Fixed-Parameter Tractability of Treewidth and Pathwidth

Hans L. Bodlaender

Department of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
h.l.bodlaender@uu.nl

Abstract. In this survey, a number of results on the fixed-parameter tractability of treewidth and pathwidth are discussed. Some emphasis is placed on older results, and proofs that show that treewidth and pathwidth are fixed-parameter tractable. Also, a linear-time algorithm for testing if a graph has pathwidth at most some given constant is discussed in more detail.

1 Introduction

This overview paper is on the occasion of the 60th birthday of Mike Fellows. Already in the early development of the theory reported here, Mike's insights were at many points of great importance, and his work and his enthusiasm for the topics were always a great source of inspiration. Many of the ideas discussed in this survey were obtained from or inspired by discussions with or talks by Mike Fellows.

Treewidth, and related notions, like pathwidth, branchwidth, cliquewidth, rankwidth play an important role in many modern investigations in algorithmic graph theory, and already from its early origins, in the field of parameterized algorithms. In this survey, a look will be taken at the results that show that the problems to decide if the treewidth or pathwidth of a given graph is at most a given number k are fixed-parameter tractable. This question is an interesting one, for several reasons: the result is used as a subroutine in many recent results, and the investigations for these notions show many important techniques from the field of parameterized algorithms, and often the problem was one of the sources of inspiration for inventing these techniques.

The notions of treewidth and pathwidth were introduced by Robertson and Seymour [110, 113] in their fundamental work on graph minors. However, other, equivalent notions were invented independently, and sometimes earlier by many different authors. Already in the 1960's, it was observed that many problems that are intractable on general graphs become easier to solve on trees and series-parallel graph. Several authors independently noted that these results can be generalized to larger classes of graphs. E.g., Wimer introduced in the 1980's the notion of k-terminal recursive graph classes [143]. Trees can be formed by 'gluing' 1-terminal graphs together; series-parallel graphs by 'gluing' 2-terminal graphs together; and a similar algorithmic behavior is obtained when using some other constant number of vertices. An often used equivalent version of treewidth is the notion of *partial k-trees* by Arnborg et al. [3, 7]. An overview of several notions that are equivalent (or imply a constant bound) to treewidth or pathwidth can be found in [16].

Nowadays, the notion of treewidth plays a role in many different fields of algorithms research and graph theory. One important reason for the interest is that many problems that are intractable (e.g., NP-hard) become linear time (or sometimes polynomial time) solvable when restricted to graphs of bounded treewidth. Such algorithms have been found for many combinatorial problems (see e.g., [8, 9, 33, 94, 138, 144]), and also have been employed for problems from computational biology (see e.g., [100]), constraint satisfaction (see e.g., [40, 47, 78, 94]), and probabilistic networks (see [99]). See e.g., also [3, 2, 6, 12, 32, 39, 38, 42, 50, 70, 79, 80, 82, 85, 88, 106, 105, 108, 145]. In other words: many graph problems become fixed-parameter tractable when parameterized by the treewidth of the input graph.

This survey is further organized as follows. Section 2 gives some definitions, discusses equivalent notions, and some well known lemmas on treewidth and pathwidth. In Section 3, linear-time algorithms for problems on graphs of bounded treewidth are discussed, including the 'automata view' on these algorithms, pioneered by Fellows and Langston. Section 4 discusses some algorithmic consequences of the theory of graph minors. Section 5 looks at the complexity of deciding treewidth and pathwidth, with most emphasis on the fixed-parameter case. It includes a pathwidth version of the result that the fixed-parameter case of treewidth is linear time solvable ([14]). The paper ends with short conclusions and a few major open problems.

For easier presentation, some arguments and proofs have technical inaccuracies, and at some points, an overload of notation seemed unavoidable. I still hope that many of the ideas and techniques come across.

2 Definitions and Equivalent Notions

Throughout this paper, n denotes the number of vertices of graph $G = (V, E)$. Unless otherwise stated, graphs are considered to be simple and undirected. Several of the results generalize to directed graphs, but this will not be elaborated here.

The notions of treewidth and tree decomposition were introduced by Robertson and Seymour [113] in their fundamental work on graph minors.

Definition 1. *A* tree decomposition *of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, with $\{X_i \mid i \in I\}$ a family of subsets of V (called* bags*) and T a tree, such that*

- $\bigcup_{i \in I} X_i = V$,
- *for all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$, and*
- *for all $v \in V$, the set $I_v = \{i \in I \mid v \in X_i\}$ forms a connected subtree of T.*

The width *of tree decomposition* $(\{X_i \mid i \in I\}, T = (I, F))$ *is* $\max_{i \in I} |X_i| - 1$. *The treewidth of a graph G, $tw(G)$, is the minimum width among all tree decompositions of G.*

The third condition in the definition above can be replaced by the following equivalent condition:

For all $i_1, i_2, i_3 \in I$: if i_2 is on the path from i_1 to i_3 in T, then $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$.

An example of a graph with a tree decomposition can be found in Figure 1.

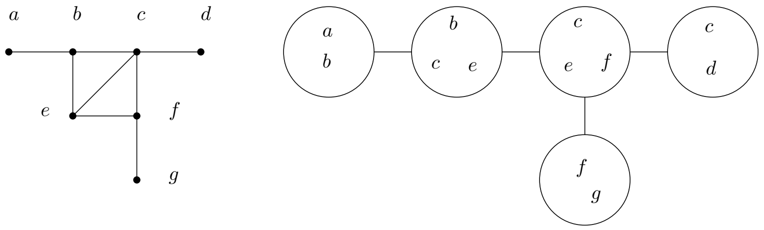

Fig. 1. A graph with a tree decomposition

A *path decomposition* is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ with T a path. The pathwidth of a graph is the minimum width of a path decomposition of G.

There are several equivalent characterizations of the notions of treewidth and pathwidth. For an overview, see e.g. [16]. We mention a few below that are useful for the further exposition of technical ideas.

2.1 Nice Tree and Path Decompositions

A tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is *nice*, if T is a rooted tree, and each node is of one of the four following types:

- **Leaf**: a leaf node i has no children and has $|X_i| = 1$, i.e., a bag size of one.
- **Join**: a join node i has two children j_1, j_2 with $X_i = X_{j_1} = X_{j_2}$, i.e., with the same bags.
- **Forget**: a forget node i has one child j such that there is a $v \in V$ with $X_i = X_j - \{v\}$.
- **Introduce**: a **forget** node i has one child j such that there is a $v \in V$ with $X_i = X_j \cup \{v\}$.

It is well known that one can transform a tree decomposition into a nice one with the same width, and with $O(n)$ nodes, in linear time. One of the first occurrences of nice tree decompositions is in [91].

We similarly have *nice path decompositions.* A nice path decomposition can be represented by a series of bags $(X_0, \ldots, X_r)$ with $r = O(n)$, and with $X_0 = \emptyset$ (the only **leaf** node) and each X_i, $i > 1$ has a vertex v with $X_i = X_{i-1} - \{v\}$ (**forget** nodes) or $X_i = X_{i-1} \cup \{v\}$ (**introduce nodes**). While X_0 is not necessary, using an empty first bag helps for easier notation later on. One can show the following result.

Theorem 1. *Suppose we are given a graph $G = (V, E)$ and a tree (path) decomposition of G, $(\{X_i \mid i \in I\}, T = (I, F))$ of G of width k. Then a nice tree (path) decomposition of G of width k can be found in $O(k(n + |I|))$ time, such that the nice tree decomposition has $O(n)$ bags.*

2.2 *k*-Terminal Graphs

A *terminal graph* is a triple (V, E, X) with $X \subseteq V$ an ordered set of vertices, called the *terminals.* (V, E, X) is a *k-terminal* graph if $|X| = k$. We define the $\oplus$ operation on pairs of k-terminal graphs: $(V, E, X) \oplus (W, F, Y)$ is obtained by taking the disjoint union of (V, E) and (W, F) and then identifying the ith terminal of X with the ith terminal of X with the jth terminal of Y; dropping parallel edges if existing.

For the description of algorithms, it is useful to associate terminal graphs (forming subgraphs of G) with nodes in a nice tree or path decomposition, in the following way. Consider a nice tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$. For each node $i \in I$, we associate a terminal graph $G_i = (V_i, E_i, X_i)$, with V_i the union of all bags X_j with $j = i$ or j a descendant of i; and and $E_i = \{\{v, w\} \in E \mid v, w \in V_i\}$, (taking some arbitrary ordering on X_i).

For a **leaf** node, the graph G_i simply consists of the unique vertex in X_i and no edges.

If i is a **join** node with children j_1 and j_2, then the graph G_i can be obtained from the graphs G_{j_1} and G_{j_2} by taking the disjoint union and then identifying the vertices in $X_i = X_{j_1} = X_{j_2}$, i.e., $G_i = G_{j_1} \oplus G_{j_2}$. An example is given in Figure 2.

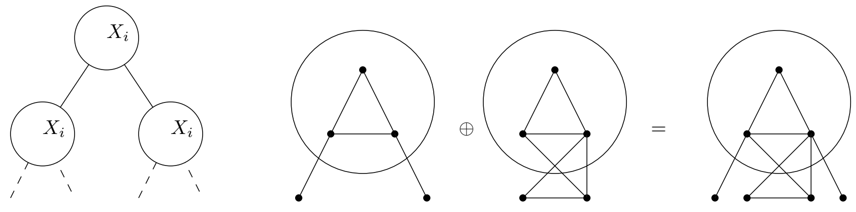

Fig. 2. The $\oplus$*operation*, a **join** node and the corresponding subgraphs

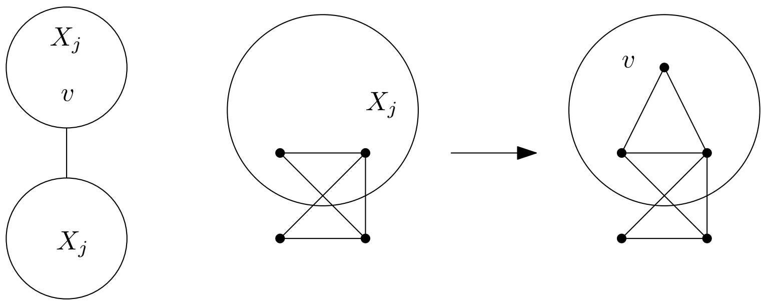

Fig. 3. An **introduce** node and the corresponding subgraphs

If i is an **introduce** node with child j with $X_i = X_j \cup \{v\}$, then G_i is formed from G_j by adding the vertex v and some edges between v and vertices in X_j. See for an example Figure 3.

For a **forget** node, the situation is simple. If i is a **forget** node with child j and $X_i = X_j - \{v\}$, then G_i and G_j have the same vertices and edges; the only difference is that v is no longer a terminal.

2.3 Representing Nice Path Decompositions by Strings

Suppose we consider graphs up to isomorphism, i.e., we ignore vertex names. Then a nice path decomposition of a graph can be represented by a finite string of characters. Assume the vertices in a bag are ordered by the order in which they were introduced. An **introduce** node can now be characterized by the subset of the indices of the neighbors of the introduced vertex. E.g., if $X_i = \{v, w, x\}$ and $X_{i+1} = \{v, w, x, y\}$, with v introduced in a bag with a smaller index than the bag where w is introduced, and w is likewise before x, then if $\{v, y\}$ and $\{x, y\}$ are edges, and $\{w, y\}$ is not an edge in G, then we can characterize node $i + 1$ by the subset $\{1, 3\}$. **Forget** nodes can be characterize by the index of the forgotten vertex. E.g., if in our example, $X_{i+2} = \{v, w, y\}$, then node $i + 2$ can be characterized by the index 3. In this way, we can characterize a nice path decomposition of width at most k by a sequence of at most $2n$ subsets of $\{1, 2, \ldots, k\}$ and elements from $\{1, \ldots, k + 1\}$, i.e., by a string of length at most $2n$ from an alphabet A_k of size $2^k + k + 1$. This representation has some important consequences: several algorithms that exploit (nice) path decompositions can be represented as finite state automata. More on this in Section 3.2.

Similarly, (nice) tree decompositions can be represented as a labeled tree, and several algorithms on (nice) tree decomposition can be represented as a *finite state tree automaton.* These latter are generalizations of finite state automata, but have as input a labeled tree instead of a string. This view was pioneered by Fellows and Langston [65]. Recently, the approach was moved to the notion of rankwidth by Ganian and Hliněný [76].

Not every string in A_k^* represents a graph of pathwidth at most k. A symbol that tells that we forget the ith terminal should only occur when there are at least i terminals; a symbol that tells that we introduce a vertex with edges to terminals with indices in $I \subseteq \{1, \ldots, k\}$ should only occur when we have at most k terminals, and each index in I corresponds to an existing terminal. It is a trivial exercise to see that the set of strings that correspond to a graph of pathwidth at most k is regular, i.e., that we can build a finite state automaton that recognizes this set. Similarly, labeled trees that correspond to tree decompositions of width k can be recognized by a finite state tree automaton.

2.4 Notions Equivalent to Pathwidth

Two other notions that are equivalent with pathwidth are the following. A linear ordering of a graph $G = (V, E)$ is a bijective function $f : V \to \{1, \ldots, n\}$.

Definition 2. *The vertex separation number of a linear ordering f of a graph G equals*

$$\max_{v \in V} |\{w \in V \mid f(w) \leq f(v) \wedge \exists \{w, x\} \in E : f(x) > f(v)\}$$

The vertex separation number of a graph G equals the minimum vertex separation number of a linear ordering of G.

Theorem 2 (Kinnersley [89]). *The vertex separation number of a graph G equals the pathwidth of G.*

We also have the following folklore result. For a proof, see e.g. [16].

Theorem 3. *Let G be a graph. The pathwidth of G is at most k, if and only if $G = (V, E)$ is a subgraph of an interval graph $H = (V, F)$ with maximum clique size at most $k + 1$.*

2.5 Minors

Another important notion for the theory of treewidth is the notion of *minor*, see e.g., [110]. A graph H is a minor of a graph G, if H can be obtained from G by zero or more of the following operations: removing vertices, removing edges, and contracting edges (an edge contraction replaces two adjacent vertices by one that is incident to the neighbors of the contracted vertices).

More on graph minors in Section 4.

2.6 Cliques

A folklore result on treewidth is often of great help. As observed in [28], it directly follows from the Helly property for trees.

Lemma 1. *Let $(\{X_i \mid i \in I\}, T = (I, F))$ be a tree decomposition of $G = (V, E)$ and let $W \subseteq V$ be a clique in G. Then there exists an $i \in I$ with $W \subseteq X_i$.*

3 Algorithms on Tree and Path Decompositions

One of the most important reasons for the interest in the notion of treewidth (or its related notions) its that many problems become polynomial, and often linear solvable on graphs with some constant upper bound on their treewidth. See e.g., [8, 9, 94, 138, 144].

Most of these algorithms employ dynamic programming in some form. These algorithms consist of two steps. In the first step, a tree decomposition of bounded width is found. This step will be discussed in more detail in Section 5. The tree decomposition then is transformed to a nice one with the same width, with a linear number of nodes, cf. Section 2. In the second step, the (nice) tree decomposition is exploited: in some bottom-up order (e.g., postorder), a table is computed for each node of the tree. To compute a table for a node, all what is (usually) needed is the information of the nodes of its children and a little "local" information (e.g., what vertices in the bag of the node are incident). The problem then can be decided using the table of the root. Construction versions often can be solved by going top-down in the tree, using the information stored in the tables.

Our example of the algorithm uses the 3-COLORING problem.. A 3-coloring of a graph $G = (V, E)$ is a function $c : V \to \{1, 2, 3\}$ such that for all $\{v, w\} \in E$, $c(v) \neq c(w)$. In the 3-COLORING problem, we are given a graph $G = (V, E)$, and have to decide if there exists a 3-coloring of G.

3.1 Solving 3-Coloring on Nice Tree Decompositions

For the 3-Coloring problem, we compute for each node in the tree decomposition $i \in I$, a table A_i. The table has an entry for each function $f : X_i \to \{1, 2, 3\}$. The entry maps to a Boolean value, and expresses if the function f can be extended to a 3-coloring of G_i. I.e., $A_i(f)$ is true, if and only if there exists a 3-coloring g of G_i such that for all $v \in X_i$, $f(v) = g(v)$.

Proposition 1. *If $G = (V, E)$ is given with a nice tree decomposition of width at most k, and with $O(n)$ nodes, then the 3-coloring problem on G can be solved in $O(3^k n)$ time.*

Proof. We discuss for each of the four types of nodes: **leaf**, **introduce**, **forget**, **join** how the table A_i can be computed, given such tables of the children of i, in $O(3^k)$ time. The algorithm then is as follows: in postorder, we compute for each node of the nice tree decomposition the table A_i. In $O(3^k n)$ time we thus have the table A_r for the root r of the nice tree decomposition. Finally, note that G_r equals G, and thus, G has a 3-coloring, if and only if at least one entry in G_r is true. So, we end the algorithm by inspecting A_r for a value true.

Computing A_i for a **leaf** node i is trivial. Recall that G_i just has one vertex and no edges; each of the three possible colorings of this vertex corresponds to a true entry in the table A_i.

Suppose i is an **introduce** node with child j with $X_i = X_j \cup \{v\}$. Consider a coloring $c : X_i \rightarrow \{1, 2, 3\}$. Let c' be the restriction of c to X_j. It is not difficult to see that we have:

$$A_i(c) \Leftrightarrow A_j(c') \wedge \forall w \in X_j : \{v, w\} \notin E \vee c(v) \neq c(w).$$

Suppose i is a **forget** node with child j with $X_i = X_j - \{v\}$. Now we have for all colorings $c : X_i \rightarrow \{1, 2, 3\}$, that $A_i(c)$ is true, iff there is a coloring c' of X_j with $A_j(c')$ true and c is the restriction of c'.

For a **join** node i with children j' and j'', we have for each $c : X_i \rightarrow \{1, 2, 3\}$, that $A_i(c) = A_{j'}(c) \wedge A_{j''}(c)$.

Correctness can easily be derived. In each case, the way G_i is obtained from the graphs associated with the children of i is used; see the discussion in Section 2. It is also easy to see that the time to compute a table is linear in its size. □

Designing an algorithm of the type given above follows a number of steps:

– What information should be stored at a table of a node? This information characterizes the subgraph G_i. Often, the notion of a *partial solution* is used; each partial solution has a characterization, and we tabulate the different characterizations. In case of optimization problems, one can assign costs to partial solutions, and then tabulate for each characteristic the minimum or maximum cost of a partial solution with this characteristic. For an example of the latter, see our discussion of the DOMINATING SET problem. In the case of the 3-COLORING problem, a 3-coloring of G_i is a partial solution, which is characterized by the colors given to the vertices in X_i. A value true implies that there is a partial solution with this characteristic.
– Design for each of the four types of nodes (**leaf**, **introduce**, **forget**, **join**), an algorithm that computes the table for the node, given the tables of the children.
– Show that the answer to the problem can be derived from the table for the root r, using that $G = G_r$.

The second step is not always necessary: Fellows and Langston [65] introduce the *Myhill-Nerode* perspective of algorithms on tree decompositions. We discuss this briefly in the next section.

3.2 Dynamic Programming and Finite State Automata

In this section, we look at the algorithm from a perspective, first introduced by Fellows and Langston [65], namely, we view the algorithm as running on a finite state automaton or finite state tree automaton. For an easier exposition, we consider the algorithm as running on a path decomposition of bounded width. The discussion can be extended to tree decompositions. When using path decompositions, our algorithm corresponds to a finite state automaton; when using tree decompositions, this generalizes to a *finite state tree automaton.*

Consider the algorithm that was given in the previous section. We assume it runs on a path decomposition of width k, with k a constant; i.e., we do not have **join** nodes. For each node in the path decomposition, we computed a table. To denote a table, we need a constant number of bits, i.e., there the number of possible tables is a constant (only depending on the width of the path decomposition.)

As discussed earlier, we can represent a nice path decomposition of width k by a string in an alphabet whose size is bounded by a function of k $(2k + k + 1)$. Now, for a bag, the table that is computed by the algorithm for that bag only depends on the table of the previous bag, whether the bag is an **introduce** or **forget** node, and which vertex is forgotten, or what incidences there are to the **introduce** node. Thus, the table depends on the previous table and the 'character' of the bag.

Thus, we can view the algorithm as a finite state automaton: each possible table corresponds to a state of the automaton, and the next state only depends on the previous state and the character. The table for the last bag decides if the input is accepted or rejected.

Many dynamic programming algorithms on path decompositions can be seen as finite state automata: the main ingredients are that tables must have a number of bits that is a function of the width, and that tables only depend on the previous table and the type of bag, as discussed above. Algorithms on tree decompositions can be viewed in a similar way as *finite state tree automata.*

This way of viewing algorithms as automata has important consequences: several classic results of automata theory can be used. For instance, it is decidable whether two finite state automata recognize the same set of strings, and thus, if we have two dynamic programming algorithms of the proper form, we can determine if these give the same output for all graphs of pathwidth at most k. Some corollaries of this will be discussed later.

3.3 Finite Index

When designing dynamic programming algorithms for problems on graphs, usually the first step ("what should we store in tables") is the most important. When tables have a constant number of bits, this step gives us equivalence relations on k-terminal graphs (for each k).

Suppose we have a decision problem Q on graphs. Let $\sim_{Q,k}$ be the equivalence relation on k-terminal graphs, with for all k-terminal graphs G, H, $G \sim_{Q,k} H$, if and only if for all k-terminal graphs K, $Q(G \oplus K)$, if and only if $Q(H \oplus K)$.

Suppose we have a dynamic programming algorithm A, that runs in $f(k)n$ time when given a tree decomposition of width k, and each table has $O(1)$ bits. Let $\sim_{A,k}$ be the equivalence relation on k-terminal graphs, with $G \sim_{A,k} H$ if when the table that is computed by A when G is the k-terminal subgraph associated with a bag equals the table that is computed when H is the k-terminal subgraph associated with a bag. Now, by closely observing the working of the dynamic programming algorithm, one can observe that the output of A will be the same for $G \oplus K$ as for $H \oplus K$, for any K, and thus $\sim_{A,k}$ is a refinement of $\sim_{Q,k}$.

We say that Q is *finite index*, if for each k, $\sim_{Q,k}$ has a finite number of equivalence classes. Now, the famous Myhill-Nerode theorem for regular languages tells us that if Q is finite index, then Q is regular. In particular, the theorem tells us that when we have established an equivalence relation that is (a refinement of) $\sim_{Q,\ell}$ for all $\ell \leq k$, then from this, we can derive the finite state automaton, i.e., a dynamic programming algorithm for graphs of pathwidth at most k. As the Myhill-Nerode theorem also holds for tree automata, we obtain the same result for graphs of treewidth at most k.

This has two consequences: it confirms the intuition that the design of the equivalence relation is the important step in the design of the algorithms that run on path or tree decompositions, and it allows us to avoid in several cases the design by hand of the procedures that tell how to compute tables for **join**, **introduce** and **forget** nodes.

3.4 Courcelle's Theorem

As said, for many problems, linear-time algorithms have been found for the problems restricted to graphs of bounded treewidth. Often, constructing such algorithms means to pay attention to many details. Fortunately, there are also algorithmic meta-theorems, that allow us to establish for a large number of problems the existence of linear-time algorithms when restricted to graphs of bounded treewidth. By far the most important of these algorithmic meta-theorems is *Courcelle's theorem.*

Theorem 4 (Courcelle [42]). *For each problem P, that can be formulated in Counting Monadic Second Order Logic, there exists an algorithm that decides P on a given graph G, and that uses linear time for graphs of treewidth bounded by some constant.*

Counting Monadic Second Order Logic (CMSOL) is a language in which we can express properties of graphs. The simpler version of Monadic Second Order Logic (MSOL) has the following elements: tests if a vertex is incident with an edge ($v \in e$), tests if two vertices are adjacent ($\{v, w\} \in E$), tests is a vertex (edge) is an element of a vertex (edge) set ($v \in W$, $e \in F$), Boolean operations ($\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\ldots$), equality of variables, quantification over vertices and edges ($\exists v \in V$, $\exists e \in E$, $\forall v \in V$, $\forall e \in E$), and quantification over vertex and edge sets ($\exists W \subseteq V$, $\exists F \subseteq E$, $\forall W \subseteq V$, $\forall F \subseteq E$). CMSOL has in addition operations that decide if the size of a set modulus some constant equals another constant, i.e., for constants c_1 and c_2, the language has expressions $|W| \bmod c_1 = c_2$ and $|F| \bmod c_1 = c_2$.

For example, the property that G is bipartite can be expressed as:

$$\exists W \subseteq V : \forall e \in E : \exists v \in V : \exists w \in V : v \in e \wedge w \in e \wedge v \in W \wedge \neg(w \in W)$$

Many well known and important graph properties, including many NP-hard properties, can be expressed in CMSOL. Besides an alternative proof of Courcelle's theorem, Borie et al. [32] show how to express many graph properties is CMSOL. See also [92] for a different proof that gives better constant factors.

Several extensions to Courcelle's theorem have been found. An important one allows us to obtain linear-time algorithms for many optimization problems restricted to graphs of bounded treewidth. Consider a CMSOL property P with one free vertex or edge set variable. The problems to find a minimum size set of vertices W or edges F such that $P(W)$ or $P(F)$ holds can also solved in linear time for graphs of bounded treewidth; the same holds when we want to find such a set of maximum weight, or if weighted variants are considered. See e.g., [6, 32, 31, 33, 45].

Another important variant is the result by Courcelle et al. [44] who show that a similar result holds for graphs of bounded cliquewidth for CMSOL without edge set quantifications. As bounded cliquewidth is equivalent (with different bounds) to bounded NLC-width, bounded rankwidth, or bounded booleanwidth, we have for each of these graph measures many problems that can be solved in linear or polynomial time when they have bounded 'width'.

The 'automaton view' also helps to see another result by Courcelle: for each graph property P in CMSOL and integer k, it is decidable if all (or no) graphs of treewidth (or pathwidth) at most k fulfill property P. The main idea of the proof (sketched here for the case of pathwidth) is the following: build the finite state automaton for path (tree) decompositions of width at most k. Also, build the finite state automaton that checks if a sequence of bag types represents a possible path decomposition (cf. the discussion in Section 2). Now use Myhill-Nerode theory to check if these two automata accept the same set of strings.

Theorem 5. *Let P be a property in CMSOL, and k be an integer. It is decidable whether all graphs of pathwidth (treewidth) at most k have property P, and whether no graphs of pathwidth (treewidth) at most k have property P.*

3.5 Courcelle's Conjecture

Courcelle's theorem (Theorem 4) shows that expressibility in CMSOL implies finite index. Courcelle conjectured that the reverse also holds. (See also e.g., [43].) Proofs of the conjecture for special cases were obtained by Kabanets [86] (graphs of bounded pathwidth) and Kaller [87] (graphs of treewidth 3 and k-connected graphs of treewidth k). In 1998, Lapoire [98] announced a proof for the conjecture, but a refereed full version of the proof has not been published.

3.6 Running Times as Function of Pathwidth and Treewidth

For many problems, Courcelle's theorem gives a relatively fast way of establishing that the problem is fixed-parameter tractable with respect to treewidth, i.e., that there is an algorithm that solves the problem in linear time for graphs of bounded treewidth. The constant factors of such algorithms will however be large, and better constant factors can often be obtained when designing tailor-made algorithms for specific problems.

For some problems, the running time can be improved with help of additional techniques. One of these was introduced by van Rooij et al. [142], see also [30].

Here, a generalization of fast subset convolution is used to speed up algorithms on tree decompositions, in particular the **join** operation. The main idea is the following: the information stored in a table in the dynamic programming algorithm can often be represented in different ways. Some of these allow for a faster **join** operation, while others allow for faster **introduce** (or **forget**). With help of fast subset convolution or generalizations of it, one can quickly transform a table in one representation to its equivalent table in the other representation. Tables are again computed in postorder, i.e., bottom-up, but when necessary, the representation is changed.

Very recently, Cygan et al. [46] introduced a new technique that speeds up several computations on tree decompositions, which they call *Cut and Count.* Here, algorithms on tree decompositions are made faster by using a *randomized* approach. In this way, Cygan et al. [46] obtain randomized algorithms whose dependence on the width of the given tree decomposition is only single exponential, (i.e., of the form $O^*(c^k)$ for some constant c) while the known 'classic' dynamic programming algorithms have a running time $\Theta^*(2^{k \log k})$ or worse for these problems.

3.7 Lower Bounds

For a number of problems, there are also lower bounds known (for the dependency of the running time on the treewidth). Lokshtanov et al. [104] have shown such lower bounds for a large number of problems. For instance, consider the 3-COLORING problem. We have seen that this problem can be solved in $O(3^k n)$ time; Lokshtanov et al. [104] prove that there exists no algorithm that uses $(3-\epsilon)^{tw(G)} n^{O(1)}$ time for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis [84, 48] does not hold. Other problems where the known upper bound matches this type of lower bound include DOMINATING SET, q-COLORING for constant q; INDEPENDENT SET. See also [46].

3.8 Special Classes of Graphs

Efficient algorithms for graphs of bounded treewidth can also help to obtain fast(er) algorithms for problems on special types of graphs. Two important examples of this are the *planar* graphs and *graphs of bounded degree.*

Planar graphs have treewidth $O(\sqrt{n})$. The fact can be shown to follow from the Lipton-Tarjan planar separator theorem [102, 103]; and vice versa, the planar separator theorem can be obtained as corollary from the fact that planar graphs have treewidth $O(\sqrt{n})$, see [16]. Fomin and Thilikos [74] showed that the treewidth of a planar graph is bounded by $3.182\sqrt{n}$, and also showed that the branchwidth of a planar graph is at most $2.122\sqrt{n}$.

As a consequence, for many graph problems, there are $O(c^{\sqrt{n}})$ algorithms, and sometimes $O(c^{\sqrt{n}\log n})$ time algorithms when the inputs are restricted to planar graphs. An example is 3-COLORING, see Section 3.1.

For several problems, dynamic programming as discussed above leads to algorithms that use $O(c^{\sqrt{n}\log n})$ time. With help of additional arguments, algorithms that use $O(c^{\sqrt{n}\log n})$ time can be obtained for several problems on planar graphs, like Hamiltonian Circuit. One can either exploit planarity (leading to an analysis with Catalan structures), see e.g., [53, 52]; or use the probabilistic approach by Cygan et al. [46] which was discussed above. Other algorithms for planar graphs that exploit treewidth (or the related notion of branchwidth) can be found in e.g., [74, 95, 136].

For graphs of bounded degree, we have the following theorem by Fomin et. al. [71].

Theorem 6 (Fomin et al. [71]). *For $\epsilon > 0$, there exists an n_ϵ, such that for all graphs G with $n \geq n_\epsilon$ vertices of which n_3 have degree 3, n_4 have degree 4, n_5 have degree 5, n_6 have degree 6, and $n_{>6}$ have degree more than 6, the pathwidth of G is at most*

$$\frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{>6} + \epsilon \cdot n$$

The result can in several cases be used to obtain faster exact (exponential time) algorithms for graph problems (i.e., 'problems parameterized by the number of vertices n'), see e.g., [71, 141]. Kneis et al. [93] showed that graphs have pathwidth at most $m/5.769 + O(\log n)$, m the number of edges. This also has several algorithmic consequences, e.g., faster exact algorithms for sparse graphs for Max Cut and for Max 2SAT.

4 Graph Minors

In this section, we briefly review a few results from graph minor theory, with some emphasis on its role for the theory of treewidth and related notions. For more extensive overviews, see e.g., [10, 75, 112], or [54, Chapter 7].

In a long series of papers [110, 113, 111, 117, 114–116, 119, 118, 120, 122–126, 128, 127, 129–132, 121, 133], Robertson and Seymour obtained a number of important and fundamental results on graph minors. The central result is the graph minor theorem. (A graph G is minor minimal in a set of graphs if no other graph in the set is a minor of it. Isomorphic graphs are considered to be identical.)

Theorem 7 (Robertson and Seymour). *Any set of graphs has a finite number of minor-minimal elements.*

Equivalent to Theorem 7 is the following.

Theorem 8 (Robertson and Seymour [128]). *Let $\mathcal{G}$ be a collection of graphs that is closed under taking minors. Then there exists a finite set ob($\mathcal{G}$), called the* obstruction set *of $\mathcal{G}$, such that for each graph G, we have that $G \in \mathcal{G}$, if and only if there is no graph $H \in$ ob($\mathcal{G}$) that is a minor of G.*

Theorem 8 has important algorithmic consequences. Several such results were established in the 1980's and 1990's by Fellows and Langston, see e.g., [62, 64, 66, 63, 60, 67, 68]. As also for fixed graphs H, testing if H is a minor can be done in $O(n^3)$ time [124], there exists for each set of graphs that is closed under taking of minors an $O(n^3)$ time membership test. This result however is non-constructive: we know that the algorithm exists but do not know the algorithm itself, as we may not know the obstruction set.

For graphs of bounded treewidth, faster algorithms exist: for a fixed H and fixed integer ℓ, there is a (dynamic programming) algorithm that tests in linear time whether H is a minor of an input graph G, given with a tree decomposition of width at most ℓ. Combined with the result discussed in Section 5.6, we have that each class of graphs that is closed under minors and has bounded treewidth can be recognized in linear time. Then, we use the following result.

Theorem 9 (Robertson et al. [114, 134]). *For each planar graph H, there is a constant c_H, such that each graph G that does not have H as a minor has treewidth at most c_H.*

(A similar result bounds the pathwidth of graphs that do not have some fixed forest H as a minor [110, 11].) Thus, any minor closed class of graphs that does not include all planar graphs has a linear-time recognition algorithm. This result, however, is again non-constructive.

Theorem 10. *Let $\mathcal{G}$ be a class of graphs that is closed under taking of minors. Suppose we can construct a dynamic programming algorithm on tree decompositions of bounded width, that uses $O(1)$ bits per table/node for the problem to recognize graphs in $\mathcal{G}$. Suppose an integer k is known such that all graphs in $\mathcal{G}$ have treewidth at most k. Then the obstruction set of $\mathcal{G}$ is computable.*

Proof. The result follows from the Myhill-Nerode perspective, as discussed in Section 3.2, as introduced by Fellows and Langston [65]. The dynamic programming algorithm on tree decompositions corresponds to a finite state tree automaton. For each finite set of graphs $\mathcal{Z}$, the property that an input graph G has no graph from $\mathcal{Z}$ as a minor can be formulated in monadic second order logic (see e.g., the discussion in [32]) and thus, by Courcelle's theorem (Theorem 4), we can construct a tree automaton that gets as input the representation of a nice tree decomposition of width at most k, and tests whether G has a tree decomposition of width at most k. We now can decide, using a tree automaton equivalent of the classic Myhill Nerode theorem for finite state automata, whether the two machines accept the same language. Thus, for each finite set of graphs, we can check if this is the obstruction set of $\mathcal{G}$. By enumerating all finite sets of graphs, we eventually find the obstruction set. □

See also e.g., [37].

5 Deciding Treewidth and Pathwidth

In this section, we discuss the problems, for fixed integers k, to decide for a given graph $G = (V, E)$ whether its treewidth is at most k. We also look at the constructive variant: if the answer is yes, the algorithm also has to output a tree decomposition of width at most k, and we consider the variants of this problem where we consider pathwidth and path decompositions instead.

The problem to determine for a given graph G and integer k, whether the treewidth of G is at most k is NP-complete [4]. The NP-completeness proof of Arnborg et al. [4] shows that treewidth is NP-complete for co-bipartite graphs, i.e., graphs that are obtained by adding some edges between vertices in two cliques. They also show that for these graphs, the treewidth equals the pathwidth, and thus obtain also the NP-completeness of pathwidth. An independent NP-completeness proof of pathwidth (or, more precisely, for a notion equivalent to pathwidth) was found by Lengauer [101].

In the remainder of this section, we consider the fixed-parameter cases for Treewidth and Pathwidth.

5.1 Membership in XP

Downey and Fellows [54] define the class XP, as the class of parameterized problems that are solvable in time $O(n^{f(k)})$ for some function f.

The result that Treewidth belongs to XP dates from far before the terminology. In the 1980s, Arnborg et al. [4] give a clever dynamic programming algorithm for Treewidth that uses $O(n^{k+2})$ time. The first algorithm whose running time is in XP for Pathwidth was found by Ellis et al.; this complicated algorithm (formulated on the equivalent notion of vertex separation number) only appears in a technical report in 1987 [58]. Both algorithms solve the constructive versions of the problem, i.e., they also give tree or path decompositions of width at most k, if existing.

5.2 Nonconstructive Advances

The fact that Treewidth is fixed-parameter tractable was first obtained as a consequence from the work of Robertson and Seymour on graph minors. We briefly discuss how the results discussed in Section 4 show that Treewidth and Pathwidth are (non uniform) fixed-parameter tractable. We use Pathwidth as running example.

Lemma 2. *For each fixed k, the class of graphs with pathwidth at most k is closed under minor taking.*

Proof. Suppose H is a minor of G, and G has pathwidth at most k. Consider a path decomposition of G of width at most k. Consider the sequence of operations that shows that H is a minor of G. For a deletion of a vertex v, we remove v from all bags of the path decomposition. For a deletion of an edge, we do nothing. For the contraction of an edge $\{v, w\}$ to a vertex x, we replace each occurrence of v and/or w in a bag by an occurrence of x. As a result, we obtain a path decomposition of H of the same or smaller width. □

Thus we have by the results in Section 4:

Proposition 2. *For each fixed k, there is an $O(n^3)$ time algorithm, that given a graph, tests if G has pathwidth at most k.*

The result can be improved in several ways: the cubic time can be brought back to linear time. But also: this algorithm is *non-uniform*, and *non-constructive* in two ways: it does not provide a corresponding path decomposition, and we do not have the algorithm itself: as the proof of Theorem 8 is non-constructive, we know that the obstruction set and thus the algorithm exists, but we do not know this set and thus this algorithm (so far). In later parts of this section, we will overcome these points.

To speed up the algorithm, we can use the fact that the treewidth of graphs is bounded by its pathwidth, and that we can formulate for each fixed graph H, the property that H is a minor of a given graph $G = (V, E)$ in monadic second order logic. Thus, by Courcelle's theorem (see Section 3.4), we have that for fixed k, there exist a linear-time algorithm, that given a tree or path decomposition of bounded width of the input graph G, tests if the pathwidth of G is at most k, by verifying whether G contains any of the graphs from the obstruction set of graphs of pathwidth at most k as a minor.

To find such a tree or path decomposition, one could either use an approximation algorithm for treewidth (or pathwidth); such an algorithm should use time that is polynomial in n but can be exponential in k. The first such algorithm was given in terms of branchwidth and branch decompositions by Robertson and Seymour in [124]: this algorithm finds in $O(3^{3k}n^2)$ time a branch decomposition of width $3k$. This result can easily be transferred to a similar result giving tree decompositions (with factor 4.5 instead of 3 for treewidth). Similar results with some improvements in bounds or running times were obtained by different authors, see e.g., [1, 51, 96, 109] or [90, Sec. 10.5]. Reed [109] obtained a running time of $O(n \log n)$.

Further speedup can be obtained with different methods, which will be discussed later.

5.3 Fighting Non-constructiveness: Self-reduction

One approach to overcome non-constructiveness is by the use of self-reduction. Fellows and Langston [69] (see also [36]) introduced a general technique to turn a non-constructive proof of the existence of an algorithm into a constructive one. We showcase the technique by using the pathwidth problem as example.

Self-reduction is a well known technique to turn algorithms for decision problems into algorithms for the constructive version of the problem: by running the decision algorithm multiple times on slightly modified inputs, the output for the constructive version is generated (e.g., we construct a certificate for a problem in NP.) In the approach of Fellows and Langston, the technique is taken one step further: besides constructing the certificate (in this case, a path decomposition of width at most k, or, equivalently, an interval supergraph with maximum clique size at most $k + 1$), but we also circumvent the fact that we do not know the obstruction set in advance.

First, suppose we have a decision algorithm A for the problem to test for a given graph G if the pathwidth of G is at most k. First, run A on G. If A tells that the pathwidth of G is more than k, we halt. Otherwise, we use $O(n^2)$ runs of A to build an interval graph H with maximum clique size k. (Recall Theorem 3.) Take an auxiliary graph H, which we initially set to be equal to G. Now, for each pair of disjunct vertices $v, w \in V$, $\{v, w\} \notin E$, we test if the pathwidth of the graph, obtained by adding $\{v, w\}$ to H is at most k. If so, we add the edge $\{v, w\}$ to H. Call this algorithm B. The output of algorithm B is a maximal supergraph H of of G that has pathwidth at most k; more specifically, this graph H is an interval graph with maximum clique size $k + 1$.

Suppose we have a set of graphs X that is a subset of the obstruction set of the graphs of pathwidth at most k. We build an algorithm C that, given a graph G, either decides that the pathwidth of G is at most k, or gives a path decomposition of G of width at most k, or decides that X is a *proper* subset of the obstruction set of graphs of pathwidth at most k, as follows: run the following modification of algorithm B: instead of using A, we test if the input graph has a minor in X. If this algorithm tells that G has pathwidth more than k, then this is because a graph from X is a minor of G, and thus this is a correct output. Otherwise, we check if the output is indeed an interval graph with maximum clique size k. (This can be done in polynomial time, see e.g. [77].) If so, we are done; if not, we know that X was not equal to the obstruction set of graphs of pathwidth at most k.

We now can build an algorithm D, that given a graph G, either correctly decides that G has pathwidth more than k, or outputs a path decomposition of G of width at most k, as follows. Initially, let X be the empty set. Now, repeat the following step, until we are done. Enumerate all graphs G, and for each, test if G is not in X, and if G is a member of the obstruction set of graphs of pathwidth at most k, i.e., if the pathwidth of G is $k + 1$ and if each proper minor of G has pathwidth at most k. (We can use any algorithm for this.) If not, continue the enumeration of graphs. If the test succeeds, add G to X; stop the enumeration of graphs, and run algorithm C with X. If algorithm C produces as output that G has pathwidth more than k, or a path decomposition of G, we are done; otherwise, we restart the enumeration of graphs, but now with the larger set X.

This is an fpt-algorithm, i.e., its running time is bounded by a function of k times a polynomial in n: we never have to enumerate graphs beyond the last graph in the obstruction set of pathwidth-k graphs, and thus X and the time for enumeration of graphs are bounded by a function of k. The algorithm is, of course, highly impractical, but showcases an important idea how we can turn non-constructive algorithms into constructive ones.

With some addition techniques, one can modify this algorithm such that it runs in $O(f(k)n^2)$ time, see [13]. The technique works for a large number of problems; see [69] for more details. For pathwidth, there exist more efficient algorithms, which will be discussed in later sections.

5.4 Graph Reduction Techniques

In this subsection, we discuss some algorithmic results for graphs of bounded treewidth that are based on the technique of graph reduction, now known under the name of *protrusions*. A simple example of this technique is the following algorithm that recognizes the graphs of treewidth at most one, i.e., the set of forests: while possible, remove vertices of degree one with their incident edge and vertices of degree zero. The empty graph results, if and only if the input graph was a forest.

If we add the reduction rule that removes vertices of degree two while adding an edge between their neighbors (if not already present), we obtain a recognition algorithm for graphs of treewidth at most two. Arnborg and Proskurowski gave a fast reduction algorithm for graphs of treewidth at most three [7], see also [107]. For treewidth 4, Sanders [135] found a linear-time recognition algorithm. An experimental evaluation of this algorithm by Hein and Koster [83] shows that this algorithm is practical.

In a more generalized setting, consider the equivalence relation $\sim_{Q,k}$ discussed in Section 3.2 for some decision problem Q on graphs. If we have k-terminal graphs G_1 and G_2 with $G_1 \sim_{Q,k} G_2$ and G_2 is smaller than G_1, then this leads to the following algorithmic step: if we have G_1 as subgraph, with terminals of G_1 the only vertices in the subgraph with neighbors outside the subgraph, then we can replace G_1 by G_2; i.e., we transform $G = G_1 \oplus H$ to $G_2 \oplus H$. As $Q(G) = Q(G_2 \oplus H)$, the step is *safe*, as the answer to the problem at hand does not change.

A graph reduction algorithm can thus be based on a collection of such *safe reduction rules*. In 1993, Arnborg et al. [5] showed that for each fixed k, each graph problem that is finite state (and thus, including, all problems that can be formulated in monadic second order logic) there is a collection of reduction rules that give a linear time (on a random access machine with the uniform cost model) algorithm for graphs of treewidth at most k. Bodlaender and Hagerup [22] showed that one can obtain parallel algorithms based on graph reduction that use $O(\log n)$ time and $O(n)$ work; their version leads to linear-time sequential algorithms on the more standard pointer machine model. Bodlaender and van Antwerpen-de Fluiter [29, 49] showed that the technique can also be applied to some optimization problems (terming these *finite integer index*); a reduction rule not only changes the graph, but also adds a constant to one integer variable.

Graph reduction techniques are often used for preprocessing and kernelization. For the problem to determine the treewidth of a graph, graph reduction has been used in the setting of preprocessing [27, 56] and, recently, in the setting of kernelization [23]. Recently, graph reduction techniques were used to obtain kernelization results for other problems, including 'meta-kernelization' results: proofs that large collections of problems have kernelization algorithms when restricted to certain special graph classes (e.g., graphs embeddable on a fixed surface) [20, 72, 73].

A very recent (spring 2012) result by Drucker [55], combined with the kernelization lower bound techniques from [17], shows that TREEWIDTH (in its standard parameterization) does not have a polynomial kernel, unless $NP \subseteq coNP/poly$.

5.5 An Explicit Finite Congruence

The use of non-constructive methods can be avoided altogether by giving an explicit equivalence relation on path decompositions for certain types of subgraphs. The techniques can be generalized for treewidth and tree decompositions; we briefly discuss what additional technical difficulties are to be faced at the end of this section.

The results shown here were obtained by Bodlaender and Kloks [24] and Lagergren and Arnborg [97] in 1991; Fellows and Langston obtained similar results independently at the same time. Bodlaender et al. [19] discussed how such algorithms can be automatically be derived, and part of the discussion below is based on the ideas from [19].

Theorem 11. *Let k, ℓ be constants. There is (and we can explicitly describe) an algorithm, that given a graph $G = (V, E)$ with a path decomposition of G of width at most ℓ, decides if the pathwidth of G is at most k, and if so, finds a path decomposition of G of width at most k.*

Of course, we may assume that $k < \ell$, otherwise the problem is trivial.

We define a simple operation on sequences of integers, which we call *compacting*: if $(a_1, \ldots, a_q)$ is an sequence of integers, its *compacted* sequence is obtained by repeating the following step:

- If there are $i, j, j \geq i+2$, such that $a_i = \min_{i \leq i' \leq j} a_{i'}$ and $a_j = \max_{i \leq i' \leq j} a_{i'}$, then remove the numbers $a_{i+1}, \ldots, a_{j-1}$ from the sequence.
- If there are $i, j, j \geq i+2$, such that $a_i = \max_{i \leq i' \leq j} a_{i'}$ and $a_j = \min_{i \leq i' \leq j} a_{i'}$, then remove the numbers $a_{i+1}, \ldots, a_{j-1}$ from the sequence.

E.g., the compacted sequence of $3, 5, 7, 4, 2, 6$ is $3, 7, 2, 6$. The compacted sequence is unique, i.e., it does not depend on the order in which the steps are carried out.

Recall that pathwidth is equivalent to vertex separation number (Theorem 2.)

The *uncompacted fingerprint* of a linear ordering f of a terminal graph (V', E_i, X) is defined as follows. We partition f in *pieces* as follows: the first piece starts with the first vertex in the ordering, $f^{-1}(1)$. Now, visit the vertices from low to high number. Start a new piece when we see a terminal, i.e., a vertex in X, and start a new piece directly after a vertex that is the highest numbered neighbor of a vertex in X. We have for each piece an uncompacted fingerprint, and the uncompacted fingerprint of f is the sequence of uncompacted fingerprints of the pieces: Suppose we have the piece $f^{-1}(i), f^{-1}(i+1), \ldots, f^{-1}(j)$. The first part of the uncompacted fingerprint is $X \cap \{f^{-1}(i)\}$, i.e., it tells whether the first vertex is a terminal and if so, what terminal; the second part is the sequence $n_i, n_{i+1}, \ldots, n_j$, with $n_r = |\{w \in V \mid f(w) \leq r \wedge \exists \{w, x\} \in E : f(x) > r\}|$.

The *compacted fingerprint* is obtained by taking the uncompacted fingerprint and then compacting in each piece its sequence of numbers.

Lemma 3. *Let f and g be linear orderings of ℓ-terminal graph (V', E', X), and let H be an ℓ-terminal graph, and $G = (V', E', X) \oplus H$. Let k be an integer.*

(i). Suppose f and g have the same uncompacted fingerprints. There exists a linear ordering of G with vertex separation number at most k that contains f as a subsequence, if and only if a linear ordering of G with vertex separation number at most k that contains g as a subsequence.

(ii). Suppose f and g have the same compacted fingerprints. There exists a linear ordering of G with vertex separation number at most k that contains f as a subsequence, if and only if a linear ordering of G with vertex separation number at most k that contains g as a subsequence.

The first part of the lemma is more or less trivial (except for an overload of terminology and notation). The second part contains the essential insight of the algorithms in [97, 24, 19]: the numbers that are forgotten when compacting are not essential when we need to determine if we can extend the ordering to an ordering of G of vertex separation number at most k.

Compacted sequences of integers in $\{0, \ldots, k\}$ have length $O(k)$ [24], and thus for fixed ℓ, the number of compacted fingerprints of ℓ-terminal graphs is bounded by a constant.

The main idea of the algorithm of Theorem 11 is the following. Suppose we have a nice path decomposition $(X_1, \ldots, X_r)$ of width ℓ of G. For each i, we compute the set of compacted fingerprints of the linear orderings of the terminal graphs (V_i, E_i, X_i) of vertex separation number at most k. For **introduce** and for **forget** nodes, we have a subroutine that tells how such a set can be computed from the previous set. The pathwidth of G is at most k, if and only if the last of these sets (for (V_r, E_r, X_r)) is nonempty; note that $V = V_r$ and $E = E_r$. Similar as for many other dynamic programming algorithms, we can also construct (if existing) a corresponding linear ordering of width at most k, by going backwards through the tables. This linear ordering can easily be transformed to a path decomposition of width at most k (as in [89].)

Corollary 1. *For each k, the obstruction set of graphs of pathwidth at most k is computable.*

Proof. This follows directly from Theorem 10 and the discussion above. We have two ℓ-terminal subgraphs in the same equivalence class if they have the same set of fingerprints of linear orderings of vertex separation number at most k. □

Similar results hold for treewidth. There are however several additional technical difficulties: the fingerprints are much harder to describe because of the tree structure, and a procedure has to be built for the **join** nodes. Similar results have been designed for other width parameters, e.g., [139, 140].

5.6 A Win-Win Theorem and a Linear-Time Algorithm

In this section, we sketch a linear-time algorithm for the fixed-parameter case of pathwidth. The algorithm follows the main ideas of the linear-time algorithm for the fixed-parameter case of treewidth [14]; some arguments are somewhat simpler for the case of pathwidth.

We denote with $G + \{v, w\}$ the graph obtained from G by adding the edge $\{v, w\}$. The following lemma is well known in its variant for treewidth, see e.g., [14]. Its statement and proof are identical for pathwidth.

Lemma 4. *Let $G = (V, E)$ be a graph, and $k \geq 0$. Suppose v and w have at least $k+1$ common neighbors., Each path decomposition of width at most k of G is also a path decomposition of width at most k of $G + \{v, w\}$, and the pathwidth of G is at most k, if and only if the pathwidth of $G + \{v, w\}$ is at most k.*

Proof. Suppose v and w have at least $k + 1$ common neighbors.

Now, suppose that $(X_1, \ldots, X_r)$ is a path decomposition of G of width at most k. If there is a bag X_i with $v, w \in X_i$, then this is also a path decomposition of $G + \{v, w\}$ and hence the pathwidth of $G + \{v, w\}$ is at most k. Suppose such a bag does not exist. W.l.o.g., suppose the first bag that contains v is before the first bag that contains w. Let $v \in X_i$ with i maximal; and let $w \in X_j$ with j minimal. Now all common neighbors of v and w must belong to a bag containing v and to a bag containing w, and hence must belong to the first bag that contains w: this bag hence has size at least $k + 2$ as it contains w and at least $k + 1$ common neighbors of v and w, contradiction. The equivalence now follows from this, and the trivial observation that the pathwidth of G is never larger than the pathwidth of $G + \{v, w\}$. □

Lemma 4 allows us to add edges between vertices with at least $k + 1$ common neighbors, without changing the answer to the question if the graph at hand has pathwidth at most k. In order to get a linear-time algorithm, we only look at neighbors of bounded degree. In this case, we define a number b_k and use it as upper bound for the degree of neighbors to make the proof work.

Define $a_k = \frac{k}{2}(2k + 2)(2k + 1) + k + 2$, and $b_k = a_k + k + 1$.

The *k-improved graph* of a graph $G = (V, E)$ is obtained from G by adding an edge between each pair of nonadjacent vertices v, w such that there are at least $k + 1$ vertices of degree at most b_k that are a common neighbor of v and w.

Building the k-improved graph is *not* an iterative process: the new edges are added simultaneously for all pairs in one round. It is well possible that the k-improved graph of the k-improved graph of G has more edges than the k-improved graph of G, but taking the closure of the improvement operation might take too much time.

Suppose we have a graph $G = (V, E_G)$ and its k-improved graph $H = (V, E_H)$. We say that a vertex is i-simplicial, if its neighbors in G form a clique in H, i.e., for each pair of edges $\{v, w\} \in E_G$, $\{v, x\} \in E_G$, we have $w = x$ or $\{w, x\} \in E_H$.

The following theorem gives us a 'win-win' approach to computing treewidth: we first make the improved graph; then greedily compute some maximal matching M. The theorem shows that we either have 'a large maximal matching' or 'many simplicial vertices'; in both cases, we can solve the problem by first solving the problem on a graph with linearly many fewer vertices and then running the algorithm that was discussed in Section 5.5. (For other win-win theorems, see e.g., [66],[54, Chapter 8.1].)

Theorem 12. *Let $G = (V, E)$ be a graph of pathwidth at most k. Let H be the k-improved graph of G. Let M be a maximal matching in H. Let X be the set of i-simplicial vertices in G. Then $2|M| + |X| \geq \lfloor n/a_k \rfloor$.*

Proof. By Lemma 4, the pathwidth of H is at most k. Consider a nice path decomposition $(X_1, \ldots, X_r)$ of H (and hence also of G) of width at most k. This path decomposition has n **introduce** nodes or **leaf** nodes: each vertex is introduced exactly once (with one vertex introduced in X_1). A *piece* of the path decomposition is a collection of successive nodes that contains exactly a_k introduce nodes. Note that the path decomposition contains $\lfloor n/a_k \rfloor$ non-overlapping pieces. A central part of the proof is the following claim.

Claim. Let $(X_i, X_{i+1}, \ldots, X_j)$ be a piece. Let $W = \bigcup_{s=i+1}^{j-1} X_i - (X_i \cup X_j)$. W contains a vertex that is an endpoint of an edge in M or that is i-simplicial.

Proof. Let W' be the set of vertices that are 'introduced' by an **introduce** node in the piece. As we have a_k **introduce** nodes in the piece, $|W'| = a_k$. Vertices in X_i are introduced in a node with index at most i, so $W = W' - X_j$, and hence $W \geq a_k - (k+1) = \frac{k}{2}(2k+2)(2k+1) + 1$.

Consider a vertex $v \in W$. If v is i-simplicial, then the claim holds, so suppose v is not i-simplicial. Thus, v must have two neighbors in G that are not adjacent in H, say x and y. As v belongs to a bag $X_{i'}$ with $i < i' < j$, but v does not belong to X_i or X_j, the only bags v can belong to are the bags $X_{i''}$ with $i < i'' < j$, and hence all neighbors of v belong to $W \cup X_i \cup X_j$, and hence v has degree at most $a_k + k + 1 = b_k$.

First, suppose $x \in W$. Then either v is an endpoint of an edge in M, x is an endpoint of an edge in M, or M is not a maximal matching. So, the claim holds in this case. Similarly if $y \in M$. The case that remains is that both x and y belong to $X_i \cup X_j$.

I.e., we have that each vertex in W has two nonadjacent neighbors in $X_i \cup X_j$, and degree at most b_k. As there are at most $\frac{1}{2}(2k+2)(2k+1)$ pairs of vertices in $X_i \cup X_j$, there must be a pair of nonadjacent vertices in $X_i \cup X_j$ that has at least $k+1$ common neighbors in W, each with of degree at most b_k. But then the edge $\{v, w\}$ must have been added during the improvement step, i.e., $\{v, w\} \in E_H$, contradiction. □

The proof of Theorem 12 can now easily be concluded: we have $\lfloor n/a_k \rfloor$ nonoverlapping pieces. Each of these pieces contains a vertex that is i-simplicial or endpoint of an edge in the matching M. As these vertices never belong to the first or last bag of a piece, none of these vertices can belong to two or more pieces, and hence we have $\lfloor n/a_k \rfloor$ vertices that are i-simplicial or endpoint of an edge in M, which implies that $2|M| + |X| \geq \lfloor n/a_k \rfloor$. □

We now sketch the linear-time algorithm. The algorithm gets as input a graph $G = (V, E)$, and either outputs **no** (the pathwidth of G is more than k), or outputs a path decomposition of width at most k of G. It uses the algorithm of Section 5.5 as a subroutine. Correctness and running time will be argued later.

(i). If G has at most $3a_k$ vertices, then solve the pathwidth problem by any deterministic algorithm, e.g., [4].
(ii). Compute the k-improved graph $H = (V, E_H)$.
(iii). Compute a maximal matching M in H.
(iv). Compute the set X of i-simplicial vertices of degree at most k in H.
(v). If $2|M| + |X| < \lfloor n/a_k \rfloor$ then output **no**.
(vi). If $|X| \geq |M|$ then
  (a) Let H' be obtained from H by removing all vertices in X from H.
  (b) Recursively call the algorithm on H'.
  (c) If the pathwidth of H' is larger than k, then output **no**.
  (d) Otherwise, transform the path decomposition of H' of width at most k to a path decomposition of width at most $k + 1$ of H.
(vii). Else ($|X| < |M|$)
  (a) Let H'' be obtained from H by contracting all edges in M.
  (b) Recursively call the algorithm on H''.
  (c) If the pathwidth of H'' is larger than k, then output **no**.
  (d) Otherwise, transform the path decomposition of H'' of width at most k to a path decomposition of width at most $2k + 1$ of H.
(viii). (Now, we have a path decomposition of H of width at most $2k+1$.) Use the algorithm of Section 5.5 on H using the path decomposition constructed in the earlier step.

Several of the steps need more detail, and a proof that they can be performed in linear time. First, we argue correctness of the algorithm. We first consider Step 3. If the pathwidth of G is at most k, then the pathwidth of H is at most k (Lemma 4). Thus H has no clique of size $k + 1$ or more, and hence there cannot be i-simplicial vertices of degree more than k. Thus, by Theorem 12, $2|M| + |X| \geq \lfloor n/a_k \rfloor$. So, if we decide **no** in Step 3, the pathwidth of G indeed is more than k. H' is a subgraph of H, so if H' has pathwidth more than k, then H and hence G has treewidth more than k. H'' is a minor of H, and as pathwidth cannot increase when taking minors (see Section 4), if the pathwidth of H'' is more than k, then the pathwidth of H and thus G is more than k.

We now discuss a few of the steps in more detail. Computing the k-improved graph can be done in linear time with help of the use of radix sort techniques (see [41, Chapter 8.3]). Take an initially empty multiset S. For each vertex v of degree at most b_k, insert each pair of neighbors of v in S. Radix sort S, and then detect which pairs appear at least $k + 1$ times. Add these pairs to G. By radix sorting the set of edges of G, we can remove parallel edges.

Computing i-simplicial vertices again needs to use of radix sort. We leave the details as an easy exercise.

For step (vi)(d), we must find for each i-simplicial vertex $v \in X$ a bag X_{i_v} in the path decomposition of H' that contains all neighbors of v. Such a bag exists, by Lemma 1. To find the bags, one can either again exploit radix sort, or note that $v \in X_i$ with

$$i = \min_{w \in N_H(v)} \max\{j \mid w \in X_j\}$$

Add a bag with vertex set $X_{i_v} \cup \{v\}$, directly after X_{i_v}. (When more vertices are mapped to the same bag, we add a number of bags, each with one new vertex.)

Consider now step (vii)(d). For each edge $\{v, w\} \in M$, replace in each bag, each occurrence of the newly formed vertex by the contraction by v and w. In this way, bag sizes at most double, so the width is at most $2k + 1$.

We now can argue that the algorithm uses linear time. As $2|M|+|X| \geq \lfloor n/a_k \rfloor$, we have $|M| \geq \frac{1}{3}n/a_k\rfloor$ or $|X| \geq \frac{1}{3}\lfloor n/a_k \rfloor$. So, when we recursively call the algorithm on H' or H'', this graph has at most $(1 - \lfloor\frac{1}{3a_k}\rfloor)n$ vertices. So, the time of the algorithm on a graph with n vertices fulfills:

$$T(n) = T((1 - \frac{1}{3a_k})n) + O(n)$$

which implies $T(n) = O(n)$.

We now have shown the following result.

Theorem 13. *Let k be a constants. There is (and we can explicitly describe) an algorithm, that given a graph $G = (V, E)$ decides if the pathwidth of G is at most k, and if so, finds a path decomposition of G of width at most k.*

A generalization of the techniques shown above lead to a similar result for treewidth and tree decompositions [15].

6 Conclusions

In this paper, a number of results have been surveyed on algorithmic aspects of treewidth. There are still a large number of topics that have not been touched here, including most practical aspects of treewidth computations (see e.g., [25, 26]), computing treewidth on special graph classes (including the celebrated results of Bouchitté and Todinca on potential maximal cliques [34, 35]), the role of treewidth for bidimensionality theory, logspace algorithms [57], $W[1]$-hardness proofs for some problems on graphs of bounded treewidth (e.g., [18, 61]), dynamic algorithms [81], and much much more. The area of algorithmic research of treewidth is a very lively one, but can already look back to a lively history with several intriguing aspects, like the special role on nonconstructive results.

I end with mentioning a few probably very hard challenges:

– What is the complexity of TREEWIDTH, restricted to *planar graphs*. For the related BRANCHWIDTH problem, the famous ratcatcher algorithm by Seymour and Thomas [137] solves it in polynomial time; for TREEWIDTH on planar graphs, neither a polynomial time algorithm nor an NP-completeness proof is known.

– Is it possible to approximate treewidth up to a constant factor? There is an approximation with ratio $O(\sqrt{\log n})$ [59], and it is easy to show that approximation with an additive constant term is not possible assuming $P \neq NP$ [21].
– An accessible proof for Courcelle's conjecture, i.e., that shows that each problem that is finite index can be formulated in CMSOL.
– Is it possible to find an algorithm for TREEWIDTH that runs in $O(c^k n^{c'})$ for constants c and c'? Perhaps a probabilistic algorithm using ideas from [46]?

Acknowledgment. I thank Mike Fellows and numerous other colleagues for collaboration, discussions, and inspiration! It is hard to be sufficiently complete for a survey like this one, and I apologize to those whose work was not but should have been mentioned here.

References

1. Amir, E.: Approximation algorithms for treewidth. Algorithmica 56, 448–479 (2010)
2. Andrzejak, A.: An algorithm for the Tutte polynomials of graphs of bounded treewidth. Discrete Mathematics 190, 39–54 (1998)
3. Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. BIT 25, 2–23 (1985)
4. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Methods 8, 277–284 (1987)
5. Arnborg, S., Courcelle, B., Proskurowski, A., Seese, D.: An algebraic theory of graph reduction. Journal of the ACM 40, 1134–1164 (1993)
6. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms 12, 308–340 (1991)
7. Arnborg, S., Proskurowski, A.: Characterization and recognition of partial 3-trees. SIAM Journal on Algebraic and Discrete Methods 7, 305–314 (1986)
8. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23, 11–24 (1989)
9. Bern, M.W., Lawler, E.L., Wong, A.L.: Linear time computation of optimal subgraphs of decomposable graphs. Journal of Algorithms 8, 216–235 (1987)
10. Bienstock, D., Langston, M.A.: Algorithmic implications of the graph minor theorem. In: Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser, G.L. (eds.) Handbook of Operations Research and Management Science: Network Models, pp. 481–502. North-Holland, Amsterdam (1995)
11. Bienstock, D., Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a forest. Journal of Combinatorial Theory, Series B 52, 274–283 (1991)
12. Bodlaender, H.L.: Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. Journal of Algorithms 11, 631–643 (1990)
13. Bodlaender, H.L.: Improved self-reduction algorithms for graphs with bounded treewidth. Discrete Applied Mathematics 54, 101–115 (1994)
14. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)

15. Bodlaender, H.L.: Treewidth: Algorithmic Techniques and Results. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 19–36. Springer, Heidelberg (1997)
16. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
17. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences 75, 423–434 (2009)
18. Bodlaender, H.L., Fellows, M.R., Hallett, M.T., Wareham, H.T., Warnow, T.J.: The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs. Theoretical Computer Science 244, 167–188 (2000)
19. Bodlaender, H.L., Fellows, M.R., Thilikos, D.M.: Derivation of algorithms for cutwidth and related graph layout parameters. Journal of Computer and System Sciences 75, 231–244 (2009)
20. Bodlaender, H.L., Fomin, F.V., Lokshtanov, D., Penninkx, E., Saurabh, S., Thilikos, D.M.: (Meta) kernelization. In: Proceedings of the 50th Annual Symposium on Foundations of Computer Science, FOCS 2009, pp. 629–638. IEEE Computer Society (2009)
21. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and minimum elimination tree height. Journal of Algorithms 18, 238–255 (1995)
22. Bodlaender, H.L., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. SIAM J. Comput. 27, 1725–1746 (1998)
23. Bodlaender, H.L., Jansen, B.M.P., Kratsch, S.: Preprocessing for Treewidth: A Combinatorial Analysis through Kernelization. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 437–448. Springer, Heidelberg (2011)
24. Bodlaender, H.L., Kloks, T.: Efficient and constructive algorithms for the pathwidth and treewidth of graphs. Journal of Algorithms 21, 358–402 (1996)
25. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations I. Upper bounds. Information and Computation 208, 259–275 (2010)
26. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations II. Lower bounds. Information and Computation 209, 1103–1119 (2011)
27. Bodlaender, H.L., Koster, A.M.C.A., Van den Eijkhof, F.: Pre-processing rules for triangulation of probabilistic networks. Computational Intelligence 21(3), 286–305 (2005)
28. Bodlaender, H.L., Möhring, R.H.: The pathwidth and treewidth of cographs. SIAM Journal on Discrete Mathematics 6, 181–188 (1993)
29. Bodlaender, H.L., van Antwerpen-de Fluiter, B.: Reduction algorithms for graphs of small treewidth. Information and Computation 167, 86–119 (2001)
30. Bodlaender, H.L., van Leeuwen, E.J., van Rooij, J.M.M., Vatshelle, M.: Faster Algorithms on Branch and Clique Decompositions. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 174–185. Springer, Heidelberg (2010)
31. Borie, R.B.: Generation of polynomial-time algorithms for some optimization problems on tree-decomposable graphs. Algorithmica 14, 123–137 (1995)
32. Borie, R.B., Parker, R.G., Tovey, C.A.: Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. Algorithmica 7, 555–581 (1992)
33. Borie, R.B., Parker, R.G., Tovey, C.A.: Solving problems on recursively constructed graphs. ACM Computing Surveys 41(4) (2008)

34. Bouchitté, V., Todinca, I.: Treewidth and minimum fill-in: Grouping the minimal separators. SIAM Journal on Computing 31, 212–232 (2001)
35. Bouchitté, V., Todinca, I.: Listing all potential maximal cliques of a graph. Theoretical Computer Science 276, 17–32 (2002)
36. Brown, D.J., Fellows, M.R., Langston, M.A.: Polynomial-time self-reducibility: Theoretical motivations and practical results. International Journal of Computer Mathematics 31, 1–9 (1989)
37. Cattell, K., Dinneen, M.J., Downey, R.G., Fellows, M.R., Langston, M.A.: On computing graph minor obstruction sets. Theoretical Computer Science 233, 107–127 (2000)
38. Chaudhuri, S., Zaroliagis, C.D.: Shortest paths in digraphs of small treewidth. Part II: Optimal parallel algorithms. Theoretical Computer Science 203, 205–223 (1998)
39. Chaudhuri, S., Zaroliagis, C.D.: Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. Algorithmica 27, 212–226 (2000)
40. Chen, H.: Quantified constraint satisfaction and bounded treewidth. In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 17th European Conference on Artificial Intelligence, ECAI 2004, pp. 161–165 (2004)
41. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
42. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation 85, 12–75 (1990)
43. Courcelle, B.: The monadic second-order logic of graphs V: On closing the gap between definability and recognizability. Theoretical Computer Science 80, 153–202 (1991)
44. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique width. Theoretical Computer Science 33, 125–150 (2000)
45. Courcelle, B., Mosbah, M.: Monadic second-order evaluations on tree-decomposable graphs. Theoretical Computer Science 109, 49–82 (1993)
46. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. In: Proceedings of the 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, pp. 150–159 (2011)
47. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
48. Dantsin, E., Wolpert, A.: On Moderately Exponential Time for SAT. In: Strichman, O., Szeider, S. (eds.) SAT 2010. LNCS, vol. 6175, pp. 313–325. Springer, Heidelberg (2010)
49. de Fluiter, B.: Algorithms for Graphs of Small Treewidth. PhD thesis, Utrecht University (1997)
50. Díaz, J., Serna, M., Thilikos, D.M.: Counting H-colorings of partial k-trees. Theoretical Computer Science 281, 291–309 (2002)
51. Diestel, R., Jensen, T.R., Gorbunov, K.Y., Thomassen, C.: Highly connected sets and the excluded grid theorem. Journal of Combinatorial Theory, Series B 75, 61–73 (1999)
52. Dorn, F.: Dynamic programming and planarity: Improved tree-decomposition based algorithms. Discrete Applied Mathematics 158, 800–808 (2010)

53. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. Algorithmica 58, 790–810 (2010)
54. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
55. Drucker, A.: New limits to classical and quantum instance compression (preliminary draft) (2012) (manuscript)
56. van den Eijkhof, F., Bodlaender, H.L., Koster, A.M.C.A.: Safe reduction rules for weighted treewidth. Algorithmica 47, 138–158 (2007)
57. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. In: Proceedings of the 51st Annual Symposium on Foundations of Computer Science, FOCS 2010, pp. 143–152 (2010)
58. Ellis, J.A., Sudborough, I.H., Turner, J.: Graph separation and search number. Report DCS-66-IR, University of Victoria (1987)
59. Feige, U., Hajiaghayi, M., Lee, J.R.: Improved approximation algorithms for minimum weight vertex separators. SIAM Journal on Computing 38, 629–657 (2008)
60. Fellows, M.R.: The Robertson-Seymour theorems: A survey of applications. Contemporary Mathematics 89, 1–18 (1989)
61. Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. Information and Control 209, 143–153 (2011)
62. Fellows, M.R., Langston, M.A.: Nonconstructive advances in polynomial-time complexity. Information Processing Letters 26, 157–162 (1987)
63. Fellows, M.R., Langston, M.A.: Fast Self-reduction Algorithms for Combinatorial Problems of VLSI Design. In: Reif, J.H. (ed.) AWOC 1988. LNCS, vol. 319, pp. 278–287. Springer, Heidelberg (1988)
64. Fellows, M.R., Langston, M.A.: Nonconstructive tools for proving polynomial-time decidability. Journal of the ACM 35, 727–739 (1988)
65. Fellows, M.R., Langston, M.A.: An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, FOCS 1989, pp. 520–525 (1989)
66. Fellows, M.R., Langston, M.A.: On search, decision and the efficiency of polynomial-time algorithms. In: Proceedings of the 21st Annual Symposium on Theory of Computing, STOC 1989, pp. 501–512 (1989)
67. Fellows, M.R., Langston, M.A.: Fast search algorithms for layout permutation problems. International Journal on Computer Aided VLSI Design 3, 325–340 (1991)
68. Fellows, M.R., Langston, M.A.: On well-partial-order theory and its application to combinatorial problems of VLSI design. SIAM Journal on Discrete Mathematics 5, 117–126 (1992)
69. Fellows, M.R., Langston, M.A.: On search, decision and the efficiency of polynomial-time algorithms. Journal of Computer and System Sciences 49, 769–779 (1994)
70. Fernández-Baca, D., Slutzki, G.: Parametic problems on graphs of bounded treewidth. Journal of Algorithms 16, 408–430 (1994)
71. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica 54, 181–207 (2009)
72. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Bidimensionality and kernels. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, pp. 503–510 (2010)

73. Fomin, F.V., Lokshtanov, D., Saurabh, S., Thilikos, D.M.: Linear kernels for (connected) dominating set on H-minor-free graphs. In: Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, pp. 82–93 (2012)
74. Fomin, F.V., Thilikos, D.M.: New upper bounds on the decomposability of planar graphs. Journal of Graph Theory 51, 53–81 (2006)
75. Friedman, H., Robertson, N., Seymour, P.D.: The metamathematics of the graph minor theorem. Contemporary Mathematics 65, 229–261 (1987)
76. Ganian, R., Hliněný, P.: On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. Discrete Applied Mathematics 158, 851–867 (2010)
77. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York (1980)
78. Gottlob, G., Leone, N., Scarcello, F.: A comparison of structural CSP decomposition methods. Acta Informatica 124, 243–282 (2000)
79. Gupta, A., Nishimura, N.: The complexity of subgraph isomorphism for classes of partial k-trees. Theoretical Computer Science 164, 287–298 (1996)
80. Gustedt, J., Mæhle, O.A., Telle, J.A.: The Treewidth of Java Programs. In: Mount, D.M., Stein, C. (eds.) ALENEX 2002. LNCS, vol. 2409, pp. 86–97. Springer, Heidelberg (2002)
81. Hagerup, T.: Dynamic algorithms for graphs of bounded treewidth. Algorithmica 27, 292–315 (2000)
82. Hagerup, T., Katajainen, J., Nishimura, N., Ragde, P.: Characterizing multiterminal flow networks and computing flows in networks of small treewidth. Journal of Computer and System Sciences 57, 366–375 (1998)
83. Hein, A., Koster, A.M.C.A.: An Experimental Evaluation of Treewidth at Most Four Reductions. In: Pardalos, P.M., Rebennack, S. (eds.) SEA 2011. LNCS, vol. 6630, pp. 218–229. Springer, Heidelberg (2011)
84. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. Journal of Computer and System Sciences 62, 367–375 (2001)
85. Isobe, S., Zhou, X., Nishizeki, T.: A polynomial-time algorithm for finding total colorings of partial k-trees. International Journal of Foundations of Computer Science 10, 171–194 (1999)
86. Kabanets, V.: Recognizability Equals Definability for Partial k-Paths. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 805–815. Springer, Heidelberg (1997)
87. Kaller, D.: Definability equals recognizability of partial 3-trees and k-connected partial k-trees. Algorithmica 27, 348–381 (2000)
88. Kashem, M.A., Zhou, X., Nishizeki, T.: Algorithms for generalized vertex-rankings of partial k-trees. Theoretical Computer Science 240, 407–427 (2000)
89. Kinnersley, N.G.: The vertex separation number of a graph equals its path width. Information Processing Letters 42, 345–350 (1992)
90. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley, Boston (2005)
91. Kloks, T.: Treewidth. Computations and Approximations. LNCS, vol. 842. Springer, Heidelberg (1994)
92. Kneis, J., Langer, A., Rossmanith, P.: Courcelle's theorem - a game-theoretic approach. Discrete Optimization 8(4), 568–594 (2011)
93. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: A bound on the pathwidth of sparse graphs with applications to exact algorithms 23, 407–427 (2009)
94. Koster, A.M.C.A., van Hoesel, S.P.M., Kolen, A.W.J.: Solving partial constraint satisfaction problems with tree decomposition. Networks 40(3), 170–180 (2002)

95. Koutsonas, A., Thilikos, D.M.: Planar feedback vertex set and face cover: combinatorial bounds and subexponential algorithms. Algorithmica 60, 987–1003 (2011)
96. Lagergren, J.: Efficient parallel algorithms for graphs of bounded tree-width. Journal of Algorithms 20, 20–44 (1996)
97. Lagergren, J., Arnborg, S.: Finding Minimal Forbidden Minors Using a Finite Congruence. In: Albert, J.L., Monien, B., Rodríguez-Artalejo, M. (eds.) ICALP 1991. LNCS, vol. 510, pp. 532–543. Springer, Heidelberg (1991)
98. Lapoire, D.: Recognizability Equals Definability, for Every Set of Graphs of Bounded Tree-width. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 618–628. Springer, Heidelberg (1998)
99. Lauritzen, S.J., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. The Journal of the Royal Statistical Society, Series B (Methodological) 50, 157–224 (1988)
100. Leaver-Fay, A., Liu, Y., Snoeyink, J.: Faster placement of hydrogen atoms in protein structures by dynamic programming. In: Proceedings of the 6th Workshop on Algorithm Engineering and Experimentation and the 1st Workshop on Analytic Algorithmics and Combinatorics, ALENEX/ANALCO 2004, pp. 39–48 (2004)
101. Lengauer, T.: Black-white pebbles and graph separation. Acta Informatica 16, 465–475 (1981)
102. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36, 177–189 (1979)
103. Lipton, R.J., Tarjan, R.E.: Applications of a planar separator theorem. SIAM Journal on Computing 9, 615–627 (1980)
104. Lokshtanov, D., Marx, D., Saurabh, S.: Known algorithms on graphs on bounded treewidth are probably optimal. In: Randall, D. (ed.) Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 777–789 (2011)
105. Makowsky, J.A.: Coloured Tutte polynomials and Kauffman brackets for graphs of bounded tree width. Discrete Applied Mathematics 145, 276–290 (2004)
106. Mata-Montero, E.: Resilience of partial k-tree networks with edge and node failures. Networks 21, 321–344 (1991)
107. Matoušek, J., Thomas, R.: Algorithms for finding tree-decompositions of graphs. Journal of Algorithms 12, 1–22 (1991)
108. McDiarmid, C., Reed, B.: Channel assignment on graphs of bounded treewidth. Discrete Mathematics 273, 183–192 (2003)
109. Reed, B.: Finding approximate separators and computing tree-width quickly. In: Proceedings of the 24th Annual Symposium on Theory of Computing, STOC 1992, pp. 221–228. ACM Press, New York (1992)
110. Robertson, N., Seymour, P.D.: Graph minors. I. Excluding a forest. Journal of Combinatorial Theory, Series B 35, 39–61 (1983)
111. Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. Journal of Combinatorial Theory, Series B 36, 49–64 (1984)
112. Robertson, N., Seymour, P.D.: Graph minors — a survey. In: Anderson, I. (ed.) Surveys in Combinatorics, pp. 153–171. Cambridge Univ. Press (1985)
113. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
114. Robertson, N., Seymour, P.D.: Graph minors. V. Excluding a planar graph. Journal of Combinatorial Theory, Series B 41, 92–114 (1986)
115. Robertson, N., Seymour, P.D.: Graph minors. VI. Disjoint paths across a disc. Journal of Combinatorial Theory, Series B 41, 115–138 (1986)
116. Robertson, N., Seymour, P.D.: Graph minors. VII. Disjoint paths on a surface. Journal of Combinatorial Theory, Series B 45, 212–254 (1988)

117. Robertson, N., Seymour, P.D.: Graph minors. IV. Tree-width and well-quasi-ordering. Journal of Combinatorial Theory, Series B 48, 227–254 (1990)
118. Robertson, N., Seymour, P.D.: Graph minors. IX. Disjoint crossed paths. Journal of Combinatorial Theory, Series B 49, 40–77 (1990)
119. Robertson, N., Seymour, P.D.: Graph minors. VIII. A Kuratowski theorem for general surfaces. Journal of Combinatorial Theory, Series B 48, 255–288 (1990)
120. Robertson, N., Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52, 153–190 (1991)
121. Robertson, N., Seymour, P.D.: Graph minors. XXII. Irrelevant vertices in linkage problems (1992) (manuscript)
122. Robertson, N., Seymour, P.D.: Graph minors. XI. Distance on a surface. Journal of Combinatorial Theory, Series B 60, 72–106 (1994)
123. Robertson, N., Seymour, P.D.: Graph minors. XII. Excluding a non-planar graph. Journal of Combinatorial Theory, Series B 64, 240–272 (1995)
124. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. Journal of Combinatorial Theory, Series B 63, 65–110 (1995)
125. Robertson, N., Seymour, P.D.: Graph minors. XIV. Extending an embedding. Journal of Combinatorial Theory, Series B 65, 23–50 (1995)
126. Robertson, N., Seymour, P.D.: Graph minors. XV. Giant steps. Journal of Combinatorial Theory, Series B 68, 112–148 (1996)
127. Robertson, N., Seymour, P.D.: Graph minors. XVII. Taming a vortex. Journal of Combinatorial Theory, Series B 77, 162–210 (1999)
128. Robertson, N., Seymour, P.D.: Graph minors. XVI. Excluding a non-planar graph. Journal of Combinatorial Theory, Series B 89, 43–76 (2003)
129. Robertson, N., Seymour, P.D.: Graph minors. XVIII. Tree-decompositions and well-quasi ordering. Journal of Combinatorial Theory, Series B 89, 77–108 (2003)
130. Robertson, N., Seymour, P.D.: Graph minors. XIX. Well-quasi-ordering on a surface. Journal of Combinatorial Theory, Series B 90, 325–385 (2004)
131. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. Journal of Combinatorial Theory, Series B 92, 325–357 (2004)
132. Robertson, N., Seymour, P.D.: Graph minors. XXI. Graphs with unique linkages. Journal of Combinatorial Theory, Series B 99, 583–616 (2009)
133. Robertson, N., Seymour, P.D.: Graph minors XXIII. Nash-Williams' immersion conjecture. Journal of Combinatorial Theory, Series B 100, 181–205 (2010)
134. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. Journal of Combinatorial Theory, Series B 62, 323–348 (1994)
135. Sanders, D.P.: On linear recognition of tree-width at most four. SIAM Journal on Discrete Mathematics 9(1), 101–117 (1996)
136. Sau, I., Thilikos, D.M.: Subexponential parameterized algorithms for degree-constrained subgraph problems on planar graphs. Journal of Discrete Algorithms 8, 330–338 (2010)
137. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica 14(2), 217–241 (1994)
138. Telle, J.A., Proskurowski, A.: Algorithms for vertex partitioning problems on partial k-trees. SIAM Journal on Discrete Mathematics 10, 529–550 (1997)
139. Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Cutwidth I: A linear time fixed parameter algorithm. Journal of Algorithms 56, 1–24 (2005)
140. Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Cutwidth II: Algorithms for partial w-trees of bounded degree. Journal of Algorithms 56, 25–49 (2005)
141. van Rooij, J.M.M.: Exact exponential-time algorithms for domination problems in graphs. PhD thesis, Department of Computer Science, Utrecht University (2011)

142. van Rooij, J.M.M., Bodlaender, H.L., Rossmanith, P.: Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 566–577. Springer, Heidelberg (2009)
143. Wimer, T.V.: Linear Algorithms on k-Terminal Graphs. PhD thesis, Dept. of Computer Science, Clemson University (1987)
144. Wimer, T.V., Hedetniemi, S.T., Laskar, R.: A methodology for constructing linear graph algorithms. Congressus Numerantium 50, 43–60 (1985)
145. Zhou, X., Fuse, K., Nishizeki, T.: A linear algorithm for finding $[g, f]$-colorings of partial k-trees. Algorithmica 27, 227–243 (2000)