i}$, called the *constraint relation* of c_i. The number a_i of variables in S_i, which corresponds to the number of columns of R_i is referred to as the *arity* of S_i and of R_i. A solution for instance I is a mapping $h : Var \longrightarrow D$, such that for each $1 \leq i \leq m$, $h(S_i) \in c_i$. If a solution exists, then the instance is *solvable*. We want to decide whether an input CSP instance is solvable. Note that the arities $a_1, \ldots, a_m$ of the scopes and constraint relations can be unbounded in general.

Many important problems equivalent to CSPs have been identified in the literature (see [KV00, GLS01] for examples and references). Among these are the problem of answering Boolean conjunctive queries [CM77], which is a central database problem, the problem of clause subsumption in theorem proving, and the problem of checking whether there exists a homomorphism between two finite structures. These problems, which belong to different domains, can actually be identified with the CSP.

To each instance I of a CSP, we can associate a hypergraph $H(I) = (V, H)$ whose set V of vertices coincides with Var, and whose set H of hyperedges contains for each scope S_i of I, a hyperedge H_i consisting of the set of all variables of S_i.

5.2 CSP with acyclic hypergraphs is tractable

It is well-known that CSP is NP-complete, even in case of bounded arities. NP-hardness can be seen, for instance, by a rather trivial transformation from graph 3-colorability into CSP (left to the reader). On the other hand, it is rather easy to see [Yan81, Dec92] that CSP restricted to instances I such that $H(I)$ is acyclic is tractable. As shown in [GLS01] CSP restricted to instances with acyclic hypergraphs is even highly parallelizable and complete for the low complexity class LOGCFL, in other words, the class of all decision problems that can be transformed by a logarithmic space reduction to a context-free language. Here, by acyclicity, we refer to the most general (most liberal) notion of hypergraph acyclicity studied in the literature, namely, α-acyclicity (cf [Fag83]).

A hypergraph H is *acyclic* if the repeated application of the following operations (in any applicable order) will eventually lead to the empty hypergraph:

- If H contains a vertex v that is isolated or contained in a single hyperedge, then eliminate v from H.
- If H contains a hyperedge E that is a subset of another hyperedge E' of H, then eliminate E from H.
- If H contains the empty hyperedge, then eliminate it from H.

Since α-acyclicity is the most liberal notion of acyclicity, the same favorable complexity results are easily seen to hold in the same way for classes of CSPs associated to acyclic hypergraphs according to other notions of hypergraph acyclicity [Fag83].

Note that there are several different ways to associate a graph to a hypergraph H, and thus also to a CSP instance:

- The *primal graph* $G_p(H)$ of H (sometimes also referred to as the *Gaifman graph* of H) is a graph whose vertices are the nodes of H and such that two vertices a and b are adjacent in $G_p(H)$ when there is a hyperedge of H containing both a and b.
- The bipartite *incidence graph* $G_{in}(H)$ of H is a graph, whose vertices consist of both the vertices of H and the edges of H, such that a vertex a and an edge e is adjacent in $G_{in}(H)$ if $a \in e$.
- The *dual graph* $G_d(H)$ is a graph whose vertices are the hyperedges of H, such that two hyperedges e_1 and e_2 are adjacent in $G_d(H)$ if they intersect.

For a CSP instance I, we denote by $G_p(I)$, $G_{in}(I)$, and $G_d(I)$ the graphs $G_p(H(I))$, $G_{in}(H(I))$, and $G_d(H(I))$, respectively.

It is very easy to find a class $\mathcal{H}$ of acyclic hypergraphs such that for each k there exists a hypergraph $H \in \mathcal{H}$ such that all of $G_p(H)$, $G_{in}(H)$, and $G_d(H)$ have tree-width larger than k in other words, all three graphs associated with $\mathcal{H}$ have unbounded tree-width. This means that there is something genuine in the notion of hypergraph acyclicity that cannot be captured via graph acyclicity or bounded graph tree-width. Another way to see that hypergraph acyclicity is genuinely different from graph acyclicity is to notice that unlike graph acyclicity, hypergraph acyclicity is not preserved under hyperedge deletion. For example, each hypergraph that contains a hyperedge covering *all* vertices is acyclic. Obviously, when deleting this edge, the resulting hypergraph is not necessarily acyclic.

5.3 Early Width parameters for hypergraphs

It has been observed by many researchers that a large number of practical problem instances of CSP and related hypergraph-based problems have corresponding hypergraphs that are in some sense "nearly acyclic". Thus, researchers have been searching for a parameter akin graph tree-width that would appropriately express the degree of cyclicity of a hypergraph. The following hypergraph invariants are examples for early approaches for measuring hypergraph cyclicity. These approaches were mainly considered in artificial intelligence, and, in particular, in the area of constraint processing.

Biconnected Components (BICOMP) [Fre85]. Any graph $G = (V, E)$ can be decomposed into a pair (T, χ) of a tree T and a labeling function χ associating to each vertex of T a biconnected component of G. The *biconnected width* of a hypergraph H, denoted by BICOMP-width(H), is the maximum number of vertices over the biconnected components of the primal graph $G_p(H)$ of H.

Cycle Cutset and Hypercutset (CUTSET) [Dec92]. A *cycle cutset* of a hypergraph H is a set $S \subseteq V(H)$ such that the subhypergraph of H induced by $V(H) \setminus S$ is acyclic. The CUTSET-width of H is the minimum cardinality over all its possible cycle cutsets. A generalization of this is the method is the method of hypercutsets, in short, HYPERCUTSET (for a definition, see [GLS00]).

Tree Clustering (TCLUSTER) [DP89]. The *tree-clustering* method is based on a triangulation algorithm which transforms the primal graph $G_p(H)$ of any hypergraph H into a chordal graph G'. The maximal cliques of G' are then used to build the hyperedges of an acyclic hypergraph H'. The *tree-clustering width* (TCLUSTER-*width*) of H is 1 if H is an acyclic hypergraph; otherwise it is equal to the maximum cardinality of the cliques of the chordal graph G'.

The Hinge Method (HINGE) [GP84, GJC94]. This is an interesting decomposition method generalizing acyclic hypergraphs. For space reasons, we omit a formal definition. The hinge-width of a hypergraph can be computed in polynomial time [GP84, GJC94]. One can also combine the methods HINGE and TCLUSTER, yielding a more general method $\text{HINGE}^{\text{TCLUSTER}}$.

5.4 Generalized Hypertree Width

A more recent group of hypergraph decompositions is based on the idea of covering the bags of a tree-decomposition of a hypergraph H. Before defining such decompositions, let us extend the notion of tree-decomposition from graphs to hypergraphs.

Definition. Let $H = (V, E)$ be a hypergraph on vertices V and hyperedges E. A *tree-decomposition* (T, χ) of H consists of a tree $T = (V_T, E_T)$ and a node labeling function $\chi : V_T \longrightarrow 2^V$ for T which associates to each vertex t of T a bag $\chi(t)$, such that the following three conditions hold:

1. For each hyperedge $e \in E$, there is a vertex t of T such that $e \subseteq \chi(t)$.
2. For all vertices $v \in V$, the subgraph of T induced by $\{t : v \in \chi(t)\}$ is connected. (The interpolation or connectedness property.)
3. The union of all $\chi(t)$ for all vertices t of T equals the vertex set V of H.

Example 5.1. *Figure 20 shows a hypergraph H (consisting of* 12 *hyperedges and* 13 *vertices) and a tree-decomposition of H.*

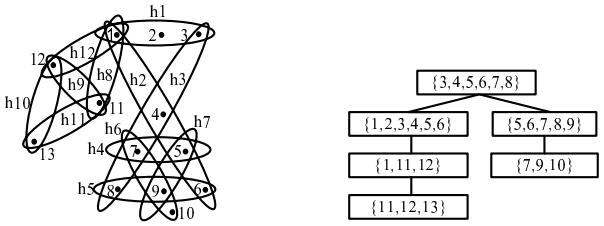


Figure 20: A hypergraph H (left) and tree-decomposition of H (right).

In general, unless stated otherwise, we assume the tree T of a tree-decomposition (T, χ) is unrooted and undirected.

Note that by this definition, a tree-decomposition of a hypergraph is defined just like a tree-decomposition of a graph, except for condition 1, which now requires that each *hyperedge* is contained in at least one bag of the decomposition. When H is a graph, the definitions coincide. Also note that the tree-width of a hypergraph H coincides with the tree-width of its primal graph $G_p(H)$.

We are now ready for defining the concept of *generalized hypertree decomposition* [GLS02].

Definition. [GLS02] Let $H = (V, E)$ be a hypergraph on vertices V and hyperedges E. A *generalized hypertree decomposition* $T = (T, \chi, \lambda)$ for H consists of a tree-decomposition (T, χ) and a node labeling function λ which associates to each vertex t of T a set of edges $\lambda(t) \subseteq E$, such that for each node t of T, the hyperedges in $\lambda(t)$ cover $\chi(t)$, in other words, $\chi(t) \subseteq \cup_{e \in \lambda(t)} e$. The *width* of such a general hypertree decomposition is defined by the maximum of all cardinalities $|\lambda(t)|$ when t ranges over all nodes of T. The generalized hypertree-width $ghw(H)$ of H is the smallest possible width of a hypertree decomposition of H.

Example 5.2. *A generalized hypertree decomposition of width* 2 *of the hypergraph H in Figure 20 is given in Figure 21. At each decomposition node, the left set within each rectangle represents the λ-labels and the right set represents the χ-labels.*

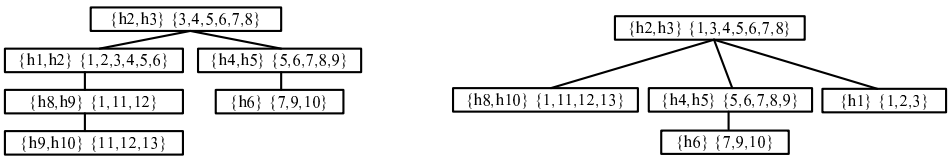


Figure 21: Generalized hypertree decomposition of hypergraph H (left) and hypertree decomposition of hypergraph H (right).

Note that for each hypergraph H, we have $ghw(H) \leq tw(H)$. Moreover, for each acyclic hypergraph H with nonempty set of hyperedges, we have $ghw(H) = 1$. Thus there are classes of hypergraphs of unbounded tree-width

whose generalized hypertree-width is 1. The "paradigm shift" in the transition from tree-width to hypertree width consists in counting the covering hyperedges rather than counting the number of vertices in a bag. This parameter seems to be more appropriate, especially with respect to constraint satisfaction problems. We have the following result, where the size of a data structure A is denoted by $\|A\|$:

Theorem 5.3 (G. Gottlob, N. Leone, and F. Scarcello [GLS02, GLS00]). *For a CSP instance I and a generalized hypertree decomposition D of width k of $H(I)$, it takes time $O(\|I\|^{k+1} \log(\|I\|) + \|D\|)$ to check whether I is solvable, and to compute a solution if so.*

Roughly, to prove this theorem, we first transform D in a normal form that eliminates potential redundancy from D, and then transform instance I in time $\|I\|^k$ into a CSP instance I' whose associated hypergraph is acyclic (by performing cartesian products or joins of the constraint relations corresponding to each node of the decomposition) and then solving the acyclic instance $\|I'\|$ by the algorithm of Yannakakis [Yan81].

Generalized hypertree width would be a very nice width parameter if for each fixed positive integer k, one could recognize in polynomial time for each hypergraph H whether $ghw(H) = k$. Unfortunately, this does not seem to be the case.

Theorem 5.4 (T. Schwentick [Sch]). *Deciding whether $ghw(H) = 4$ is NP-complete.*

5.5 Query Width

A width parameter for hypergraphs that was actually introduced before generalized hypertree-width is the notion of *query-width (qw)* [CR00]. We have defined ghw before qw in this paper, because now qw can be easily defined as a restricted version of ghw.

Definition. [CR00] A generalized hypertree decomposition (T, χ, λ) of a hypergraph H is called a query decomposition if for each vertex t of T, $\chi(t)$ is equal to the union of all edges in $\lambda(t)$. The query-width $qw(H)$ of H is the smallest width of a query decomposition of H.

Since query decompositions are special cases of generalized hypertree decompositions, it clearly holds for each hypergraph H that $ghw(H) \leq qw(H)$. There are, however, hypergraphs H for which $ghw(h) < qw(H)$ (cf. [GLS02]).

Chekuri and Rajaraman [CR00] asked whether for each constant k, $qw(H) = k$ can be decided in polynomial time. Unfortunately, this does not seem to be the case.

Theorem 5.5 ([Sch]). *Deciding whether $qw(H) = 4$ is NP-complete.*

5.6 Hypertree Width

A *rooted ghd* is a generalized hypertree decomposition (T, χ, λ) whose decomposition tree T is rooted at some vertex $p_0 = root(T)$. In case of such a rooted ghd, we conceive all edges of T as directed in the direction from the root to the leaves.

For a rooted ghd (T, χ, λ) of a hypergraph H, and a node p of the decomposition tree T, we denote by $var(\lambda(p))$ the set of all vertices that appear in some edge in $\lambda(p)$. Moreover, we denote by T_p the subtree of T rooted at p, and by $\chi(T_p)$ the union of all sets $\chi(p')$ for all vertices p' of T_p.

Definition. A *hypertree decomposition (hd)* of a hypergraph H is a rooted ghd (T, χ, λ) of H such that

$$var(\lambda(p)) \cap \chi(T_p) \subseteq \chi(p).$$

The hypertree-width of H denoted by *hw(H)* is the smallest width of a hypertree decomposition of H.

An intuitive explanation of the special condition in the above definition is given in Section 5.7, when discussing a game theoretic interpretation of hypertree-width.

Example 5.6. *The generalized hypertree decomposition of Example 5.2 violates the special condition in the root node. A hypertree decomposition for hypergraph H, equally of width 2, is given in the right part of Figure 21.*

As proven in [GLS02], hypertree-decompositions and hypertree-width enjoy the following nice properties.

Theorem 5.7 (G. Gottlob, N. Leone, and F. Scarcello [GLS02])**.** *For each fixed constant k, checking whether a hypergraph H has hypertree-width k, can be tested in polynomial time. This problem actually lies in the parallel complexity class LOGCFL. If $hw(H) = k$, then a hypertree decomposition of width k can be computed in polynomial time.*

The currently best known upper bound for checking whether, for a hypergraph $H = (V, E)$, $hw(H) = k$ is the one of the opt-k-decomp algorithm [GLS99], which runs in time $O(|E|^{2k}|V|^k)$. (The code of some hypertree decomposition algorithms is currently available on the Web [Weba, Webb].) It is currently unclear, whether this bound can be improved significantly, say, to $O(n^{k+c})$, for some constant c, where n is the size of the input hypergraph.

Recall that deciding "tw(G)=k" is fixed-parameter tractable (FPT) with respect to the parameter k, and is actually feasible in linear time for each fixed k by Bodlaender's algorithm [Bod96]. It is thus natural to ask, whether similar results hold for hypertree-width. Unfortunately, this seems to be highly unlikely:

Theorem 5.8 ([GGM$^+$05])**.** *Checking whether $hw(H) = k$ for input hypergraphs H is NP-complete and fixed-parameter intractable with respect to the parameter k. In particular, the parameterized problem is W[2]-hard. The same results hold for the problem of checking $ghw(H) = k$.*

The above fixed-parameter intractability result is not completely unexpected, given that both generalized and plain hypertree decompositions are tightly connected to set covering. In fact, one can conceive generalized hypertree-decompositions as a mixture of tree-decomposition and set covering. Now, the set covering problem is well-known to be W[2]-hard, see [DF99].

Since every hypertree decomposition (when forgetting the root of the tree T) is also a generalized hypertree decomposition, we have $ghw(H) \leq hw(H)$, and thus, by Theorems 5.7 and 5.3, we have:

Theorem 5.9 ([GLS02]). *The solvability of CSP instances whose associated hypergraphs have bounded hypertree-width is decidable in polynomial time. Moreover, for a CSP instance I and a generalized hypertree decomposition D of width k of $H(I)$, it takes time $O(\|I\|^{k+1} \log(\|I\|) + \|D\|)$ to check whether I is solvable, and to compute a solution if so.*

Hypertree-width compares favorably to all other discussed hypergraph width parameters for which "*width* $= k$" is known to be tractable.

Theorem 5.10 ([GLS00]). *For each width parameter* `X` *in* {`BICOMP`, `CUTSET`, `TCLUSTER`, `HINGE`, `HINGE`$^{\texttt{TCLUSTER}}$, `TREEWIDTH`}, *where* `TREEWIDTH` *denotes hypergraph tree-width as defined, the following holds:*

- *Each class of hypergraphs of bounded* `X`*-width has also bounded hypertree-width.*
- *There is a class of hypergraphs having bounded hypertree-width but unbounded* `X`*-width.*

Recall that a graph with n vertices may be of tree-width up to $n-1$. This naturally leads to the question about the maximum hypertree-width (or ghw, or qw) of a hypergraph with n vertices.

Theorem 5.11 (G. Gottlob [Got05]). *Let H be a hypergraph having n vertices. Then $hw(H) \leq \lfloor n/2 \rfloor + 1$, $ghw(H) \leq \lfloor n/2 \rfloor + 1$, $qw(H) \leq \lfloor n/2 \rfloor + 1$.*

Examples can be found for showing that for each n the bounds in the above theorem are attainable.

The relationship between hypertree-width, query-width, and generalized hypertree-width is more subtle. As shown in [AGG], hypertree-width and query-width approximate generalized hypertree-width by a factor of tree.

Theorem 5.12 (I. Adler, G. Gottlob, and M. Grohe [AGG]). *For each hypergraph H, $ghw(H) \leq hw(H) \leq qw(H) \leq 3ghw(H) + 1$.*

In particular, this means that each class of hypergraphs has bounded ghw if and only if it has bounded hw. In addition to the fact that "$hw(H) = k$" can be decided in polynomial time for each fixed constant k, this is, of course, a very nice property of hypertree-width.

Recently, a comparison between hypertree-width and clique-width was made [GP04]. Given that clique-width is defined for graphs, it had to be suitably adapted to hypergraphs. Defining the clique-width of a hypergraph H as the clique-width of its primal graph $G_p(H)$ makes no sense in the context of CSP-tractability, because then CSPs of bounded clique-width would be intractable (this can be seen, for example, by encoding the CLIQUE problem into a CSP of clique width 2). Therefore, in [GP04], the clique-width $cw(H)$ of a hypergraph H is defined as the clique-width of its incidence graph $G_i(H)$. With this definition, the following could be shown.

Theorem 5.13 (G. Gottlob and R. Pichler [GP04]).

1. *CSPs whose hypergraphs have bounded clique-width are tractable.*
2. *For each hypergraph H, $hw(H) \leq qw(H) \leq cw(H)$.*
3. *There are classes of hypergraphs having bounded hypertree-width and bounded query-width but unbounded clique-width.*

5.7 The Robber and Marshals Game

In [ST93], graphs G of tree-width k are characterized by the so called *Robber-and-Cops game* where $k+1$ cops have a winning strategy for capturing a robber on G. Cops can control vertices of a graph and can fly at each move to arbitrary vertices, say, by using a helicopter. The robber can move (at infinite speed) along paths of G, and will try to escape the approaching helicopter(s), but cannot go over vertices controlled by a cop. It is, moreover, shown that a winning strategy for the cops exists, if and only if the cops can capture the robber in a *monotonic* way, i.e., never returning to a vertex that a cop has previously vacated, which implies that the moving area of the robber is monotonically shrinking. Thus, with respect to tree-width, the general (possibly non-monotonic) robber and cops game and the monotonic robber and cops game coincide. For more detailed descriptions of the game, see [ST93] or [GLS01].

In order to provide a similarly natural characterization for hypertree-width, a new game, the *robber and marshals game (R&Ms game)*, was defined in [GLS03]. A marshal is more powerful than a cop. While a cop can control a single vertex of a hypergraph H only, a marshal controls an entire hyperedge. In the *R&Ms* game, the robber moves on vertices along a path of H (which is a path of the primal graph of H) just as in the robber and cops game, but now marshals instead of cops are chasing the robber.

During a move of the marshals from the set E of hyperedges to the set E' of hyperedges, the robber cannot pass through the vertices in $B = (\bigcup E) \cap (\bigcup E')$, where, for a set of hyperedges F, $\bigcup F$ denotes the union of all hyperedges in F. Intuitively, the vertices in B are those not released by the marshals during the move.

In this game, the set of all marshals is considered to be one player and the robber the other player. The objective of the marshals is thus to move a marshal (via helicopter) on a hyperedge containing the vertex occupied by the robber. The robber tries to elude capture. As for the robber and cops game, we distinguish between a *general* (not necessarily monotone) and a *monotone* version of the $R\&Ms$ game. In the monotone version of the game, the marshals have to make sure, that in each step the robber's escape space, which is, the component in which the robber can freely move around, decreases. The *(monotone) marshal-width* of a hypergraph H, $\mathrm{mw}(H)$ (and $\text{mon-mw}(H)$, respectively), is the least number k of marshals that have a (monotone) winning strategy in the robber and k marshals game played on H (see [Adl04, GLS01] for more precise definitions).

Clearly, for each hypergraph H, $\mathrm{mw}(H) \leq \text{mon-mw}(H)$. However, unlike for the robber and cops game, the marshal-width and the monotone marshal-width differ. Adler [Adl04] proved that for each constant k there is a hypergraph H such that $\text{mon-mw}(H) - \mathrm{hw}(H) = k$. In addition, several further interesting results on marshal games can be found in [Adl04].

In [GLS03] it is shown that there is a one-to-one correspondence between the winning strategies for k marshals in the monotone game and the normal-form hypertree decompositions of width at most k.

Theorem 5.14 (G. Gottlob, N. Leone, and F. Scarcello [GLS03])**.** *A hypergraph H has hypertree-width at most k, if and only if k marshals have a winning strategy for the monotone R&Ms game played on H.*

It is the special condition for hypertree-width which intuitively corresponds to the monotonicity requirement of the robber and marshals game.

5.8 Hyperlinkedness and Brambles

We discuss the following hypergraph invariants that are generalizations of well-known graph invariants.

Hyperlinkedness. Let H be a hypergraph, $M \subseteq E(H)$ and $C \subseteq V(H)$. C is *M-big*, if it intersects more than half of the edges of M, that is, $|\{e \in M : e \cap C \neq \emptyset\}| > \frac{|M|}{2}$. Note that if $S \subseteq E(H)$, then $H \setminus \bigcup S$ has at most one M-big connected component. Let k be a positive integer. A set $M \subseteq E(H)$ is *k-hyperlinked*, if for any set $S \subseteq E(H)$ with $|S| < k$, $H \setminus \bigcup S$ has an M-big component. The largest k for which H contains a k-hyperlinked set is called *hyperlinkedness of H*, hlink(H). Hyperlinkedness is an adaptation of the linkedness of a graph [Ree97] to the setting of hypergraphs.

Brambles. Let H be a hypergraph. Sets $X_1, X_2 \subseteq V(H)$ *touch* if $X_1 \cap X_2 \neq \emptyset$ or or there exists an $e \in E(H)$ such that $e \cap X_1 \neq \emptyset$ and $e \cap X_2 \neq \emptyset$. A *bramble of H* is a set B of pairwise touching connected subsets of $V(H)$. This is defined in analogy to brambles of graphs [Ree97]. The *hyper-order of a bramble B* is the least integer k such that there exists a set $R \subseteq E(H)$ with $|R| = k$ and $\bigcup R \cap X \neq \emptyset$ for all $X \in B$. The *hyperbramble number* hbramble-no(H) of H is the maximum of the hyper-orders of all brambles of H.

Theorem 5.15 (I. Adler, G. Gottlob, and M. Grohe [AGG])**.** *For each hypergraph H,* hlink(H) $\leq$ hbramble-no(H) $\leq$ mw(H) $\leq$ ghw(H) $\leq$ mon-mw(H) $=$ hw(H) $\leq 3 \cdot$ hlink(H) $+ 1$.

Corollary 5.16. *The hypergraph invariants* hlink*,* hbramble-no*,* mw*,* ghw*,* mon-mw*, and* hw *are all equivalent with respect to boundedness. In other words, the following are equivalent:*

- *A class of hypergraph has bounded* hlink*,*
- *a class of hypergraph has bounded* hbramble-no*,*
- *a class of hypergraph has bounded* mw*,*
- *a class of hypergraph has bounded* ghw*,*
- *a class of hypergraph has bounded* mon-mw*,*
- *a class of hypergraph has bounded* hw*.*

Further hypergraph invariants related to hypertree-width are discussed in [AGG].

6 Concluding remarks

At last we add notes on several research directions that have not fit into the core survey in the previous chapters, but which are currently active and which we consider interestingly related to our main topic — structural width parameters in computer science. Our overall goal is to show the readers that there is a

lot of theoretical research going on in this area, besides ordinary graph tree-width, and that there is a high potential here for new applications in efficient algorithmic design. Of course, many deep theoretical questions about width parameters remain unaswered yet, and new theoretical research directions still wait to be opened.

6.1 Vertex-free view of tree-width

Consider now the interesting fact that all definitions of graph tree-width introduced so far make explicit use of vertices. That seems to be the major obstructions when trying to extend this notion to other combinatorial structures (except possibly hypergraphs). On the other hand, the notion of branch-width, for instance, have been straightforwardly generalized to all structures possessing a reasonable measure of connectivity, and Chapter 4 shows how useful such abstract generalizations could be for solving seemingly unrelated problems (clique-width). So we return back to the tree-width notion from a new, *vertex-free* perspective [HW03], inspired by matroids.

A *VF-tree-decomposition* of a graph G is a pair (T, τ), where T is a tree, and $\tau : E(G) \to V(T)$ is an arbitrary mapping of edges to the tree nodes. (The shortcut VF refers to "vertex-free".) For a node x of T, denote the connected components (branches) of $T - x$ by $T_1, \ldots, T_d$ and set $F_i = \tau^{-1}\big(V(T_i)\big)$. The *node-width* of x is defined by $|V(G)| + (d-1)\cdot c(G) - \sum_{i=1}^{d} c(G \setminus F_i)$, where $c(H)$ denotes the number of components of a graph H. The *width* of the decomposition (T, τ) is the maximum width over all the nodes of T, and the smallest width over all VF-tree-decompositions of G is the *VF-tree-width* of G. See Figure 22.

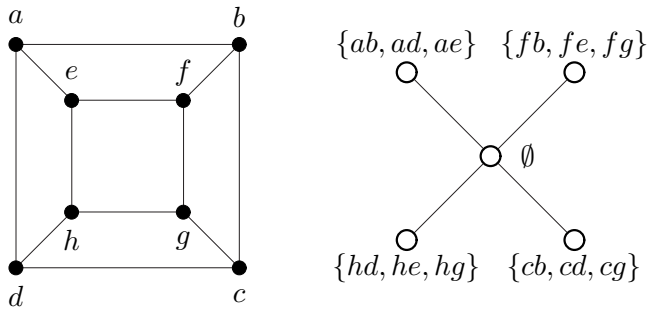


Figure 22: An example of a VF-tree-decomposition of the cube graph of width 3, where the images of edges under τ are listed at the tree nodes (cf. Fig. 3).

At first glance, it may seem surprising that this definition has anything in common with the traditional definition from Section 1.1. It is important to note that the mapping τ makes no analogue of bags. Instead, τ replaces the first condition of a tree-decomposition, and the second condition (interpolation) is "embedded" inside the formula for node-width. Yet their equality can be proved:

Theorem 6.1 (Hliněný and Whittle [HW03])**.** *The tree-width of a nonempty graph G equals the VF-tree-width of G.*

The proof of this statement makes essential use of a "geometrical" view of the VF-tree-width definition provided by its straightforward matroidal extension: The node-width in a VF-tree-decomposition of a matroid M on E is given by

$\sum_{i=1}^{d} \mathrm{rank}_M\left(E \setminus F_i\right) - (d-1) \cdot \mathrm{rank}(M)$, which is exactly equal to the above expression on cycle matroids of graphs. Even this matroidal generalization of tree-width is in a close relation with branch-width [HW03]:

Theorem 6.2. *Let M be a matroid of tree-width t and branch-width $b > 1$. Then $b \leq t + 1 \leq 2b - 1$, and both bounds are best possible.*

6.2 Width parameters on digraphs

The question of finding a suitable extension of width notions from undirected to directed graphs, with similarly nice algorithmic properties, seems to be a challenging problem. There appear to be two key properties such an extensions should possess: First, a "directed width" should be closely related with ordinary width parameters on symmetric orientations of graphs. Second, a "directed width" of acyclic digraphs should be low.

Nowadays there exist two competing extensions of ordinary tree-width with such properties – the older directed tree-width by Johnson, Robertson, Seymour, and Thomas [JRST01], and the recent DAG-width which has been independently proposed by Obdržálek [Obd06b] and by Berwanger, Dawar, Hunter and Kreutzer [BDHK06]. We briefly survey both of them here, and outline their mutual relations and differences.

Directed tree-width. This one was introduced by Johnson, Robertson, Seymour and Thomas in [JRST01] as a counterpart of tree-width for directed graphs. The decomposition structure is still a tree, though this time directed (with a designated root). For a directed acyclic graph R we use the following notation: If $r, r' \in V(R)$ we write $r < r'$ if there is a directed path with initial vertex r and terminal vertex r'. We write $r \leq r'$ if $r < r'$ or $r = r'$. Finally, for $e \in E(R)$, $e \sim r$ if e is incident with r. For a graph G, a set $S \subseteq V \setminus Z$ is *Z-normal* if there is no directed path in $G \setminus Z$ with the first and last vertices in S that uses a vertex of $G \setminus (S \cup Z)$.

Definition. An *arboreal decomposition* of a digraph G is a triple $(R, \mathcal{X}, \mathcal{W})$ where R is a directed tree, and $\mathcal{X} = \{X_e : e \in E(R)\}$, $\mathcal{W} = \{W_r : r \in V(R)\}$ are sets of vertices of G satisfying:

(R1) $\mathcal{W}$ is a partition of $V(G)$ into nonempty sets

(R2) for $e \in E(R)$, $e = (r_1, r_2)$ the set $\bigcup\{W_r : r \in V(R) \text{ and } r \geq r_2\}$ is X_e-normal.

The *width* of $(R, \mathcal{X}, \mathcal{W})$ is the least integer w such that for all $r \in V(R)$, $|W_r \cup \bigcup_{e \sim r} X_e| \leq w$. The *directed tree-width* of a digraph G is the minimum width over all possible arboreal decompositions of G.

DAG-width. Another extension resigns on having a "tree structure" as the decomposiion basis, but it uses a general acyclic digraph (DAG). (This indeed looks quite natural for algorithmic applications on directed graphs.) Here is the formal definition [Obd06b, BDHK06]:

Definition. A *DAG-decomposition* of a digraph G is a pair $(D, \mathcal{X})$ where D is a DAG and $\mathcal{X} = \{X_d : d \in V(D)\}$ is a family of subsets of $V(G)$ satisfying:

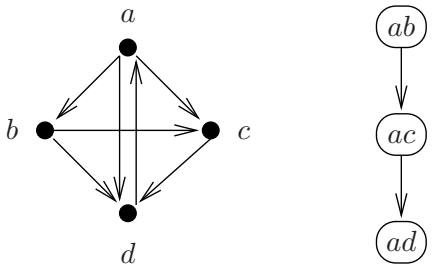


Figure 23: An illustration of a digraph and its DAG-decomposition.

(D1) $V(G) = \bigcup_{d \in V(D)} X_d$

(D2) If $(d, d') \in E(D)$, then for each $(v, w) \in E$ s.t. $v \in X_{\geq d'} \setminus X_d$ we have $w \in X_{\geq d'}$, where $X_{\geq c} = \bigcup_{c' \geq c} X_{c'}$. If d' is a root we replace X_d with $\emptyset$.

(D3) For all $d, d', d'' \in D$, if d' lies on a path from d to d'', then $X_d \cap X_{d''} \subseteq X_{d'}$.

The *width* of a DAG-decomposition $(D, \mathcal{X})$ is $\max_{d \in D} |X_d| - 1$. The *DAG-width* of a digraph G is the minimum width over all DAG-decompositions of G.

Acyclic digraphs have DAG-width zero. To get a better intuition, see Fig. 23. As ordinary tree-width of a graph [ST93] (also Section 5.7), DAG-width of a digraph can be exactly characterized [Obd06b] by a natural directed version of a robber-and-cops game in which the robber may freely move along cop-free directed paths. (That means there is no unnatural restriction of robber movement only to strong components as in [JRST01].)

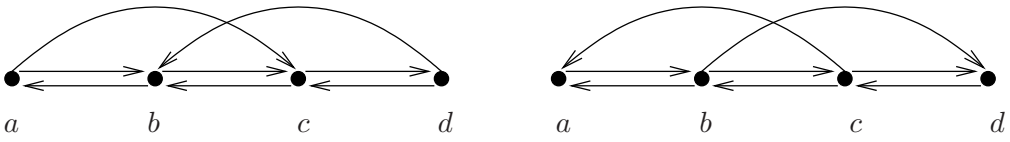


Figure 24: [Obd06b] Graphs G (left) and $\overline{G}$, having different DAG-widths.

One of the crucial properties of DAG-width is that, unlike directed tree-width, DAG-width is not preserved under the operation of reversing edges [Obd06a]. Check the digraphs G and $\overline{G}$ in Fig. 24: It is easy to see that 3 cops are needed to catch the robber in the digraph G, but only 2 are required in the case of the "reversed" digraph $\overline{G}$. In the opposite direction it is possible to find a graph such that its DAG-width is lower then its directed tree-width [Obd06a]. Actually the only reason why directed tree-width of a graph is not always at most its DAG-width is a single word in the definition of an arboreal decomposition. If we drop the requirement for $\mathcal{W}$ to have all sets non-empty, it is not hard to translate a DAG-decomposition into an arboreal decomposition (with empty nodes allowed) of the same width. As could be expected, the DAG-width of a digraph can be arbitrarily higher than its directed tree-width. A canonical example of this is a complete directed binary tree with "back" edges.

6.3 Width and decidability of theories

Let us finally ask; is it possible to find classes of graphs (structures) for which all MSO problems can be solved in polynomial time, but is not possible to reduce them to trees in the strong (interpretation) way?

Decidability of theories. It seems reasonable here to consider *theories* of classes of structures. Until now we have focused only on single algorithmic problems. From now on we shall consider whole theories, in other words collections of algorithmic problems in the above sense. We assume that a suitable language L is fixed for a certain class $\mathcal{K}$ of structures. The *L-theory of $\mathcal{K}$* is defined as $Th_L(\mathcal{K}) := \{\varphi : \varphi \text{ is a sentence of } L, \text{ and } G \models \varphi \text{ for all } G \in \mathcal{K}\}$. Hence, a theory is just the set of all the properties which all structures of $\mathcal{K}$ possess. When $\mathcal{K} = \{G\}$ we write $Th_L(G)$ instead of $Th_L(\mathcal{K})$. Using this notation we obtain $Th_L(\mathcal{K}) = \bigcap_{G \in \mathcal{K}} Th_L(G)$. A theory is said to be *decidable* if there is an algorithm deciding for an arbitrary sentence φ in L, whether $\varphi \in Th_L(\mathcal{K})$ or not, i.e. whether φ is true in all structures of $\mathcal{K}$. Otherwise this theory is said to be *undecidable*.

The surprising fact concerning decidability of theories is that the trade off between decidability or undecidability of a theory, the structure of the models of the theory and the expressive power of the corresponding logic shows the same behavioral patterns that we have already observed for the complexity of algorithmic problems. A good introduction into the decidability of theories can be found in [Rab77]. Surveys on first order theories, monadic second order theories and theories with generalized quantifiers are given in [San78, Gur85, BSTW80, BSTW85]. One of the strongest decidability results for MSO-theories of classes of graphs is the following result.

Theorem 6.3. (Rabin [Rab69], Shelah [She75]) *The monadic second order theory of the class of all (infinite) trees is decidable.*

The result has been proved by Rabin in the countable case by reducing it to the MSO-theory of the two successor functions $S2S$ (we introduced this structure as the complete infinite binary tree), which is decidable using tree automata running on the complete binary tree. The result was extended then by Shelah and Stupp to arbitrary trees. One of the main tools to prove decidability or undecidability of theories is again interpretability (Section 2.3): If $Th_L(\mathcal{K})$ is interpretable in $Th_{L'}(\mathcal{K}')$, then the decidability of $Th_{L'}(\mathcal{K}')$ implies the decidability of $Th_L(\mathcal{K})$. In this way one readily proves that the MSO-theory of the class of all graphs as incidence structures (MS_2) of tree-width at most k is decidable, for every k. For instance a slight modification of Theorem 3.9 extends such a decidability result to all matroids representable over any fixed finite field.

There are plenty of other interesting studies and results in this direction, such as on TreeMSO-classes (or tree-like) [CM00, CM02], on tree-interpretable infinite structures [Blu04, BG00], or on the use of interpretability technique [Rab65, Rab69, Rab77, See75, See76, See79, RH71, BSTW80, BSTW85].

Width and decidability. We should, however, return back to the motivation question of this section — whether it is possible to find classes of structures for which all MSO problems are efficiently solvable and which do not fall into our scheme of tree-like (interpretable) structures. Unfortunately, as it is usual in

the complexity area, we are not able to give absolute answers here, but we can say something substantial about the related theory decidability question:

Theorem 6.4. (Seese [See91]) *If $\mathcal{K}$ is a class of planar graphs such that $Th_{\mathrm{MS}_2}(\mathcal{K})$ is decidable, then there is k such that each $G \in \mathcal{K}$ has tree-width at most k.*

The proof of this theorem uses Theorem 1.4 and an encoding of the undecidable tiling problem (Theorem 1.1) in arbitrarily large grids.

Conjecture 6.5. (Seese [See91]) *Assume that $\mathcal{K}$ is a class of countable simple graphs with a decidable MSO-theory. Then there exists a class T of trees such that $Th_{\mathrm{MSO}}(\mathcal{K})$ is interpretable into $Th_{\mathrm{MSO}}(T)$, or equivalently $\mathcal{K}$ has bounded clique-width.*

This conjecture has been confirmed for several classes of structures by Courcelle [Cou00, Cou03, Cou06] (see also [Blu04]), and recently a slight weakening (using MSO plus parity counting) of the conjucture has been proved by Courcelle and Oum [CO06]. Also within some other classes of structures till now there are no known counterexamples to Conjecture 6.5, see [HS06] for more details.

Concluding message. To conclude this section we gather here the main message again. We have shown how structural and algorithmic properties of graphs, networks and other structures can be described in logical languages, which were basically extensions of monadic second order logic (using quantification on sets). These languages have a suitable model theory especially for trees, which allows deciding the truth of formulas for given trees in a uniform way by reducing formulas equivalently to automata working on trees. These decisions can be performed in linear time since the automata are nothing else than a compact code for dynamic programming algorithms running on the trees. This technique can be understood as a sort of universal problem solver for structures of bounded width and algorithmic problems expressible in the discussed calculus.

The corresponding results explain a part of the borderline between NP-complete and polynomial or even linear time solvable problems for graphs (Figure 4). This borderline, moreover, gets more clear if one looks at a stronger problem, the decidability of MSO theories.

For space restrictions we have had to exclude other concepts and results from logic which nevertheless are of interest in this area. One of these concepts are results related to the Feferman-Vaught theorem [FV59, Mak04], which investigate how the truth of a formula can be computed for strutures generated by certain operations. This proved to be a very fruitful concept for graphs of bounded width if a graph is built by special operations or can be decomposed via special decomposition operations [Mak04, CMR98, ACPS93].

Acknowledgments

P. Hliněný acknowledges support by Czech research grant GAČR 201/05/0050, and by the Institute of Theoretical Computer Science, project 1M0545. S. Oum has been partially supported by NSF grant 0354742. G. Gottlob's work has been supported by the Austrian Science Fund Project P17222-N04, "Complementary

Approaches to Constraint Satisfaction". The authors would also like to thank Jan Obdržálek for giving his expert opinion on digraph width notions.

References

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation*. Springer-Verlag, Berlin, 1999.

[ACPS93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. *An algebraic theory of graph reduction*. J. Assoc. Comput. Mach., 1993. 40(5):1134–1164.

[AD96] C. Allauzen and B. Durand. *Tiling problems*. 1996 pages 407–420. In [BGG97].

[Adl04] I. Adler. *Marshals, monotone Marshals, and hypertree-width*. J. Graph Theory, 2004. 47(4):275–296.

[AGG] I. Adler, G. Gottlob, and M. Grohe. *Hypertree-width and related hypergraph invariants*. European Journal of Combinatorics. To appear. (Short version in *Proc. 2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05)*, DMTCS Proceedings Volume AE (2005), Stefan Felsner Editor.).

[ALS88] S. Arnborg, J. Lagergren, and D. Seese. *Problems easy for tree-decomposable graphs (extended abstract)*. In *Automata, languages and programming (Tampere, 1988)*, volume 317 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1988 pages 38–51.

[ALS91] S. Arnborg, J. Lagergren, and D. Seese. *Easy problems for tree-decomposable graphs*. J. Algorithms, 1991. 12(2):308–340.

[AP86a] S. Arnborg and A. Proskurowski. *Characterization and recognition of partial 3-trees*. SIAM J. Algebraic Discrete Methods, 1986. 7(2):305–314.

[AP86b] S. Arnborg and A. Proskurowski. *Problems on graphs with bounded decomposability*. In *Proceedings of the 17th South–Eastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, Winnipeg*, volume 53 of *Congressus Numerantium*. 1986 pages 167–170.

[AP89] S. Arnborg and A. Proskurowski. *Linear time algorithms for NP-hard problems restricted to partial k-trees*. Discrete Appl. Math., 1989. 23(1):11–24.

[APS91] S. Arnborg, A. Proskurowski, and D. Seese. *Monadic second order logic, tree automata and forbidden minors*. In *Computer science logic (Heidelberg, 1990)*, volume 533 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1991 pages 1–16.

[Bar77] J. Barwise. *An introduction to first-order logic*. In *Handbook of Mathematical Logic*. North-Holland Publishing Company, 1977 pages 5–46.

[BDHK06] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. *DAG-width and parity games*. In STACS '06, volume 3884 of *Lecture Notes in Comput. Sci.*. Springer, 2006 pages 524–536.

[BDLM05] A. Brandstädt, F. F. Dragan, H.-O. Le, and R. Mosca. *New graph classes of bounded clique-width*. Theory Comput. Syst., 2005. 38(5):623–645.

[BELL05] A. Brandstädt, J. Engelfriet, H.-O. Le, and V. V. Lozin. *Clique-width for four-vertex forbidden subgraphs*. In M. Liskiewicz and R. Reischuk, editors, *Proc. FCT2005*, volume 3623 of *Lecture Notes in Computer Science*. Springer Verlag, 2005 pages 185–196.

[BELL06] A. Brandstädt, J. Engelfriet, H.-O. Le, and V. V. Lozin. *Clique-width for four-vertex forbidden subgraphs*. Theory Comput. Syst., 2006. Accepted, extended abstract in [BELL05].

[Ber66] R. Berger. *The undecidability of the domino problem*. Mem. Amer. Math. Soc. No., 1966. 66:1–72.

[BG00] A. Blumensath and E. Grädel. *Automatic structures*. In *15th Annual IEEE Symposium on Logic in Computer Science (Santa Barbara, CA, 2000)*. IEEE Comput. Soc. Press, Los Alamitos, CA, 2000 pages 51–62.

[BGG97] E. Börger, E. Grädel, and Y. Gurevich. *The classical decision problem*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1997. With an appendix by Cyril Allauzen and Bruno Durand.

[BK06] H. L. Bodlaender and A. Koster. *Combinatorial optimisation on graphs of bounded treewidth*. THIS JOURNAL, 2006. THIS VOLUME.

[BL02] R. Boliac and V. Lozin. *On the clique-width of graphs in hereditary classes*. In *Algorithms and Computation : 13th International Symposium, ISAAC 2002, Vancouver, BC, Canada, November 21-23, 2002*, volume 2518 of *Lecture Notes in Comput. Sci.*. Springer, 2002 pages 44–54.

[BLM05] A. Brandstädt, H.-O. Le, and R. Mosca. *Chordal co-gem-free and (P_5,gem)-free graphs have bounded clique-width*. Discrete Appl. Math., 2005. 145(2):232–241.

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph classes: a survey*. SIAM Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[Blu04] A. Blumensath. *Axiomatising tree-interpretable structures*. Theory Comput. Syst., 2004. 37(1):3–27. Symposium on Theoretical Aspects of Computer Science (Antibes-Juan les Pins, 2002).

[BLW87] M. W. Bern, E. L. Lawler, and A. L. Wong. *Linear-time computation of optimal subgraphs of decomposable graphs*. J. Algorithms, 1987. 8(2):216–235.

[BM86] H.-J. Bandelt and H. M. Mulder. *Distance-hereditary graphs*. J. Combin. Theory Ser. B, 1986. 41(2):182–208.

[BO95] L. Babel and S. Olariu. *On the isomorphism of graphs with few P_4's*. In *Graph-theoretic concepts in computer science (Aachen, 1995)*, volume 1017 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1995 pages 24–36.

[Bod88] H. L. Bodlaender. *Dynamic programming on graphs with bounded treewidth*. In *Automata, languages and programming (Tampere, 1988)*, volume 317 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1988 pages 105–118.

[Bod96] H. L. Bodlaender. *A linear-time algorithm for finding tree-decompositions of small treewidth*. SIAM J. Comput., 1996. 25(6):1305–1317.

[Bou87] A. Bouchet. *Isotropic systems*. European J. Combin., 1987. 8(3):231–244.

[Bou88] A. Bouchet. *Transforming trees by successive local complementations*. J. Graph Theory, 1988. 12(2):195–207.

[Bou89] A. Bouchet. *Connectivity of isotropic systems*. In *Combinatorial Mathematics: Proceedings of the Third International Conference (New York, 1985)*, volume 555 of *Ann. New York Acad. Sci.*. New York Acad. Sci., New York, 1989 pages 81–93.

[BP68] L. W. Beineke and R. E. Pippert. *The enumeration of labelled 2-trees*. Notices Amer. Math. Soc., 1968. 15:384.

[BP69] L. W. Beineke and R. E. Pippert. *The number of labeled k-dimensional trees*. J. Combinatorial Theory, 1969. 6:200–205.

[BPT92] R. B. Borie, R. G. Parker, and C. A. Tovey. *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*. Algorithmica, 1992. 7(5-6):555–581.

[BSTW80] A. Baudisch, D. Seese, H.-P. Tuschik, and M. Weese. *Decidability and generalized quantifiers*, volume 3 of *Mathematical Research*. Akademie-Verlag, Berlin, 1980.

[BSTW85] A. Baudisch, D. Seese, P. Tuschik, and M. Weese. *Decidability and quantifier-elimination*. In *Model-theoretic logics*, Perspect. Math. Logic. Springer, New York, 1985 pages 235–270.

[BT97] H. L. Bodlaender and D. M. Thilikos. *Constructive linear time algorithms for branchwidth*. In *Automata, languages and programming (Bologna, 1997)*, volume 1256 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1997 pages 627–637.

[BT99] H. L. Bodlaender and D. M. Thilikos. *Graphs with branchwidth at most three*. J. Algorithms, 1999. 32(2):167–194.

[CH87] K. J. Compton and C. W. Henson. *A uniform method for proving lower bounds on the computational complexity of logical theories*. Technical Report CRL-TR-04-87, Computing Research Laboratory, University of Michigan, 1987.

[CHL+00] D. G. Corneil, M. Habib, J.-M. Lanlignel, B. Reed, and U. Rotics. *Polynomial time recognition of clique-width ≤ 3 graphs (extended abstract)*. In *Gonnet, Gastón H. (ed.) et al., LATIN 2000: Theoretical informatics. 4th Latin American symposium, Punta del Este, Uruguay, April 10-14, 2000.*, volume 1776 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 2000 pages 126–134.

[CLB81] D. G. Corneil, H. Lerchs, and L. S. Burlingham. *Complement reducible graphs*. Discrete Appl. Math., 1981. 3(3):163–174.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[CM77] A. K. Chandra and P. M. Merlin. *Optimal implementation of conjunctive queries in relational data bases*. In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing (Boulder, Colo., 1977)*. Assoc. Comput. Mach., New York, 1977 pages 77–90.

[CM93] B. Courcelle and M. Mosbah. *Monadic second-order evaluations on tree-decomposable graphs*. Theoret. Comput. Sci., 1993. 109(1-2):49–82.

[CM00] B. Courcelle and J. A. Makowsky. *VR and HR graph grammars: a common algebraic framework compatible with monadic second-order logic (extended abstract)*, 2000. Preprint, accepted to GRATRA'2000.

[CM02] B. Courcelle and J. A. Makowsky. *Fusion in relational structures and the verification of monadic second-order properties*. Math. Structures Comput. Sci., 2002. 12(2):203–235.

[CMR98] B. Courcelle, J. A. Makowsky, and U. Rotics. *Linear time solvable optimization problems on graphs of bounded clique width (extended abstract)*. In *Graph-theoretic concepts in computer science (Smolenice Castle, 1998)*, volume 1517 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1998 pages 1–16.

[CMR00] B. Courcelle, J. A. Makowsky, and U. Rotics. *Linear time solvable optimization problems on graphs of bounded clique-width*. Theory Comput. Syst., 2000. 33(2):125–150.

[CMR01] B. Courcelle, J. A. Makowsky, and U. Rotics. *On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic*. Discrete Appl. Math., 2001. 108(1-2):23–52.

[CO00] B. Courcelle and S. Olariu. *Upper bounds to the clique width of graphs*. Discrete Appl. Math., 2000. 101(1-3):77–114.

[CO06] B. Courcelle and S. Oum. *Vertex-minors, monadic second-order logic, and a conjecture by Seese*. J. Combin. Theory Ser. B, 2006. Accepted.

[Cou90] B. Courcelle. *The monadic second-order logic of graphs IV: Definability properties of equational graphs*. Ann. Pure Appl. Logic, 1990. 49(3):193–255.

[Cou92a] B. Courcelle. *The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues*. RAIRO Inform. Théor. Appl., 1992. 26(3):257–286.

[Cou92b] B. Courcelle. *The monadic second-order logic of graphs VII: Graphs as relational structures*. Theoret. Comput. Sci., 1992. 101(1):3–33.

[Cou94a] B. Courcelle. *Monadic second-order definable graph transductions: a survey*. Theoret. Comput. Sci., 1994. 126(1):53–75.

[Cou94b] B. Courcelle. *The monadic second order logic of graphs VI: On several representations of graphs by relational structures*. Discrete Appl. Math., 1994. 54(2-3):117–149.

[Cou97] B. Courcelle. *The expression of graph properties and graph transformations in monadic second-order logic*. In *Handbook of graph grammars and computing by graph transformation, Vol. 1*. World Sci. Publishing, River Edge, NJ, 1997 pages 313–400.

[Cou00] B. Courcelle. *The monadic second-order logic of graphs XII: Planar graphs and planar maps*. Theoret. Comput. Sci., 2000. 237(1-2):1–32.

[Cou03] B. Courcelle. *The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications*. Theoret. Comput. Sci., 2003. 299(1-3):1–36.

[Cou04] B. Courcelle. *Clique-width of countable graphs: a compactness property*. Discrete Math., 2004. 266(1-3):127–148.

[Cou06] B. Courcelle. *The monadic second-order logic of graphs XV: On a conjecture by D. Seese*. J. Appl. Log., 2006. 4(1):79–114.

[CPS85] D. G. Corneil, Y. Perl, and L. K. Stewart. *A linear recognition algorithm for cographs*. SIAM J. Comput., 1985. 14(4):926–934.

[CR00] C. Chekuri and A. Rajaraman. *Conjunctive query containment revisited*. Theoret. Comput. Sci., 2000. 239(2):211–229.

[CR05] D. G. Corneil and U. Rotics. *On the relationship between clique-width and treewidth*. SIAM J. Comput., 2005. 34(4):825–847 (electronic).

[Cun82] W. H. Cunningham. *Decomposition of directed graphs*. SIAM J. Algebraic Discrete Methods, 1982. 3(2):214–228.

[Dec92] R. Dechter. *Constraint networks*. In *Encyclopedia of artificial intelligence*. Wiley and Sons, 1992 pages 276–285.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.

[Dha94] J. S. Dharmatilake. *Binary matroids of branch-width* 3. Ph.D. thesis, Ohio State University, 1994.

[DHP01] G. Damiand, M. Habib, and C. Paul. *A simple paradigm for graph recognition: application to cographs and distance hereditary graphs*. Theoret. Comput. Sci., 2001. 263(1-2):99–111.

[Die05] R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005.

[DP87] R. Dechter and J. Pearl. *Network-based heuristics for constraint-satisfaction problems*. Artificial Intelligence, 1987. 34(1):1–38.

[DP89] R. Dechter and J. Pearl. *Tree clustering for constraint networks*. Artificial Intelligence, 1989. 38(3):353–366.

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, second edition, 1999.

[EFT94] H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical logic*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, second edition, 1994. Translated from the German by Margit Meßmer.

[EGW01] W. Espelage, F. Gurski, and E. Wanke. *How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time*. In *Graph-theoretic concepts in computer science (Boltenhagen, 2001)*, volume 2204 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 2001 pages 117–128.

[EGW03] W. Espelage, F. Gurski, and E. Wanke. *Deciding clique-width for graphs of bounded tree-width*. Journal of Graph Algorithms and Applications, 2003. 7(2):141–180.

[Fag74] R. Fagin. *Generalized first-order spectra and polynomial-time recognizable sets*. In *Complexity of computation (Proc. SIAM-AMS Sympos. Appl. Math., New York, 1973)*. Amer. Math. Soc., Providence, R.I., 1974 pages 43–73. SIAM–AMS Proc., Vol. VII.

[Fag83] R. Fagin. *Degrees of acyclicity for hypergraphs and relational database schemes*. J. Assoc. Comput. Mach., 1983. 30(3):514–550.

[FG01] M. Frick and M. Grohe. *Deciding first-order properties of locally tree-decomposable structures*. J. ACM, 2001. 48(6):1184–1206 (electronic).

[FG04] M. Frick and M. Grohe. *The complexity of first-order and monadic second-order logic revisited*. Ann. Pure Appl. Logic, 2004. 130(1-3):3–31.

[FG06] J. Flum and M. Grohe. *Parameterized complexity theory*. Springer-Verlag, 2006.

[FL89] M. R. Fellows and M. A. Langston. *An analogue of the Myhill-Nerode theorem and its use in computing finite basis characterizations*. In *Proc. Symp. Foundations of Comp. Sci.*. 1989 pages 520–525.

[Fre85] E. C. Freuder. *A sufficient condition for backtrack-bounded search*. J. Assoc. Comput. Mach., 1985. 32(4):755–761.

[FRRS05a] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. *Proving NP-hardness for clique-width I: non-approximability of sequential clique-width*. Report TR05-080, revision 01, Electronic Colloquium on Computational Complexity, 2005.

[FRRS05b] M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. *Proving NP-hardness for clique-width II: non-approximability of clique-width*. Report TR05-081, Revision 01, Electronic Colloquium on Computational Complexity, 2005.

[FT03] F. V. Fomin and D. M. Thilikos. *Dominating sets in planar graphs: branch-width and exponential speed-up*. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (Baltimore, MD, 2003)*. ACM, New York, 2003 pages 168–177.

[FV59] S. Feferman and R. L. Vaught. *The first order properties of products of algebraic systems*. Fund. Math., 1959. 47:57–103.

[GGM+05] G. Gottlob, M. Grohe, N. Musliu, M. Samer, and F. Scarcello. *Hypertree decompositions: structure, algorithms, and applications*. In D. Kratsch, editor, *Graph-Theoretic Concepts in Computer Science: 31st International Workshop, WG 2005*, volume 3787 of *Lecture Notes in Comput. Sci.*. 2005 pages 1–15.

[GGRW03] J. F. Geelen, A. M. H. Gerards, N. Robertson, and G. Whittle. *On the excluded minors for the matroids of branch-width k*. J. Combin. Theory Ser. B, 2003. 88(2):261–265.

[GGW02] J. F. Geelen, A. M. H. Gerards, and G. Whittle. *Branch-width and well-quasi-ordering in matroids and graphs*. J. Combin. Theory Ser. B, 2002. 84(2):270–290.

[GGW03] J. F. Geelen, A. M. H. Gerards, and G. Whittle. *Excluding a planar graph from* GF(q)*-representable matroids*. Research Report 03-4, School of Mathematical and Computing Sciences, Victoria University of Wellington, 2003.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman and Co., San Francisco, Calif., 1979.

[GJC94] M. Gyssens, P. G. Jeavons, and D. A. Cohen. *Decomposing constraint satisfaction problems using database techniques*. Artificial Intelligence, 1994. 66(1):57–89.

[GLS99] G. Gottlob, N. Leone, and F. Scarcello. *On tractable queries and constraints*. In *DEXA '99: Proceedings of the 10th International Conference on Database and Expert Systems Applications*. Springer-Verlag, London, UK, 1999 pages 1–15.

[GLS00] G. Gottlob, N. Leone, and F. Scarcello. *A comparison of structural CSP decomposition methods*. Artificial Intelligence, 2000. 124(2):243–282.

[GLS01] G. Gottlob, N. Leone, and F. Scarcello. *The complexity of acyclic conjunctive queries*. J. ACM, 2001. 48(3):431–498.

[GLS02] G. Gottlob, N. Leone, and F. Scarcello. *Hypertree decompositions and tractable queries*. J. Comput. System Sci., 2002. 64(3):579–627. Special issue on PODS 1999 (Philadelphia, PA).

[GLS03] G. Gottlob, N. Leone, and F. Scarcello. *Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width*. J. Comput. System Sci., 2003. 66(4):775–808. Special issue on PODS 2001 (Santa Barbara, CA).

[Got05] G. Gottlob. *Computing cores for data exchange: new algorithms and practical solutions*. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM Press, New York, NY, USA, 2005 pages 148–159.

[GP84] M. Gyssens and J. Paredaens. *A decomposition methodology for cyclic databases*. In *Advances in database theory*, volume 2. Plenum Press, New York, NY, 1984 pages 85–122.

[GP04] G. Gottlob and R. Pichler. *Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width*. SIAM J. Comput., 2004. 33(2):351–378 (electronic).

[GR00] M. C. Golumbic and U. Rotics. *On the clique-width of some perfect graph classes*. Internat. J. Found. Comput. Sci., 2000. 11(3):423–443. Selected papers from the Workshop on Theoretical Aspects of Computer Science (WG 99), Part 1 (Ascona).

[GRT97] V. Giakoumakis, F. Roussel, and H. Thuillier. *On P_4-tidy graphs*. Discrete Math. Theor. Comput. Sci., 1997. 1(1):17–41 (electronic).

[Gur85] Y. Gurevich. *Monadic second-order theories*. In *Model-theoretic logics*, Perspect. Math. Logic. Springer, New York, 1985 pages 479–506.

[GW00] F. Gurski and E. Wanke. *The tree-width of clique-width bounded graphs without $K_{n,n}$*. In *Graph-theoretic concepts in computer science (Konstanz, 2000)*, volume 1928 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 2000 pages 196–205.

[GW04] F. Gurski and E. Wanke. *Vertex disjoint paths on clique-width bounded graphs (extended abstract)*. In *LATIN 2004: Theoretical informatics*, volume 2976 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 2004 pages 119–128.

[GW05a] F. Gurski and E. Wanke. *Line graphs of bounded clique-width*, 2005. Submitted.

[GW05b] F. Gurski and E. Wanke. *On the relationship between NLC-width and linear NLC-width*. Theoret. Comput. Sci., 2005. 347(1–2):76–89.

[Han74] W. Hanf. *Nonrecursive tilings of the plane. I*. J. Symbolic Logic, 1974. 39:283–285.

[Har83] D. Harel. *Recurring dominoes: making the highly undecidable highly understandable*. In *Foundations of computation theory (Borgholm, 1983)*, volume 158 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1983 pages 177–194.

[Hic05a] I. V. Hicks. *Planar branch decompositions. I. The ratcatcher*. INFORMS J. Comput., 2005. 17(4):402–412.

[Hic05b] I. V. Hicks. *Planar branch decompositions. II. The cycle method*. INFORMS J. Comput., 2005. 17(4):413–421.

[Hli02] P. Hliněný. *On the excluded minors for matroids of branch-width three*. Electron. J. Combin., 2002. 9(1):Research Paper 32, 13 pp. (electronic).

[Hli03] P. Hliněný. *On matroid properties definable in the MSO logic*. In *Mathematical foundations of computer science 2003*, volume 2747 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 2003 pages 470–479.

[Hli04] P. Hliněný. *Using a computer in matroid theory research*. Acta Univ. M. Belii Ser. Math., 2004. (11):27–44.

[Hli05] P. Hliněný. *A parametrized algorithm for matroid branch-width*. SIAM J. Comput., 2005. 35(2):259–277, loose erratum (electronic).

[Hli06a] P. Hliněný. *Branch-width, parse trees, and monadic second-order logic for matroids*. J. Combin. Theory Ser. B, 2006. 96(3):325–351.

[Hli06b] P. Hliněný. *On matroid representability and minor problems*, 2006. Submitted.

[Hli06c] P. Hliněný. *On some hard problems on matroid spikes*. Theory Comput. Syst., 2006. To appear.

[Hli06d] P. Hliněný. *The Tutte polynomial for matroids of bounded branch-width*. Combin. Probab. Comput., 2006. 15(3):397–409.

[HLW92] F. Höfting, T. Lengauer, and E. Wanke. *Processing of hierarchically defined graphs and graph families*. In *Data structures and efficient algorithms (Berlin, 1991)*, volume 594 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1992 pages 44–69.

[HM90] P. L. Hammer and F. Maffray. *Completely separable graphs*. Discrete Appl. Math., 1990. 27(1-2):85–99. Computational algorithms, operations research and computer science (Burnaby, BC, 1987).

[HMJ05] I. V. Hicks and N. B. McMurray Jr. *The branch-width of graphs and their cycle matroids*, 2005. Working paper, `http://ie.tamu.edu/People/faculty/Hicks/branch.matroid.openrev.pdf`.

[Hoà85] C. T. Hoàng. *Perfect graphs*. Ph.D. thesis, McGill University, Montreal, Canada, 1985.

[HR89] W. Hohberg and R. Reischuk. *Decompositions of graphs — a uniform approach for the design of fast sequential and parallel algorithms on graphs*. Technical report, TH Darmstadt, 1989.

[HR90] W. Hohberg and R. Reischuk. *A framework to design algorithms for optimization problems on graphs*. Preprint, TH Darmstadt, April 1990.

[HS06] P. Hliněný and D. Seese. *Trees, grids, and MSO decidability: From graphs to matroids*. Theoret. Comput. Sci., 2006. 351(3):372–393.

[HW03] P. Hliněný and G. Whittle. *Matroid tree-width*, 2003. Submitted, extended abstract in: Eurocomb'03, ITI Series 2003–145, Charles University, Prague, Czech Republic, 202–205.

[Imm87] N. Immerman. *Languages that capture complexity classes*. SIAM J. Comput., 1987. 16(4):760–778.

[Imm98] N. Immerman. *Descriptive complexity*. Graduate Texts in Computer Science. Springer-Verlag, New York, 1998.

[JO89] B. Jamison and S. Olariu. *P_4-reducible graphs—a class of uniquely tree-representable graphs*. Stud. Appl. Math., 1989. 81(1):79–87.

[Joh98] Ö. Johansson. *Clique-decomposition, NLC-decomposition, and modular decomposition—relationships and results for random graphs*. In *Proceedings of the Twenty-ninth Southeastern International Conference on Combinatorics, Graph Theory and Computing (Boca Raton, FL, 1998)*, volume 132. 1998 pages 39–60.

[Joh01] Ö. Johansson. *Graph decomposition using node labels*. Ph.D. thesis, Royal Institute of Technology, 2001.

[Joh03] J. L. Johnson. *Polynomial time recognition and optimization algorithms on special classes of graphs*. Ph.D. thesis, Vanderbilt University, 2003.

[JRST01] T. Johnson, N. Robertson, P. Seymour, and R. Thomas. *Directed tree-width*. J. Combin. Theory Ser. B, 2001. 82(1):138–154.

[KR03] D. Kobler and U. Rotics. *Edge dominating set and colorings on graphs with fixed clique-width*. Discrete Appl. Math., 2003. 126(2-3):197–221.

[KV00] P. G. Kolaitis and M. Y. Vardi. *Conjunctive-query containment and constraint satisfaction*. J. Comput. System Sci., 2000. 61(2):302–332.

[Len82] T. Lengauer. *The complexity of compacting hierarchically specified layouts of integrated circuits*. In *23rd annual symposium on foundations of computer science (Chicago, Ill., 1982)*. IEEE, New York, 1982 pages 358–368.

[Len86] T. Lengauer. *Exploiting hierarchy in vlsi design*. In *Proc. of the Aegean workshop on computing on VLSI algorithms and architectures*. Springer-Verlag New York, Inc., New York, NY, USA, 1986 pages 180–193.

[Len87] T. Lengauer. *Efficient algorithms for finding minimum spanning forests of hierarchically defined graphs*. J. Algorithms, 1987. 8(2):260–284.

[Len89] T. Lengauer. *Hierarchical planarity testing algorithms*. J. Assoc. Comput. Mach., 1989. 36(3):474–509.

[Len90] T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. Applicable Theory in Computer Science. John Wiley & Sons Ltd., Chichester, 1990. With a foreword by Bryan Preas.

[Lib04] L. Libkin. *Elements of finite model theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2004.

[Lov06] L. Lovász. *Graph minor theory*. Bull. Amer. Math. Soc. (N.S.), 2006. 43(1):75–86 (electronic).

[LW87] T. Lengauer and C. Wieners. *Efficient solutions of hierarchical systems of linear equations*. Computing, 1987. 39(2):111–132.

[LW88] T. Lengauer and E. Wanke. *Efficient solution of connectivity problems on hierarchically defined graphs*. SIAM J. Comput., 1988. 17(6):1063–1080.

[LW92] T. Lengauer and K. W. Wagner. *The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems*. J. Comput. System Sci., 1992. 44(1):63–93.

[Mak04] J. A. Makowsky. *Algorithmic uses of the Feferman-Vaught theorem*. Ann. Pure Appl. Logic, 2004. 126(1-3):159–213.

[Mon86] B. Monien. *The bandwidth minimization problem for caterpillars with hair length* 3 *is NP-complete*. SIAM J. Algebraic Discrete Methods, 1986. 7(4):505–512.

[MR99] J. A. Makowsky and U. Rotics. *On the clique-width of graphs with few P_4's*. Internat. J. Found. Comput. Sci., 1999. 10(3):329–348.

[MS94] T. H. Ma and J. Spinrad. *An $O(n^2)$ algorithm for undirected split decomposition*. J. Algorithms, 1994. 16(1):145–160.

[MS99] R. M. McConnell and J. Spinrad. *Modular decomposition and transitive orientation*. Discrete Math., 1999. 201(1-3):189–241.

[MT05] F. Mazoit and S. Thomassé. *Branchwidth of graphic matroids*, 2005. Preprint.

[Nob98] S. D. Noble. *Evaluating the Tutte polynomial for graphs of bounded tree-width*. Combin. Probab. Comput., 1998. 7(3):307–321.

[Obd06a] J. Obdržálek. *Algorithmic analysis of parity games*. Ph.D. thesis, University of Edinburgh, 2006.

[Obd06b] J. Obdržálek. *Dag-with: connectivity measure for directed graphs*. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (Miami, FL, 2006)*. ACM, New York, 2006 pages 814–821.

[OS05] S. Oum and P. Seymour. *Testing branch-width*, 2005. Submitted.

[OS06] S. Oum and P. Seymour. *Approximating clique-width and branch-width*. J. Combin. Theory Ser. B, 2006. To appear.

[Oum05a] S. Oum. *Approximating rank-width and clique-width quickly*. In *Graph-theoretic concepts in computer science (Metz, 2005)*, volume 3787 of *Lecture Notes in Comput. Sci.*. Springer, 2005 pages 49–58.

[Oum05b] S. Oum. *Graphs of bounded rank-width*. Ph.D. thesis, Princeton University, 2005.

[Oum05c] S. Oum. *Rank-width and vertex-minors*. J. Combin. Theory Ser. B, 2005. 95(1):79–100.

[Oum05d] S. Oum. *Rank-width and well-quasi-ordering*, 2005. Submitted.

[Oxl92] J. G. Oxley. *Matroid theory*. Oxford University Press, New York, 1992.

[Pap94] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Reading, MA, 1994.

[Pro84] A. Proskurowski. *Separating subgraphs in k-trees: cables and caterpillars*. Discrete Math., 1984. 49(3):275–285.

[Rab65] M. O. Rabin. *A simple method for undecidability proofs and some applications*. In *Logic, Methodology and Philos. Sci. (Proc. 1964 Internat. Congr.)*. North-Holland, Amsterdam, 1965 pages 58–68.

[Rab69] M. O. Rabin. *Decidability of second-order theories and automata on infinite trees.*. Trans. Amer. Math. Soc., 1969. 141:1–35.

[Rab77] M. O. Rabin. *Decidable theories*. In J. Barwise, editor, *Handbook of mathematical logic*. North-Holland Publishing Co., Amsterdam, 1977 pages 595–629.

[Ree97] B. A. Reed. *Tree width and tangles: a new connectivity measure and some applications*. In *Surveys in combinatorics, 1997 (London)*, volume 241 of *London Math. Soc. Lecture Note Ser.*. Cambridge Univ. Press, Cambridge, 1997 pages 87–162.

[RH71] W. Rautenberg and K. Hauschild. *Interpretierbarkeit und Entscheidbarkeit in der Graphentheorie. I*. Z. Math. Logik Grundlagen Math., 1971. 17:47–55.

[Rig01] K. D. Riggins. *On characterizing graphs with branchwidth at most four,*. Master's thesis, Rice University, 2001.

[Ros74] D. J. Rose. *On simple characterizations of k-trees*. Discrete Math., 1974. 7:317–322.

[RRT99] F. Roussel, I. Rusu, and H. Thuillier. *On graphs with limited number of P_4-partners*. Internat. J. Found. Comput. Sci., 1999. 10(1):103–121.

[RS83] N. Robertson and P. D. Seymour. *Graph minors. I. Excluding a forest*. J. Combin. Theory Ser. B, 1983. 35(1):39–61.

[RS84] N. Robertson and P. D. Seymour. *Graph minors. III. Planar tree-width*. J. Combin. Theory Ser. B, 1984. 36(1):49–64.

[RS85] N. Robertson and P. D. Seymour. *Graph minors—a survey*. In *Surveys in combinatorics 1985 (Glasgow, 1985)*, volume 103 of *London Math. Soc. Lecture Note Ser.*. Cambridge Univ. Press, Cambridge, 1985 pages 153–171.

[RS86] N. Robertson and P. Seymour. *Graph minors. II. Algorithmic aspects of tree-width*. J. Algorithms, 1986. 7(3):309–322.

[RS91] N. Robertson and P. Seymour. *Graph minors. X. Obstructions to tree-decomposition*. J. Combin. Theory Ser. B, 1991. 52(2):153–190.

[RS95] N. Robertson and P. Seymour. *Graph minors. XIII. The disjoint paths problem*. J. Combin. Theory Ser. B, 1995. 63(1):65–110.

[RS04] N. Robertson and P. Seymour. *Graph minors. XX. Wagner's conjecture*. J. Combin. Theory Ser. B, 2004. 92(2):325–357.

[San78] H. P. Sankappanavar. *Decision problems: history and methods*. In *Mathematical logic (Proc. First Brazilian Conf., State Univ. Campinas, Campinas, 1977)*, volume 39 of *Lecture Notes in Pure and Appl. Math.*. Dekker, New York, 1978 pages 241–291.

[Sch] T. Schwentick. Personal communication.

[See75] D. G. Seese. *Zur Entscheidbarkeit der monadischen Theorie zweiter Stufe baumartiger Graphen*. Wiss. Z. Humboldt-Univ. Berlin Math.-Natur. Reihe, 1975. 24(6):768–772. Theoretische Kybernetik und mathematische Logik.

[See76] D. Seese. *Entscheidbarkeits- und Interpretierbarkeitsfragen Monadischer Theorien zweiter Stufe gewisser Klassen von Graphen*. Ph.D. thesis, Humboldt-Universität, Berlin, 1976.

[See79] D. G. Seese. *Some graph-theoretical operations and decidability*. Math. Nachr., 1979. 87:15–21.

[See85a] D. Seese. *Baum-partite Graphen und die Komplexitaet von Algorithmen*. In *Proc.30th Intern. Wiss. Koll.* TH Ilmenau, 1985 pages 101–103.

[See85b] D. Seese. *Tree-partite graphs and the complexity of algorithms (extended abstract)*. In *Fundamentals of computation theory (Cottbus, 1985)*, volume 199 of *Lecture Notes in Comput. Sci.*. Springer, Berlin, 1985 pages 412–421.

[See86] D. Seese. *Tree-partite graphs and the complexity of algorithms*. Preprint P-Math 08/86, Karl-Weierstrass-Institute for Mathematics, Berlin, 1986. Full version of [See85b].

[See91] D. Seese. *The structure of the models of decidable monadic theories of graphs*. Ann. Pure Appl. Logic, 1991. 53(2):169–195.

[See92] D. Seese. *Interpretability and tree automata: a simple way to solve algorithmic problems on graphs closely related to trees*. In *Tree automata and languages (Le Touquet, 1990)*, volume 10 of *Stud. Comput. Sci. Artificial Intelligence*. North-Holland, Amsterdam, 1992 pages 83–114.

[See96] D. Seese. *Linear time computable problems and first-order descriptions*. Math. Structures Comput. Sci., 1996. 6(6):505–526. (SEGRAGRA '95).

[She75] S. Shelah. *The monadic theory of order*. Ann. of Math. (2), 1975. 102(3):379–419.

[SS89] P. Scheffler and D. Seese. *A combinatorial and logical approach to linear-time computability*. In *EUROCAL '87: Proceedings of the European Conference on Computer Algebra*, volume 378 of *Lecture Notes in Comput. Sci.*. Springer-Verlag, London, UK, 1989 pages 379–380.

[SS02] D. Seese and F. Schlottmann. *Large grids and local information flow as a reason for high complexity*. In G. Frizelle and H. Richards, editors, *Tackling industrial complexity: the ideas that make a difference, Proceedings of the 2002 conference of the Manufacturing Complexity Network*. University of Cambridge, April 2002 pages 193–207.

[ST93] P. D. Seymour and R. Thomas. *Graph searching and a min-max theorem for tree-width*. J. Combin. Theory Ser. B, 1993. 58(1):22–33.

[ST94] P. D. Seymour and R. Thomas. *Call routing and the ratcatcher*. Combinatorica, 1994. 14(2):217–241.

[TNS82] K. Takamizawa, T. Nishezeki, and N. Saito. *Linear-time computability of combinatorial problems on series-parallel graphs*. J. Assoc. Comput. Mach., 1982. 29(3):623–641.

[Tod03] I. Todinca. *Coloring powers of graphs of bounded clique-width*. In *Graph-Theoretic Concepts in Computer Science, October 2003*, volume 2880 of *Lecture Notes in Comput. Sci.*. Springer, 2003 pages 370–382.

[TW68] J. W. Thatcher and J. B. Wright. *Generalized finite automata theory with an application to a decision problem of second-order logic*. Math. Systems Theory, 1968. 2:57–81.

[Val79] L. G. Valiant. *The complexity of enumeration and reliability problems*. SIAM J. Comput., 1979. 8(3):410–421.

[Van04] J.-M. Vanherpe. *Clique-width of partner-limited graphs*. Discrete Math., 2004. 276(1-3):363–374. 6th International Conference on Graph Theory.

[vEB83] P. van Emde Boas. *Dominoes are forever*. Technical Report 83-04, University of Amsterdam, January 1983.

[vLS66] J. H. van Lint and J. J. Seidel. *Equilateral point sets in elliptic geometry*. Nederl. Akad. Wetensch. Proc. Ser. A 69=Indag. Math., 1966. 28:335–348.

[Wag84] K. Wagner. *The complexity of problems concerning graphs with regularities*. Technical Report N/84/52, Friedrich-Schiller-Universität Jena, 1984. Extended abstract appeared in Proceedings of the 11th MFCS. Lecture Notes in Computer Science, Vol. 176, Springer-Verlag, 1984, 544–552.

[Wan94] E. Wanke. *k-NLC graphs and polynomial algorithms*. Discrete Appl. Math., 1994. 54(2-3):251–266.

[Weba] `http://www.dbai.tuwien.ac.at/proj/hypertree/`.

[Webb] `http://wwwinfo.deis.unical.it/~frank/Hypertrees/`.

[Wim87] T. V. Wimer. *Linear algorithms on k-terminal graphs*. Ph.D. thesis, Dept. of Computer Science, Clemson University, South Carolina, 1987.

[Yan81] M. Yannakakis. *Algorithms for acyclic database schemes*. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*. IEEE Computer Society, 1981 pages 82–94.