Advance Access publication on July 19, 2007 doi:10.1093/comjnl/bxm037

Combinatorial Optimization on Graphs of Bounded Treewidth

HANS L. BODLAENDER[1,*] AND ARIE M. C. A. KOSTER[2]

[1]Institute of Information and Computing Sciences, Utrecht University, PO Box 80.089, 3508 TB Utrecht, the Netherlands
[2]Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, Coventry CV4 7AL, UK
*Corresponding author: hansb@cs.uu.nl

There are many graph problems that can be solved in linear or polynomial time with a dynamic programming algorithm when the input graph has bounded treewidth. For combinatorial optimization problems, this is a useful approach for obtaining fixed-parameter tractable algorithms. Starting from trees and series-parallel graphs, we introduce the concepts of treewidth and tree decompositions, and illustrate the technique with the Weighted Independent Set problem as an example. The paper surveys some of the latest developments, putting an emphasis on applicability, on algorithms that exploit tree decompositions, and on algorithms that determine or approximate treewidth and find tree decompositions with optimal or close to optimal treewidth. Directions for further research and suggestions for further reading are also given.

Keywords: bounded treewidth, paramaterized algorithms

Received 17 March 2006; revised 11 December 2006

In this paper, we provide a tutorial and survey on the notion of treewidth in the context of combinatorial optimisation problems defined on graphs. We start in Section 1 with a discussion of combinatorial optimisation problems and their complexity on *trees* in comparison to general graphs. Next, we discuss how the results for trees can be generalized to other and larger classes of graphs. In particular, we discuss algorithms for *series-parallel graphs* in Section 2.1, and then introduce the notions of treewidth and tree decompositions in Section 2.2. In Section 2.3, we discuss the general shape of many algorithms that use tree decompositions, and define a special form of tree decompositions, called *nice tree decompositions*, that are useful for the design and explanation of algorithms. In Section 3, we discuss how treewidth and (nice) tree decompositions can be used to design algorithms that solve problems on graphs given with a tree decomposition with small treewidth. First, in Section 3.1, such an algorithm for Weighted Independent Set is explained and analysed, as an elaborate example. Then, in Section 3.2, we discuss other problems and problem classes that the technique can be applied to. Section 3.3 gives examples where the methods can be used to show fixed-parameter tractability for several parameterized graph problems, where the parameter is something other than treewidth. Section 4 treats the topic of determining the treewidth of a given graph, and of finding tree decompositions of optimal (or close to optimal) treewidth. We look at exact algorithms (Section 4.1), approximation algorithms, upper bound heuristics (Section 4.2) and lower bound heuristics (Section 4.3). Section 5 shows how the discussed algorithms can be used to obtain algorithmic results for planar graphs. Section 5.1 deals with the relation between treewidth and the famous planar separator theorem, and discusses how this can be used to obtain faster exact algorithms for problems on planar graphs. Section 5.2 describes how treewidth can be used to obtain approximation algorithms for several problems on planar graphs. Some extensions to other classes of graphs are discussed in Section 5.3. Some final remarks and conclusions can be found in Section 6.

Throughout this paper, we use standard graph theoretic notions (see e.g. Harary [1]). For a graph $G = (V, E)$. and a set of vertices $W \subseteq V$, the subgraph of G, *induced* by W, is the graph $G[W] = (W, \{\{v, w\} \in E | v, w \in W\})$.

1. MANY GRAPH PROBLEMS ARE EASY ON TREES

Combinatorial optimization deals with decision-making problems of a discrete nature. Out of a finite or countably infinite number of alternatives one has to choose the best one according to some quantifiable criterion. In most situations, the alternatives are composed of a number of individual decisions that interact with each other: not every combination of decisions leads to a feasible alternative and the measurement of quality depends heavily on the composition of the individual choices. In numerous situations, these interactions between the decisions can modelled by an *undirected graph* $G = (V, E)$ consisting of a set of *vertices* V and a set of unordered pairs of distinct vertices, the *edges* E. Depending on the application, the individual decisions correspond to the vertices or edges, whereas the interactions are modelled by the edges or vertices, respectively. In the vertex colouring problem, we have to choose a colour for each vertex (the decisions) such that no two vertices connected by an edge (the interactions) are coloured with the same colour. Alternatively, in the edge colouring problem, we have to choose a colour for every edge (the decisions) such that all edges incident to the same vertex (the interactions) are coloured differently. (The quantifiable criterion to optimize in both cases is the number of different colours used overall: the less the better.)

Like other decision problems, combinatorial optimisation problems are classified according to their complexity. Unless $\mathcal{P} = \mathcal{NP}$ we have problems that are easy (i.e. can be solved in polynomial time; the members of $\mathcal{P}$) and those that are hard (i.e. no polynomial time algorithm exists). Many of the well-known combinatorial optimisation problems defined on graphs (like the ones mentioned above) belong to the class of $\mathcal{NP}$-hard problems in general. However, if we know more about the structure of the graph, the problem typically turns out to be more tractable. In the best cases, the problem becomes polynomial time solvable. This in particularly holds for *trees*, connected graphs without cycles.

Let us consider the Weighted Independent Set problem. Given is a graph $G = (V, E)$ with vertex weights $c(v) \in \mathbf{Z}^+$ for each vertex $v \in V$. We are looking for a subset S of the vertices such that they are pairwise non-adjacent so that the sum of the weights $c(S) = \sum_{v \in S} c(v)$ is minimized. This problem is known to be $\mathcal{NP}$-hard for general graphs. For trees, however it turns out to be linear time solvable. Root the tree at an arbitrary vertex r and let $T(v)$ denote the subtree with v as root. We denote with $A(v)$ the maximum weight of an independent set in $T(v)$ and with $B(v)$ the maximum weight of an independent set in $T(v)$ not containing v. Thus, $A(r)$ provides the optimum value.

Now, we can compute $A(v)$ and $B(v)$ in a bottom-to-top procedure, starting at the leafs of $T(r)$. If v is a leaf, $A(v) := c(v)$ and $B(v) := 0$. If v is a non-leaf vertex and v has children $x_1, \ldots, x_r$.

$$A(v) := \max\{c(v) + B(x_1) + \cdots + B(x_r), A(x_1) + \cdots + A(x_r)\}$$

as either v is included in the maximum weighted independent set (and thus its children are not) or not (and thus all children can be). The latter value is exactly $B(v) := A(x_1) + \cdots + A(x_r)$. As every value $A(v)$ and $B(v)$ is computed once and used at most once, the algorithm runs in $\mathcal{O}(n)$ time, where n is the number of vertices. The construction of the corresponding independent sets can also be done in linear time.

Other problems that are $\mathcal{NP}$-hard in general but polynomial time solvable on trees are, for example, Dominating Set (Hedetniemi *et al.* [2]), Chromatic Index (edge colouring) (Mitchell and Hedetniemi [3]) and Optimal Linear Arrangement (Shiloach [4]).

2. BEYOND TREES: FROM SERIES–PARALLEL GRAPHS TO TREEWIDTH

2.1. Series–parallel graphs

In the previous section, we have seen the existence of linear time algorithms for several combinatorial problems, restricted to trees. Trees are a very limited structure compared to general graphs. A natural question therefore is whether such algorithms also exist for structures generalizing trees. One such structure is the *series-parallel graph*. A two-terminal labelled graph (G, s, t) consists of a graph G with a marked source $s \in V$ and sink $t \in V$. New graphs can be composed from two two-terminal labelled graphs in two ways: in series or in parallel. The series composition of two-terminal labelled graphs (G, s, t) and (H, x, y) consists of the identification of t with x and the labelling of s and y. The parallel composition of (G, s, t) and (H, x, y) consists of the identification of both s with x and t with y where s and t remain labelled (see Fig. 1). A graph is a series–parallel graph if it can be created from single two-terminal labelled edges by series and/or parallel compositions.

For every series–parallel graph, we can construct a so-called *SP-tree*, a binary tree representing the series and parallel composition of the graph from single two-terminal labelled edges. The leafs of the SP-tree $T(G)$ correspond to the edges $e \in E$, whereas the internal nodes are either labelled S or P for series and parallel composition of the series–parallel graphs associated by the child-subtrees (see Fig. 2 for a series–parallel graph and its SP-tree).

With $G(i)$ we denote the series-parallel graph that is associated with node i of the SP-tree. Let s and t denote the terminals

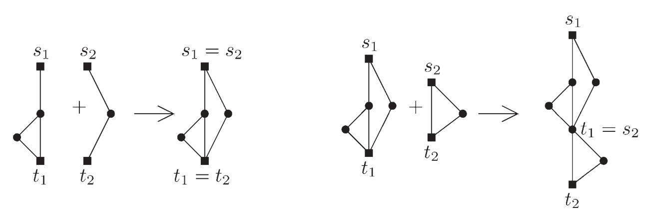

FIGURE 1. Parallel and series composition of two-terminal labelled graphs.

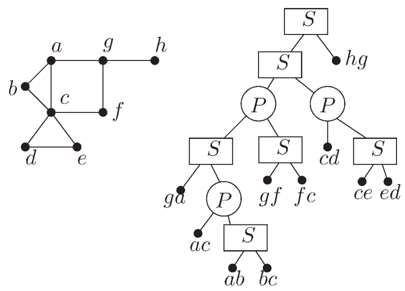

FIGURE 2. A series–parallel graph and its SP- tree.

of $G(i)$. To compute the maximum weighted independent set for series–parallel graphs in polynomial time, we introduce four values for every node i.

$AA(i)$: maximum weight of independent set containing both s and t,
$AB(i)$: maximum weight of independent set containing both s but not t,
$BA(i)$: maximum weight of independent set containing both t but not s, and
$BB(i)$: maximum weight of independent set containing neither s nor t,

For leafs of the SP-tree, the computation of the four values is trivial ($AA(i) := -\infty$, $AB(i) := c(s)$, $BA(i) := c(t)$ and $BB(i) := 0$). For an internal node i, let i_1 and i_2 be the children of i. Depending on the composition, a case analysis using the values for series–parallel graphs $G(i_1)$ and $G(i_2)$ is performed. For an S node i (with s' the terminal between i_1 and i_2),

$$AA(i) := \max\{AA(i_1) + AA(i_2) - c(s'), AB(i_1) + BA(i_2)\},$$
$$AB(i) := \max\{AA(i_1) + AB(i_2) - c(s'), AB(i_1) + BB(i_2)\},$$
$$BA(i) := \max\{BA(i_1) + AA(i_2) - c(s'), BB(i_1) + BA(i_2)\} \text{ and}$$
$$BB(i) := \max\{BA(i_1) + AB(i_2) - c(s'), BB(i_1) + BB(i_2)\}.$$

For a P node i the values are computed as follows:

$$AA(i) := AA(i_1) + AA(i_2) - c(s) - c(t),$$
$$AB(i) := AB(i_1) + AB(i_2) - c(s),$$
$$BA(i) := BA(i_1) + BA(i_2) - c(t) \text{ and}$$
$$BB(i) := BB(i_1) + BB(i_2).$$

As the computation can be done in $\mathcal{O}$ (1) time per node, the total running time is $\mathcal{O}(m)$, where m is the number of edges of G.

Dynamic programming algorithms for other combinatorial problems on series–parallel graphs can be found in, e.g. Bern *et al.* [5], Borie *et al.* [6], Kikuno *et al.* [7] and Takamizawa *et al.* [8].

2.2. Graphs of bounded treewidth

To see that series–parallel graphs generalize trees *and* to generalize further, we introduce the notion of *treewidth* of a graph. In a long series of papers, Robertson and Seymour [9] gave a deep proof of Wagner's conjecture. In this series, they introduced the notions of pathwidth (Robertson and Seymour [10]), treewidth (Robertson and Seymour [11]) and branchwidth (Robertson and Seymour [12]). The notion of treewidth appears to be equivalent to other notions, independently proposed, e.g. a graph has treewidth at most k, if and only if it is a *partial k-tree*. See, e.g., Bodlaender [13] for an overview of several notion equivalent to treewidth or pathwidth.

DEFINITION 1. *A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ where each node $i \in I$ has associated to it a subset of vertices $X_i \subseteq V$, called the bag of i, such that*

(1) *Each vertex belongs to at least one bag: $\cup_{i \in I} X_i = V$.*
(2) *For all edges, there is a bag containing both its endpoints, i.e. for all $\{v, w\} \in E$: there is an $i \in I$ with $v, w \in X_i$.*
(3) *For all vertices $v \in V$, the set of nodes $\{i \in I \mid v \in X_i\}$ induces a subtree of T.*

The width of a 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 is the minimum width over all tree decompositions of G.

The -1 in the definition of the width of a tree decomposition is introduced for convenience only: in this way trees have treewidth one. As soon as a graph contains a cycle, condition (3) forces at least three vertices to belong to a subset X_i and hence the treewidth is at least two. In fact, series–parallel graphs have treewidth at most two. To be precise, a graph has treewidth at most two (Fig. 3) if and only if every biconnected component is series–parallel (Bodlaender and van Antwerpen-de Fluiter [14]).

Related notions are pathwidth and path decomposition. A *path decomposition* is a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ where T is a path; the pathwidth of a graph G is the minimum width over all its path decompositions.

The notion of branchwidth is closely related to treewidth, and we refer to the overview paper by Hliněný *et al.* [15] for its definition and discussion (see also Hicks *et al.* [16]). Here, we restrict ourselves to the quantification of this relation.

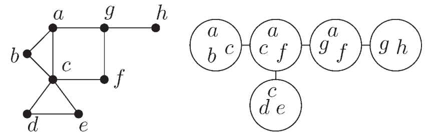

FIGURE 3. An example of a graph with a tree decomposition of width two.

Lemma 2. (Robertson and Seymour [12]). *Let $G = (V, E)$ be a graph with treewidth $\tau \geq 1$ and branchwidth β. Then*

$$\max\{\beta, 2\} \leq \tau + 1 \leq \max\{\lfloor 3/2 \cdot \beta \rfloor, 2\}$$

It is easy to verify that the treewidth of a graph equals the maximum treewidth of its connected components. Therefore, we consider only connected graphs in the rest of this paper.

A small variation on the algorithm of Section 2.1 can solve the Weighted Independent Set problem on graphs of treewidth two. Now, this result, and the result of Section 1 can be restated as follows: the Weighted Independent Set problem is solvable for graphs of treewidth one or two in polynomial time. Hence, a natural question is whether this result also holds for treewidths 3, 4, ... In Section 3.1, we present an algorithm that solves the Weighted Independent Set problem on graphs, given a tree decomposition of width at most k. For fixed k, this algorithm has polynomial running time. The increasing interest in the notion of treewidth in recent years is due to such results. A large number of graph theoretic problems become linear or polynomial time solvable when restricted to graphs of bounded treewidth, i.e. the treewidth is bounded by a constant k not part of the input. Posed in the terminology of parameterized complexity: if we have a graph problem, we can look at its parameterized variant where we take the treewidth of the input graph as parameter. For many graph problems, this parameterized variant is fixed-parameter tractable (FPT) (for more information on fixed-parameter tractability and complexity, see Downey and Fellows [17], Flum and Grohe [18], and Niedermeier [19]).

2.3. The structure of algorithms on graphs of small treewidth

In general, an FPT algorithm for a problem that takes treewidth as the parameter has the following structure.

(1) First, a tree decomposition (X, T) of G is found; the width of this tree decomposition is small (bounded by a constant, not necessarily optimal). Possibly, the tree decomposition is converted to one with a 'nice' structure.
(2) A *dynamic programming algorithm* is executed on the tree decomposition; for each node of the tree decomposition, a table is computed. For a decision problem, the table for the root of the tree T shows the answer. (For construction variants of problems, additional bookkeeping and a construction step follow the run of the dynamic programming.)

There are a few algorithms that deviate from the scheme. In particular, some algorithms exploit *reduction* techniques (see Arnborg *et al.* [20], Bodlaender and van Antwerpen-de Fluiter [21]).

For the algorithms that follow the scheme above, there are thus two important issues to discuss: how can we solve the problem using dynamic programming, assuming a small width tree decomposition is available, and how can we find 'good' tree decompositions? The first of these topics is discussed in Section 3; the second is discussed in Section 4. However, we first discuss a special type of tree decomposition that is very useful for describing dynamic programming algorithms, called a *nice tree decomposition*.

In a nice tree decomposition, one node in T is considered to be the root of T, and each node $i \in I$ is of one of the four following types.

- Leaf: node i is a leaf of T, and $|X_i| = 1$
- Join: node i has exactly two children, say j_1 and j_2 and $X_i = X_{j_1} = X_{j_2}$.
- Introduce: node i has exactly one child, say j, and there is a vertex $v \in V$ with $X_i = X_j \cup \{v\}$.
- Forget: node i has exactly one child, say j, and there is a vertex $v \in V$ with $X_j = X_i \cup \{v\}$

It is not hard to see that if G has treewidth at most k, then G also has a nice tree decomposition of width at most k, which has $O(n)$ tree nodes. Such a nice tree decomposition can be built in linear time, given a (not nice) tree decomposition (Kloks [22]). A nice tree decomposition can also be viewed as a *parse tree* of a graph.

3. DYNAMIC PROGRAMMING ON TREE DECOMPOSITIONS

The algorithmic potential of treewidth for practical use is highest when the graphs at hand have small treewidth. As mentioned before, for many problems, there are linear time algorithms solving the problems on graphs with bounded treewidth. These algorithms are usually exponential in the treewidth. For instance, as we will see in Section 3.1, there is an algorithm that solves the Weighted Independent Set problem in $O(2^k \cdot n)$ time, given a graph G together with a tree decomposition of width k. Fortunately, there are many cases where graphs have small treewidth. For several known classes of graphs, there is a constant upper bound on the treewidth, e.g. outerplanar graphs have treewidth two. For an overview, see Bodlaender [13]. Also, in many applications, it appears that the input graphs often have small treewidth. For instance, Thorup [23] has shown that for several programming languages, the control-flow graph of a program without goto commands has treewidth bounded by a small constant. Later, this result was extended to goto-free programs in Java by Gustedt *et al.* [24], and to programs in Ada without goto's and labelled loops by Burgstaller *et al.* [25]. See also Section 4.

3.1. An algorithm for Weighted Independent Set

In this section, we give an example of a dynamic programming algorithm that exploits a tree decomposition. We use again the Weighted Independent Set problem.

Suppose we are given a graph $G = (V, E)$ with vertex weights $c(v) \in \mathbf{Z}^+$ and a tree decomposition of G of width at most k. For a simpler explanation of the algorithm, we assume the tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ is nice; otherwise it can be made nice without increasing its width (see Section 2.3).

We associate to each node $i \in I$ a graph, which we call $G_i = (V_i. E_i)$. V_i is the union of all bags X_j, with $j = i$ or j a descendant of i in T, and $E_i = E \cap (V_i \times V_i)$ is the set of all edges in E, which have both endpoints in V_i (hence, G_i is the subgraph of G induced by V_i).

For each node $i \in I$, we will compute a table, which we term C_i. The C_i contains an integer value for each subset $S \subseteq X_i$. Hence, when the width of the nice tree decomposition that we are using is at most k, each table C_i contains at most 2^{k+1} values.

Each of these values $C_i(S)$, for $S \subseteq X_i$, equals the maximum weight of an independent set $W \subseteq V_i$ in G_i such that $X_i \cap W = S$, i.e. $C_i(S)$ is the maximum weight of an independent set W in the graph G_i such that all vertices in S belong to W and all vertices in $X_i - S$ do not belong to W. In case no such independent set exists (in particular, when there are two adjacent vertices in S), we set $C_i(S) = -\infty$.

The algorithm will compute for all nodes $i \in I$ the table C_i: this is done in bottom-up order, i.e. a table for node i is computed after all tables of all descendants of i are computed. In particular, we will use the tables of the children of i to compute the table for node i.

3.1.1. *Computing the table for a leaf node*

If i is a leaf of T, then $|X_i| = 1$, say $X_i = \{v\}$. The table C_i has only two entries, and we can compute these trivially: $C_i(\emptyset) = 0$ and $C_i(\{v\}) = c(v)$.

3.1.2. *Computing the table for an introduce node*

Suppose i is an introduce node with child j. Suppose $X_i = X_j \cup \{v\}$. Note that G_i is formed from G_j by adding v and zero or more edges from v to vertices in X_j. In particular, we have that v is not adjacent to any vertex in $V_j - X_j$.

LEMMA 3. (Introduce nodes). *Let $S \subseteq X_j$.*

(1) $C_i(S) = C_j(S)$.
(2) *If there is a vertex $w \in S$, with $\{v, w\} \in E$, then $C_i(S \cup \{v\}) = -\infty$.*
(3) *If for all $w \in S$, $\{v, w\} \notin E$, then $C_i(S \cup \{v\}) = C_j(S) + c(v)$.*

Proof

(1) This follows directly from the observation that for each $S \subseteq X_i$, we have that the collection of independent sets W in G_i with $W \cap X_i = S$ is the same as the collection of independent sets W in G_j with $W \cap X_i = S$.
(2) If $W \cap X_i = S \cup \{v\}$, then $\{v, w\} \subseteq W$, so W is not an independent set.
(3) Let W be an independent set in G_j with weight $C_j(S)$ and $W \cup X_i = S$, and suppose for all $w \in S$: $\{v, w\} \notin E$. As all neighbours of v in G_i must belong to X_i, $W \cup \{v\}$ is an independent set. Clearly, $(W \cup \{v\}) \cap X_i = S \cup \{v\}$, hence $C_i(S \cup \{v\}) \geq c(W \cup \{v\}) = C_j(S) + c(v)$.

Let W' be an independent set in G_i with $W' \cap X_i = S \cup \{v\}$ and weight $C_i(S \cup \{v\})$. $W' - \{v\}$ is an independent set in G_j with weight $C_j(S \cup \{v\})$-$c(v)$ and $(W' - \{v\}) \cap X_j = S$. So $C_j(S) \geq C_i(S \cup \{v\}) - c(v)$. □

Lemma 3 shows us how to compute the table for an introduce node, when we have the table for the child of the node. Each of the at most 2^{k+1} entries can be computed in $O(k)$ time, as we need to do at most one table lookup and k adjacency checks. With some simple modifications, we can avoid doing these adjacency checks for each entry separately, and compute the entire table in $O(2^{k+1})$ time.

3.1.3. *Computing the table for a Forget node*

The situation for Forget nodes is more or less similar to Introduce nodes, but not symmetric. If i is a forget node with child j, then G_i and G_j are the same graph: only X_i and X_j differ, as there is one vertex that belongs to X_j and not to X_i. Suppose $v \in X_j - X_i$ is this unique vertex.

LEMMA 4 (Forget nodes). *Let $S \subseteq X_i$, $C_i(S) = \max\{C_j(S), C_j(S \cup \{v\})\}$.*

Proof. Observe that $C_i(S)$ is the maximum of the following two terms: the maximum weight over all independent sets W in G_i with $W \cap X_i = S$ and $v \in W$, and the maximum weight over all independent sets W in G_i with $W \cap X_i = S$ and $v \notin W$. As $G_i = G_j$ the first term equals $C_j(S)$, and the second term equals $C_j(S \cup \{v\})$. □

3.1.4. *Computing the table for a join node*

Suppose i is a join node with children j_1 and j_2. Now G_i can be seen as a kind of union of G_{j_1} and G_{j_2}. An independent set in G_i has a part in G_{j_1}, and a part in G_{j_2}; in the calculation we subtract a term to make sure we do not count vertices twice that appear in both parts.

LEMMA 5 *Let i, j_1, j_2 be as above. If $v \in V_{j_1}$, $w \in V_{j_2}$ and v, $w \notin X_i$, then $\{v, w\} \notin E$.*

Proof. This makes essential use of the properties of tree decompositions. Suppose $\{v, w\} \in E$. There must be a bag $X_{i'}$ with $v, w \in X_{i'}$. Suppose, without the loss of generality that i' does not belong to the subtree of T, rooted at j_1. (The case that i' does not belong to the subtree rooted at j_2 is symmetric.) As $v \in V_{j_1}$, there must be a bag $X_{i''}$, $i'' = j_1$ or i'' a descendant of j_1 with $v \in X_{i''}$. Now, as i is on the path from i' to i'' in T, $v \in X_{i'}$ and $v \in X_{i''}$, we must have that $v \in X_i$, which is a contradiction. □

LEMMA 6 (Join nodes). *Let* $S \subseteq X_i$, $C_i(S) = C_{j_1}(S) + C_{j_2}(S) - c(S)$.

Proof. Consider an independent set W in G_i with weight $C_i(S)$ such that $W \cap X_i = S$. $W \cap V_{j_1}$ is an independent set in G_{j_1} with $W \cap X_{j_1} = W \cap X_i = S$; likewise for $W \cap X_{j_2}$ in G_{j_2}. We have $W \cap V_{j_1} \cap V_{j_2} = W \cap X_i = S$. Now

$$\begin{aligned} C_i(S) &= c(W) \\ &= c(W \cap V_{j_1}) + c(W \cap V_{j_2}) - c(W \cap V_{j_1} \cap V_{j_2}) \\ &\leq C_{j_1}(S) + C_{j_2}(S) - c(S). \end{aligned}$$

Suppose we have independent sets W_1 in G_j with weight $C_{j_1}(S)$ such that $W_1 \cap X_{j_1} = S$, and W_2 in G_{j_2} with weight $C_{j_2}(S)$ such that $W_2 \cap X_{j_2} = S$. We claim that $W_1 \cup W_2$ is an independent set in G_i. Consider two vertices v, $w \in W_1 \cup W_2$. As W_1 and W_2 are independent sets, v and w are not adjacent when they both belong to W_1 or both belong to W_2. So suppose without the loss of generality that $v \in W_1$, $w \in W_2$. If neither v and w belongs to X_i, then v and w are not adjacent, by Lemma 5. If $v \in X_i$, then $v \in S$, so $v \in W_2$, and now v and w are not adjacent as W_2 is independent. Similarly, when $w \in X_i$. We have now

$$\begin{aligned} C_i(S) &\geq c(W_1 \cup W_2) \\ &= c(W_1) + c(W_2) - c(W_1 \cap W_2) \\ &= C_{j_1}(S) + C_{j_2}(S) - c(S). \end{aligned}$$

The result now follows. □

We leave it as a simple algorithmic exercise to the reader to see that one can compute $c(S)$ for all subsets $S \subseteq X_i$ in total time $O(2^{|X_i|})$. Then, with Lemma 6, we can compute the table C for join node i in $O(2^{|X_i|})$ time, assuming the tables for the two children of C_i are known.

3.1.5. *Putting it all together*

A *postorder tree walk* is an ordering of the nodes of a tree, such that each node is later in the ordering than any of its descendants. There are simple and well-known algorithms to find a postorder tree walk of a given tree in linear time.

In our algorithm for Weighted Independent Set on graphs given with a tree decomposition, we compute tables for nodes in postorder, i.e. first we determine a postorder tree walk for the tree T, and then we compute for each node $i \in I$ the table C_i, in this postorder. In this way, all tables for the children of i are already computed before we compute C_i, and we thus can use the methods described above to compute these tables.

Suppose r is the root of T. After all other tables have been computed, we compute the table C_r. Given this table, we can determine the maximum weight of an independent set in G.

LEMMA 7. *The maximum weight of an independent set in G is* $\max_{S \subseteq X_r} C_r(S)$.

Proof. Note that $G_r = (V, E) = G$. The lemma thus follows trivially. □

For each node i, the time we spend on computing the table for node i is bounded by $O(2^{|X_i|})$; for the root, we spend an additional time of $O(2^{|X_r|})$ to find the answer to the problem, using Lemma 7. As we assume we have a nice tree decomposition of width at most k and with $O(n)$ bags, the total time is bounded by $O(2^k \cdot n)$.

THEOREM 8. *Given a graph $G = (V, E)$, weights $c(v)$ for all vertices $v \in V$, and a nice tree decomposition of G with $O(n)$ nodes of width k, there is an algorithm that determines the maximum weight of an independent set in G in $O(2^k \cdot n)$ time.*

It is also possible to solve the *construction* variant of the Weighted Independent Set problem using this method, i.e. the variant of the problem where we have as output an independent set W of maximum weight. As is usual for dynamic programming, we can do this with additional bookkeeping, using also $O(2^k \cdot n)$ time. Here, we skip the details.

3.2. More dynamic programming algorithms using tree decompositions

The dynamic programming algorithm for Weighted Independent Set is a simple example of a technique with wide applicability. Many well-known graph and network problems can be solved in $O(f(k) \cdot n)$ time when the graph is given with a (nice) tree decomposition of width k (with $O(n)$ nodes), for some function f. As such tree decompositions can be found in $O(f'(k) \cdot n)$ time for some other function f' (see Section 4), we have that these problems are FPT if we take the treewidth of the input graph as the parameter.

Examples of which problems can be solved in linear time on graphs of bounded treewidth include classical problems like Hamiltonian Circuit, Chromatic Number (vertex colouring), Vertex Cover, Steiner Tree and many others. See for instance Telle and Proskurowski [26] for several vertex partitioning problems, Koster *et al.* [27] for partial constraint satisfaction problems (in particular frequency assignment problems) or Arnborg and Proskurowski [28], Bern *et al.* [5], Wimer *et al.* [29] for some of the pioneering papers with these types of algorithms. See Arnborg [30] for an early survey on treewidth and algorithms that use tree decompositions.

Figure 4 shows the theoretical and actual table sizes (i.e. number of entries) considered in the dynamic programming algorithm to solve an exemplary partial constraint satisfaction problem. The theoretical sizes would go beyond the size of memory in state-of-the- art computers. With (pre) processing

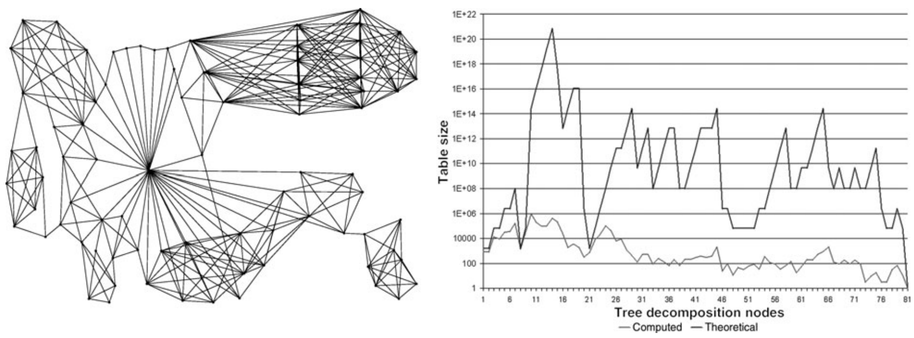

FIGURE 4. Partial constraint satisfaction graph and actual table sizes versus theoretical table sizes during dynamic programming algorithm (note: logarithmic scale).

and reduction techniques the actual table sizes could be kept within the main memory size (see Koster *et al.* [27] for further details). A systematic investigation of memory requirements for dynamic programming algorithms on graphs of small treewidth has been carried out by Aspvall *et al.* [31].

There are also general characterisations of large classes of problems that all allow linear time algorithms on graphs of bounded treewidth. The most successful of these have been the notion of Monadic Second-Order Logic (MSOL), and extensions of these. Courcelle [32] has shown that for each graph property that can be formulated in MSOL, there is a linear time algorithm that verifies if the property holds for a given graph G if we have a bounded width tree decomposition of G. This result has been extended a number of times. For instance, in Arnborg *et al.* [33] and Borie *et al.* [6], it is shown that the result can be extended to optimisation problems, like Independent Set. The result was further generalized by Courcelle and Mosbah [34]. A more extensive overview of MSOL and its applications can be found in Hliněný *et al.* [15].

Important examples of the use of tree decompositions for solving problems can also be found in the area of probabilistic networks. Probabilistic networks are a technology underlying several decision support systems. A central problem for such networks is the inference problem: while this problem is $\#\mathcal{P}$-hard and its decision variant is $\#\mathcal{P}$-complete Cooper [35], it can be solved in linear time when a tree decomposition of bounded width is given of the moralization[1] of the network — this algorithm by Lauritzen and Spiegelhalter [36] is employed in many modern systems for probabilistic networks.

3.3. Establishing fixed-parameter tractability

Treewidth can be used in several cases to quickly establish that a problem is FPT. We give two examples. In both cases, there exist faster algorithms, but the argument using treewidth has the benefit of simplicity.

[1]A moralization of a directed graph $G = (V, A)$ is the undirected graph, obtained by adding an edge between each pair of distinct vertices that are head of arcs with the same tail, and then dropping direction of edges.

First, we consider the Longest Cycle problem. Here, we are given an undirected graph $G = (V, E)$, and an integer k, and ask if G has a cycle of at least k edges. The following algorithm using linear time for fixed k has been designed by Fellows and Langston [37] (see also Bodlaender [38]). Without the loss of generality, suppose G is connected. First, build a depth-first search spanning tree $T = (V, F)$ of G. If this spanning tree has a back edge that spans at least $k - 1$ levels, then we have a cycle of at least k edges: the back edge, and the path on T between the endpoints of the back edge. Otherwise, we can build a tree decomposition of G in the following way: we use T as tree in this tree decomposition, and for each $v \in V$, we let X_v consist of v and the (at most) $k - 2$ direct ancestors in the tree. One can verify that this is a tree decomposition of G of width at most $k - 2$. The longest cycle problem is one of the problems that can be solved in linear time when we have a tree decomposition of bounded width, so we use this algorithm on the just constructed tree decomposition and obtain a linear time algorithm for Longest Cycle when k is a fixed parameter (see for instance Gabow [39] for recent work on the Longest Cycle problem).

The second example is the Feedback Vertex Set problem. Here we are given an undirected graph $G = (V, E)$ and an integer k, and ask for a set of vertices W of size at most k that is a feedback vertex set, i.e. $G[V - W]$ is a forest.

Lemma 9. *If G has a feedback vertex set of size at most k, then the treewidth of G is at most $k + 1$.*

Proof. Suppose W is a feedback vertex set of size $\ell \leq k$. As $G[V - W]$ is a forest, it has treewidth at most one. Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of $G[V - W]$. Now, we add all vertices of W to each bag: $(\{X_i' | i \in I\}, T = (I, F))$ with for all $i \in I$: $X_i' = X_i \cup W$ is a tree decomposition of G of width at most $k + 1$. □

As Feedback Vertex Set can be solved in linear time on graphs of bounded treewidth, we can use the following FPT algorithm for Feedback Vertex Set. First, test if G has treewidth at most $k + 1$. If not, we know that the minimum size of a feedback vertex set is at least $k + 2$. Otherwise, we build a tree decomposition of minimum width (see Section 4), and then solve the problem optimally using dynamic programming on this tree decomposition. For fixed k, the algorithm uses linear time.

Much faster FPT algorithms for Feedback Vertex Set have been designed recently (see, e.g. Guo *et al.* [40] and Kanj *et al.* [41]). See also the discussion in Downey and Fellows [17].

For several other problems, fixed-parameter tractability can be established using treewidth. Several such examples can for instance be found in Downey and Fellows [17].

4. FINDING TREE DECOMPOSITIONS

The algorithms that have been discussed in the previous section assumed that a tree decomposition of small width was given together with the graph. This raises the issue: how fast can we determine the treewidth of a graph, and find tree decompositions with optimal (or close to optimal) width? The Treewidth problem: given a graph $G = (V, E)$, and an integer $k < |V|$, determine if the treewidth of G at most k, is $\mathcal{NP}$-complete (Arnborg *et al.* [42]). Several special cases are also $\mathcal{NP}$-complete, e.g. bipartite graphs (the treewidth of a graph does not change if we subdivide each edge), co-bipartite graphs (Arnborg *et al.* [42]), or graphs of bounded degree (Bodlaender and Thilikos [43]).

4.1. FPT and exact algorithms

As we want to find tree decompositions of small width, the *fixed parameter* case is most significant here, i.e. for a fixed k, we want to determine if the treewidth of a given graph G is at most k, and if so, find a tree decomposition of width at most k. This problem can be solved in linear time (Bodlaender [44]), i.e. Treewidth belongs to FPT. However, the constant factor of that algorithm, hidden in the O-notation is too large to make this algorithm useful in practice (see the experimental evaluation by Röhrig [45]). A polynomial time algorithm by Arnborg *et al.* [42] runs in $O(n^{k+2})$ time. A modified version has been implemented by Shoikhet and Geiger [46] to resolve the treewidth of randomly generated partial k-trees (with $k = 10$) of up to 100 vertices.

A branch and bound algorithm based on vertex-ordering has been proposed by Gogate and Dechter [47]. The worst-case time complexity of this algorithm is $O(n^{n-k})$ and thus is not an FPT or polynomial time algorithm since n and k occur in the exponent. Further exact algorithms have running times of $O(2^n p(n))$ (Arnborg *et al.* [4] and Bodlaender *et al.* [48]) and $O(1.8899^n p(n))$ (Fomin *et al.* [49] and Villanger [50]). Here $p(n)$ denotes a polynomial in n. As the running time of such algorithms is dominated by the exponential term, the polynomial in n is typically ignored in this branch of algorithm theory and the O^* notation is used (see Woeginger [51] for a survey). An algorithm that requires $O^*(2^n)$ time has been designed and evaluated by Bodlaender *et al.* [48]. It has been shown to perform very well on small size graphs (provided sufficient memory).

4.2. Approximating treewidth

Unless $\mathcal{P} = \mathcal{NP}$ we cannot expect to have a polynomial time algorithm that computes the treewidth exactly. Instead of designing an exact, but exponential time algorithm for treewidth, we also can aim at obtaining in polynomial time tree decompositions whose width is not necessarily optimal, but (hopefully or provably) close to optimal. Several different approaches have been proposed for this problem. Several approximation algorithms which give a guarantee on the width of the returned tree decomposition and that use a technique, sometimes known as *nested dissection* have been designed. An early result of this type was given by Bodlaender *et al.* [52], who designed a polynomial time algorithm that outputs a tree decomposition whose width is $O(\log k)$ times the treewidth of the input graph. More recently, Amir [53, 54] and Bouchitté *et al.* [55] gave polynomial time algorithms that guarantee a tree decomposition of width $O(k \log k)$, i.e. an $(\log k)$-approximation. The currently best known is by Feige *et al.* [56]: one can find in polynomial time for graphs of treewidth k a tree decomposition of width $O(k\sqrt{\log k})$.

There are algorithms that have a constant approximation ratio, with a running time that is polynomial in the number of vertices, but exponential in the treewidth (e.g. Amir [53, 54], Lagergren [57] and Reed [58]). It is not known whether there exist constant approximation algorithms that run in time polynomial in n and k.

There are not only approximation algorithms that give a guarantee, but there are also a number of heuristic algorithms for treewidth for which no approximation ratio is known. Several of these algorithms are based on the relationship between treewidth and chordal graphs.

It is a folklore result that for each clique $C \subseteq V$ there must be a node $i \in I$ such that $C \subseteq X_i$ (see, e.g. Bodlaender and Möhring [59]). Hence, the treewidth can be bounded from below by the maximum clique size minus 1. Now, suppose we have a vertex $v \in V$ that together with its neighbours $N(v) := \{w \in V | \{v, w\} \in E\}$ induces a clique. Consider the optimal tree decomposition for $G - v$. Since $N(v)$ induces a clique in $G - v$ as well, for at least one node j of this tree decomposition we have $N(v) \subseteq X_j$. If we attach a node i^* to j with $X_{i^*} = N(v) \cup \{v\}$ we have a tree decomposition for G with width max $\{|N(v)|, tw(G - v)\}$. Since the treewidth of a subgraph can be at most the treewidth of the original graph, we found an optimal tree decomposition this way.

Now, suppose G has the property that its vertices can be ordered $v_1, \ldots, v_n$ such that the neighbours of v_i in $G[v_i, \ldots, v_n]$ induce a clique for all $i = 1, \ldots, n$. Then, we can recursively apply the above technique to find an optimal tree decomposition. Hence, Treewidth can be solved in polynomial time on these graphs. The following definition and lemma reveal the graph-theoretical interpretation of such graphs.

DEFINITION 10 Let $G = (V, E)$ be a graph.

- *G is called chordal if every cycle of at least four edges has a chord.*
- *v is called a simplicial vertex if $N(v) \cup \{v\}$ induces a clique.*
- *A perfect elimination scheme of a graph $G = (V, E)$ is an ordering of the vertices $v_1, \ldots, v_n$ such that for all $i = 1, \ldots, n$, v_i is simplicial in $G[v_i, \ldots, v_n]$*

LEMMA 11. (Gavril [60], see also Golumbic [61]). *A graph G is chordal if and only if there exists a perfect elimination scheme.*

So for chordal graphs H, Treewidth can be solved in polynomial time by first constructing a perfect elimination scheme and next an optimal tree decomposition. The width of the tree decomposition is $tw(H) = \omega(H) - 1$, where $\omega(H) = \max_{i=1,\ldots,n} |N(v_i)| + 1$ is the size of the maximum clique in H.

For non-chordal graphs, we can analogously construct a good tree decomposition by constructing a *chordalization* (also called *triangulation*) of G, i.e. a supergraph $H = (V, E \cup F)$ that is chordal (where F is a set of edges added compared to G). In fact, given an optimal tree decomposition (X, T), the graph $(V, E \cup F)$ with $F := \{\{v, w\} \notin E | \exists i \in I, v, w \in X_i\}$ is chordal.

The following 'folklore' lemma follows from old results on chordal graphs from Gavril [62] and Rose [63, 64]; for more details, see e.g., Bodlaender [13].

LEMMA 12. *Let G be a graph, and let $\mathcal{H}$ be the set of all chordalizations of G. Then, $tw(G) = \min_{H \in \mathcal{H}} \omega(H) - 1$.*

Many treewidth heuristics are based on this lemma. For chordal graphs, a number of recognition algorithms are known, e.g. two variants of Lexicographic Breadth First Search: LEX-P and LEX-M (Rose *et al.* [65]), two variants of Maximum Cardinality Search: MCS (Tarjan and Yannakakis [66]) and MCS-M (Berry *et al.* [67]). If G is chordal, a perfect elimination scheme is found by all these algorithms. If G is not chordal, the algorithms can be adapted so that a number of edges are added in order to obtain a chordal supergraph. This approach was first proposed by Dechter and Pearl [68]. LEX-M and MCS-M have the property that the returned chordalization is inclusion-minimal, i.e. removal of an edge from F returns in a non-chordal graph. Computational experiments with these algorithms with respect to the width of the tree decomposition are reported by Koster *et al.* [69] as well as on Treewidthlib [70].

The set of edges F added to a non-chordal graph G in order to obtain a chordalization is typically called the *fill-in*. The problem Fill-In is to determine the minimum number of edges to be added to a graph G such that the result is chordal. Fill-In is, like Treewidth, $\mathcal{NP}$-hard (Yannakakis [41]). Two simple heuristics to determine chordal supergraphs with low fill-in are *greedy fill-in* (GFI) and *minimum degree fill-in* (MDFI). The GFI selects repeatedly a vertex v for which the number of edges to be added among its neighbours to obtain a simplicial vertex is minimal, adds those edges to G, and removes v temporarily. The MDFI does the same except that it selects repeatedly a vertex of minimum degree (see TreewidthLib [71], Bachoore and Bodlaender [72], and Clautiaux *et al.* [73, 74] for computational results and fine tuning of these algorithms).

Starting with a (non-optimal) tree decomposition, several ways exist to improve the decomposition. In Koster [75], improvements are obtained by replacing one of the maximum sized bags with smaller ones, preserving all conditions of a tree decomposition. For this, a set of vertices separating at least two vertices in an auxiliary graph are computed. The algorithm can start with the trivial tree decomposition consisting of a single node or with a tree decomposition compiled by another algorithm.

Finally, meta-heuristics have been applied to find good tree decompositions. Tabu search (Clautiaux *et al.* [74]), simulated annealing (Kjærulff [76]), and genetic algorithms (Larrañaga *et al.* [77]) have so far been applied on the Treewidth problem or closely related problems.

4.3. Lower bound algorithms for treewidth

Another way to approach the Treewidth problem is to bound the treewidth from below. In this way, no tree decomposition is obtained but a good estimate of the true treewidth of a graph might obtained. Moreover, high-quality lower bounds are helpful within branch-and-bound approaches to bound the size of the search tree. Also a high lower bound on the treewidth for a particular class of graphs indicates that the applicability of the dynamic programming methodology described in Section 3 is limited, as the running time is usually exponential in the treewidth.

Recently, significant effort has attended to the problem of computing lower bounds on treewidth. Key points that describe the situation as addressed for most of these efforts are as follows.

- The treewidth of graphs is closed under taking subgraphs and minors (i.e. the treewidth cannot increase by taking a subgraph or minor).
- However, easy-to-compute lower bounds for treewidth are *not* closed under taking subgraphs and minors.

Thus, instead of computing a lower bound for the original graph only, we would like to compute it for all subgraphs or minors as well. Or even better, to compute it for a subgraph or minor for which the maximum is achieved.

Consider the minimum degree $\delta(G){:=}\min_{v\in V}|N(v)|$ of a graph G. Scheffler [78] proved that $\delta(G) \leq tw(G)$. Hence, we can define two more lower bounds:

$$\delta D(G) := \max_{H\subseteq G} \delta(H) \quad \text{and} \quad \delta C(G) := \max_{H\subseteq G} \delta(H),$$

where $H\subseteq G$ ($H\preceq G$) denotes that H is a subgraph (minor) of G. Since the minimum degree can increase by taking subgraphs or minors, the values $\delta D(G)$ and $\delta C(G)$ can be significantly higher than $\delta(G)$. Experiments indeed confirm this (Gogate and Dechter [47] and Bodlaender *et al.* [79]), although $\delta C(G)$ can can only be approximated (from below) due to its $\mathcal{NP}$-hardness as graph parameter itself. For other lower bounds, similar behaviour could be observed (Bodlaender *et al.* [79] and Koster *et al.* [80]).

One remarkable result has been obtained by Lucena [81]. He showed that the MCS algorithm mentioned in the previous section can also be used for lower bounding. If the fill-in edges are not added on the fly, the highest degree observed along the algorithm turns out to be a lower bound for treewidth. Bodlaender and Koster [82] proved that determining this bound is $\mathcal{NP}$-complete. Also in this case combining the algorithm with taking minors substantially increases the achieved bounds in practice (Bodlaender *et al.* [79]).

Another vital idea to improve lower bounds for treewidth is based on the following result.

THEOREM 13. (Bodlaender [83]). *Let $G = (V, E)$ be a graph with $tw(G) \leq k$ and $\{v, w\} \notin E$. If there exist at least $k+2$ vertex disjoint paths between v and w, then $\{v, w\} \in F$ for every chordalisation $H = (V, F)$ of G with $\omega(H) \leq k$.*

Hence, if we know that $tw(G) \leq k$ and there exist $k+2$ vertex disjoint paths between v and w, adding $\{v, w\}$ to G should not hamper the construction of a tree decomposition with small width. Clautiaux *et al.* [73] explored this result in a creative way. First, they compute a lower bound ℓ on the treewidth of G by any of the above methods (e.g. $\ell = \delta C(G)$). Next, they assume $tw(G) \leq \ell$ and add edges $\{v, w\}$ to G for which there exist $\ell+2$ vertex disjoint paths in G. Let G' be the resulting graph. Now, if it can be shown that $tw(G') > \ell$ by a lower bound computation on G', our assumption that $tw(G) \leq \ell$ is false. Hence, $tw(G) > \ell$ or stated equally $tw(G) \geq \ell + 1$: an improved lower bound for G is determined. This procedure can be repeated until it is not possible anymore to prove that $tw(G') > \ell$ (which of course does not imply that $tw(G') = \ell$). Experiments with this algorithm can be found in Clautiaux *et al.* [73] and Bodlaender *et al.* [79] (see also Treewidthlib [70]). In many cases optimality could be proved by combining the described lower and upper bounds.

For (close-to) planar graphs, the above described lower bounds are typically far from the real treewidth. For planar graphs, we can profit from Lemma 2. Treewidth is bounded from below by branchwidth and branchwidth can be computed in polynomial time on planar graphs (see Hliněný *et al.* [15]). Hence, a polynomial time computable lower bound for treewidth of planar graphs is obtained. Another lower bound for (close-to) planar graphs was obtained by Bodlaender *et al.* [84] by exploiting the *brambles* concept, introduced by Seymour and Thomas [85].

5. EXPLOITING TREEWIDTH ON PLANAR GRAPHS

In this section, we show how treewidth can be used to derive algorithms for several problems on planar graphs. Also, this section discusses a connection between the treewidth of planar graphs and the famous planar separator theorem by Lipton and Tarjan [86], see also Lipton and Tarjan [87]. Several of the results can be reformulated and obtained using the notion of branchwidth instead of treewidth — this often also gives somewhat better running times for the algorithms. This reformulation can usually quickly be seen from Lemma 2.

A very useful tool for these algorithms on planar graphs is the *ratcatcher algorithm* by Seymour and Thomas [88]. This algorithm computes the branchwidth of a planar graph in polynomial time and can also be used to find a branch decomposition of a planar graph of optimal width in polynomial time. Hicks [89, 90] has performed an experimental evaluation of the algorithm, that shows that this algorithm indeed can be used in practice. Thus, in polynomial time we can obtain a tree decomposition of width at most 1.5 k, for any planar graph of treewidth k. It is an open question whether the treewidth of planar graphs can be computed in polynomial time (or is $\mathcal{NP}$-hard).

5.1. Treewidth of planar graphs

Two well-known results in algorithmic graph theory are the following. Say a set S is a 1/2-balanced separator in a graph $G = (V, E)$, if each connected component of $G[V - S]$ has at most $1/2n$ vertices.

THEOREM 14. (The Planar Separator Theorem — Lipton and Tarjan [86]). *Every planar graph G has a 1/2-balanced separator S of size $O(\sqrt{n})$.*

THEOREM 15. *Every planar graph G has treewidth $O(\sqrt{n})$.*

Theorems 14 and 15 are equivalent, in the sense that each can be simply proved from the other. Such proofs can be found, e.g. in Bodlaender [13]. For instance, a planar graph of treewidth k has a 1/2-balanced separator of size at most $k+1$. In the other direction, a tree decomposition can be found using a technique known as nested dissection. Finding the tree decomposition can be done in two ways: either we follow the steps from the constructive proofs of Theorems 14 and 15 or we use the ratcatcher algorithm of Seymour and Thomas [88].

Theorem 14 can be used to obtain faster exponential time algorithms for $\mathcal{NP}$-hard problems on planar graphs. For example, it directly follows that we have an algorithm that solves Weighted Independent Set on planar graphs in $O^*(c^{\sqrt{n}})$ time for some constant c. Similar algorithms exist for other problems, such as Dominating Set, Vertex Cover, etc. Clearly, if a problem has an algorithm solving it in $O(c^k \cdot n)$ time when a tree decomposition of width k is given, then it can be solved on planar graphs in $O^*(c'^{\sqrt{n}})$ time (c, c' constants). For some problems, such as Hamiltonian Circuit, Steiner Tree, Connected Dominating Set, it is not known whether such $O(c^k \cdot n)$ algorithms exist for general graphs, but for planar graphs, the existence of such algorithms, and hence of $O^*(c'^{\sqrt{n}})$ exact algorithms can be shown (Dorn *et al.* [91]).

For some problems, we can obtain even faster algorithms: here the running time only depends on the parameter of the problem. The most famous example is here the Dominating Set problem. It can be shown that if a planar graph G has a dominating set of size at most k, then its treewidth is $O(\sqrt{k})$. Thus, we can use the following algorithm for testing whether a given planar graph has a dominating set of size at most k. First, determine the branchwidth using the ratcatcher algorithm. If it is larger than the given $O(\sqrt{k})$ bound $(+1)$, then the treewidth is also larger than this $O(\sqrt{k})$ bound, and we know there is no dominating set of size at most k. Otherwise, we have a tree decomposition of width $O(\sqrt{k})$ and thus can solve the dominating set problem optimally in $O(c^{\sqrt{k}} \cdot n)$ time, for some constant time. The total time is thus $O(c^{\sqrt{k}} \cdot n + p(n))$ time, c a constant and p a polynomial. The first algorithm of this type was obtained by Alber *et al.* [92]; this result has been extended and improved (with respect to the constant c) several times.

5.2. Approximation algorithms on planar graphs

There is a general method first given by Baker [93] to obtain for several problems on planar graphs a polynomial time approximation scheme. We illustrate the method on the example of Weighted Independent Set.

Given a plane embedding of a planar graph $G = (V, E)$, we divide its vertices into layers $L_1, L_2, \ldots, L_T$ in the following way. All vertices that are incident to the exterior face are in layer L_1. For $i \geq 1$, suppose we remove from the embedding all vertices in layers $L_1, \ldots, L_i$, and their incident edges. All vertices that are then incident to the exterior face are in layer L_{i+1}. L_T is thus the last nonempty layer. A plane graph that has an embedding where the vertices are in k layers is called *k-outerplanar*. A proof of the following result can for instance be found in Bodlaender [13]. The proof is constructive, i.e. the corresponding tree decomposition can be found in polynomial time.

LEMMA 16. *Let G be a k-outerplanar graph. Then the treewidth of G is at most 3k − 1.*

We have now set the stage for a description of a polynomial time approximation scheme for Weighted Independent Set on planar graphs.

Suppose we want to achieve a performance ratio $\varepsilon > 0$, and are given a planar graph $G = (V, E)$. We first find an arbitrary plane embedding of G, and compute the collection of layers $L_1, \ldots, L_T$. Set $k = \lceil 1/\epsilon \rceil$. For each $i \in \{1, \ldots k\}$, let $G_i = (V_i, E_i)$ be the graph obtained by removing from G all vertices on levels $L_i, L_{i+k}, L_{i+2k}, \ldots$, i.e. $G_i = G[\cup_{j, j \bmod k \neq i} L_j]$. We discuss below that we can compute the maximum weighted independent set in G_i in polynomial time. The algorithm computes such a set for each $i \in \{1, \ldots, k\}$, and then reports the one with maximum weight over these k possibilities.

To compute the maximum weight independent set of a graph G_i, note that each connected component of G_i contains at most $k - 1$ levels, hence each connected component of G_i, and hence also G_i itself, is ℓ- outerplanar for some $\ell \leq k - 1$, and thus G_i has treewidth at most $3k - 1$. We can build a tree decomposition of G of width at most $3k - 1$ in polynomial time, e.g. by following the construction of the proof in Bodlaender [13], using an algorithm for the fixed parameter case of the treewidth problem (see Section 4), or using the ratcatcher algorithm of Seymour and Thomas [85] and transforming the branch decomposition to a tree decomposition. Using this tree decomposition, we can solve the Weighted Independent Set problem on G_i then in $O(2^{3k-1}\, n)$ time. So, for fixed ϵ, we have a polynomial time algorithm; using the algorithm of Bodlaender [44] for finding the tree decompositions, the algorithm is linear.

We now show that for every graph $G = (V, E)$, the algorithm outputs an independent set whose weight is at least $(1 - \epsilon)$ times the maximum weight of an independent set in G. Suppose W is an independent set of G of maximum weight. Let $W_i = W \cap V_i$ be those vertices in W that are in G_i. Note that each vertex in W belongs to exactly $k - 1$ of the sets $W_1, W_2, \ldots, W_k$, and hence $\sum_{i=1}^{k} c(W_i) = (k - 1) \cdot c(W)$. Thus, there must be an $i \in \{1, \ldots, k\}$ with $c(W_i) \geq (k - 1)/k \cdot c(W)$. W_i is an independent set of G_i, and hence the maximum weighted independent set in G_i has weight at least $c(W_i) \geq (k - 1)/k \cdot c(W) \geq (1 - \epsilon)c(W)$. So, the algorithm outputs an independent set of weight at least $1 - \epsilon$ times the maximum weight of an independent set in G.

A similar scheme can be used for many other problems. Some problems (like dominating set) need a small variation on the scheme: instead of working with a subgraph G_i, we solve the problem for a collection of $(k + 1)$-outerplanar graphs, each overlapping in one layer with the next one, i.e. we look to the graphs $G[L_1 \cup \cdots \cup L_i]$, $G[L_i \cup \cdots \cup L_{i+k}]$, $G[L_{i+k} \cup \cdots \cup L_{i+2k}]$, etc (for a more in-depth overview, see e.g. Demaine and Hajiaghayi [94]).

5.3. Extensions to larger classes of graphs

The techniques discussed here for planar graphs can be extended to several other, more general classes of graphs, e.g. graphs that can be embedded on a fixed surface, graphs do not have a fixed graph H as minor, etc. (see e.g. Demaine and Hajiaghayi [94] for an overview of these and related topics).

6. TREEWIDTH HORIZONS

In this paper, we have surveyed the concept of treewidth in the context of FPT algorithms. First of all, the concept provides a powerful tool for determining the fixed-parameter tractability of general $\mathcal{NP}$-hard combinatorial optimization problems. For graphs of bounded treewidth, in many cases there exists a dynamic programming algorithm that runs in time polynomial (and often linear) in the size of the graph, but exponential in the treewidth.

Second, recent research has exposed the strong potential of the concept of bounded treewidth, and the algorithm design opportunities this presents, for potently addressing the challenges posed by NP-hard combinatorial optimization problems, in realistic computing situations. On the one hand, the tool kit of algorithms to compute a good tree decomposition or lower bound on the treewidth has been enlarged substantially in recent years (cf. Section 4). On the other hand, more and more researchers have investigated the possibility for applying treewidth in innovative ways to help solve their problems for realistic datasets, in a wide range of application areas, including the solving of huge integer programs, satisfiability and constraint satisfaction problems, query processing and computational biology.

Besides the fixed-parameter tractability of certain combinatorial problems, (theoretical) research on treewidth includes its generalization to matroids (Hliněný and Whittle [95]), and the consideration of *hypertree width* for hypergraphs (Gottlob *et al.* [96]) within the field of database theory and artificial intelligence. For several applications, a *weighted* version of treewidth (Eijkhof *et al.* [97]) or a different measure of the cost of a tree decomposition (Bodlaender and Fomin [98]) is of interest as these better approximate the space and time requirements of the subsequent dynamic programming algorithm.

ACKNOWLEDGEMENT

We thank Mike Fellows and an anonymous referee for very helpful comments on this paper.

REFERENCES

[1] Harary, F. (1969) *Graph Theory*. Addison-Wesley, Reading, MA.

[2] Hedetniemi, S.M., Hedetniemi, S.T. and Laskar, R. (1985) Domination in trees: models and algorithms. *Graph Theory with Applications to Algorithms and Computer Science*, 423–442, New York, Wiley.

[3] Mitchell, S.L. and Hedetniemi, S.T. (1979) Linear algorithms for edge-coloring trees and unicyclic graphs. *Inf. Process. Lett.*, **9**, pp. 110–112.

[4] Shiloach, Y. (1979) A minimum linear arrangement algorithm for undirected trees. *SIAM J. Comput.*, **8**, pp. 15–32.

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

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

[7] Kikuno, T., Yoshida, N. and Kakuda, Y. (1982) A linear algorithm for the domination number of a series-parallel graph. *Discret. Appl. Math.*, **5**, pp. 299–311.

[8] Takamizawa, K., Nishizeki, T. and Saito, N. (1982) Linear-time computability of combinatorial problems on series-parallel graphs. *J. ACM*, **29**, pp. 623–641.

[9] Robertson, N. and Seymour, P.D. (2004) Graph minors. XX. Wagner's conjecture. *J. Comb. Theory Series B*, **92**, pp. 325–357.

[10] Robertson, N. and Seymour, P.D. (1983) Graph minors. I. Excluding a forest. *J. Comb. Theory Series B*, **35**, pp. 39–61.

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

[12] Robertson, N. and Seymour, P.D. (1991) Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, **52**, pp. 153–190.

[13] Bodlaender, H.L. (1998) A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sci.*, **209**, pp. 1–45.

[14] Bodlaender, H.L. and van Antwerpen-de Fluiter, B. (2001) Parallel algorithms for series parallel graphs and graphs with treewidth two. *Algorithmica*, **29**, pp. 543–559.

[15] Hliněný, P., Oum, S., Seese, D. and Gottlob, G. (2007) Width parameters beyond tree-width and their applications. The Computer Journal, this issue.

[16] Hicks, I.V., Koster, A.M.C.A. and Kolotoğlu, E. (2005) Branch and tree decomposition techniques for discrete optimization. In Smith, J.C. (ed), *INFORMS Annual Meeting, TutORials 2005, INFORMS Tutorials in Operations Research Series*, chapter1, pp. 1–29.

[17] Downey, R.G. and Fellows, M.R. (1998) *Parameterized Complexity*. Springer.

[18] Flum, J. and Grohe, M. (2006) *Parameterized Complexity Theory*. Springer.

[19] Niedermeier, R. (2006) *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press.

[20] Arnborg, S., Courcelle, B., Proskurowski, A. and Seese, D. (1993) An algebraic theory of graph reduction. *J. ACM*, **40**, pp. 1134–1164.

[21] Bodlaender, H.L. and van Antwerpen-de Fluiter, B. (2001) Reduction algorithms for graphs of small treewidth. *Inf. Comput.*, **167**, pp. 86–119.

[22] Kloks, T. (1994) *Treewidth. Computations and Approximations*. Lecture Notes in Computer Science, **vol. 842**. Springer-Verlag, Berlin.

[23] Thorup, M. (1998) Structured programs have small tree-width and good register allocation. *Inf. Comput.*, **142**, pp. 159–181.

[24] Gustedt, J., Mæhle, O.A. and Telle, J.A. The treewidth of Java programs. In Mount, D.M. and Stein, C. (eds), *Proc. 4th Int. Workshop on Algorithm Engineering and Experiments*, pp. 86–97. Lecture Notes in Computer Science, **vol. 2409**, Springer-Verlag.

[25] Burgstaller, B., Blieberger, J. and Scholz, B. (2004) On the tree width of Ada programs. In Lamosí, A. and Strohmeier, A. (eds), *Proc. Ada-Europe Int. Conf. Reliable Software Technologies*, pp. 78–90. Lecture Note in Computer Science, vol. 3063, Springer-Verlag.

[26] Telle, J.A. and Proskurowski, A. (1997) Algorithms for vertex partitioning problems on partial *k*-trees. *SIAM J. Discret. Math.*, **10**, pp. 529–550.

[27] Koster, A.M.C.A., van Hoesel, S.P.M. and Kolen, A.W.J. (2002) Solving partial constraint satisfaction problems with tree decomposition. *Networks*, **40**, pp. 170–180.

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

[29] Wimer, T.V., Hedetniemi, S.T. and Laskar, R. (1985) A methodology for constructing linear graph algorithms. *Congr. Numer.*, **50**, pp. 43–60.

[30] Arnborg, S. (1985) Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, **25**, pp. 2–23.

[31] Aspvall, B., Proskurowski, A. and Telle, J.A. (2000) Memory requirements for table computations in partial *k*-tree algorithms. *Algorithmica*, **27**, pp. 382–394.

[32] Courcelle, B. (1990) The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Inf. Comput.*, **85**, pp. 12–75.

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

[34] Courcelle, B. and Mosbah, M. (1993) Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sci.*, **109**, pp. 49–82.

[35] Cooper, G.F. (1990) The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, **42**, pp. 393–405.

[36] Lauritzen, S.J. and Spiegelhalter, D.J. (1988) Local computations with probabilities on graphical structures and their application to expert systems. *J. R. Stat. Soc.. Series B (Methodological)*, **50**, pp. 157–224.

[37] Fellows, M.R. and Langston, M.A. (1989) On search, decision and the efficiency of polynomial-time algorithms. *Proc 21st Annual Symp. Theory of Computing*, pp. 501–512.

[38] Bodlaender, H.L. (1993) On linear time minor tests with depth first search. *J. Algorithms*, **14**, pp. 1–23.

[39] Gabow, H.N. (2007) Finding paths and cycles of superpolylogarithmic length. *SIAM J. Comput.*, **36**, pp. 1648–1671.

[40] Guo, J., Gramm, J., Hüffner, F., Niedermeier, R. and Wernicke, S. (2005) Improved fixed-parameter algorithms for two feedback set problems. In Dehne, F.K.H.A., López-Ortiz, A., and Sack, J.-R. (eds), *Proc. 9th Int. Workshop on Algorithms and Data Structures WADS 2005*, pp. 158–168. Lecture Notes in Computer Science, **vol. 3608**, Springer-Verlag.

[41] Kanj, I.A., Pelsmajer, M.J. and Schaefer, M. (2004) Parameterized algorithms for feedback vertex set. In Downey, R.G. and Fellows, M.R. (eds), *Proc. 1st Int. Workshop on Parameterized and Exact Computation, IWPEC 2004*, pp. 235–248. Lecture Notes in Computer Science, **vol. 3162**, Springer-Verlag.

[42] Arnborg, S., Corneil, D.G. and Proskurowski, A. (1987) Complexity of finding embeddings in a *k*-tree. *SIAM J. Algebraic Discrete Methods*, **8**, pp. 277–284.

[43] Bodlaender, H.L. and Thilikos, D.M. (1997) Treewidth for graphs with small chordality. *Discret. Appl. Math.*, **79**, pp. 45–61.

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

[45] Röhrig, H. (1998) Tree decomposition: a feasibility study. Master's Thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[46] Shoikhet, K. and Geiger, D. (1997) A practical algorithm for finding optimal triangulations. *Proc. Nat. Conf. on Artificial Intelligence (AAAI'97)*, pp. 185–190. Morgan Kaufmann.

[47] Gogate, V. and Dechter, R. (2004) A complete anytime algorithm for treewidth. *Proc. 20th Annual Conf. Uncertainty in Artificial Intelligence UAI-04*, Arlington, VA, USA, pp. 201–208, AUAI Press.

[48] Bodlaender, H.L., Fomin, F.V., Koster, A.M.C.A., Kratsch, D. and Thilikos, D.M. (2006) On exact algorithms for treewidth. In Azar, Y. and Erlebach, T. (eds), *Proc. 14th Annual European Symp. Algorithms, ESA 2006*, pp. 672–683. Lecture Notes in Computer Science, **vol. 4168**, Springer-Verlag.

[49] Fomin, F.V., Kratsch, D. and Todinca, I. (2004) Exact (exponential) algorithms for treewidth and minimum fill-in. In Diaz, J., Karhumäki, J., Lepistö, A., and Sanella, D. (eds), *Proc. 31st Int. Colloquium on Automata, Languages and Programming, ICALP 2004*, pp. 568–580. Lecture Notes in Computer Science, **vol. 3124**, Springer-Verlag.

[50] Villanger, Y. (2006) Improved exponential-time algorithms for treewidth and minimum fill-in. In Correa, J.R., Hevia, A. and Kiwi, M.A. (eds), *Proc. 7th Latin American Theoretical Informatics Symposium, LATIN 2006*. pp. 800–811. Lecture Notes in Computer Science, **vol. 3887**, Springer-Verlag.

[51] Woeginger, G.J. (2003) Exact algorithms for NP-hard problems: A survey. In Jünger, M., Reinelt, G., and Rinaldi, G. (eds), *Combinatorial Optimization: 'Eureka, you shrink'*, pp. 185–207. Lecture Notes in Computer Science, **vol. 2570**, Springer-Verlag.

[52] Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H. and Kloks, T. (1995) Approximating treewidth, pathwidth, frontsize, and minimum elimination tree height. *J. Algorithms*, **18**, pp. 238–255.

[53] Amir, E. (2001) Efficient approximation for triangulation of minimum treewidth. In Breese, J.S. and Koller, P. (eds.), *Proc 17th Conf. Uncertainity in Artificial Intelligence, UAI'01.*, pp. 7–15, Morgan Kaufmann.

[54] Amir, E. Approximation algorithms for treewidth. *Algorithmica*. To appear.

[55] Bouchitté, V., Kratsch, D., Müller, H. and Todinca, I. (2004) On treewidth approximations. *Discret. Appl. Math.*, **136**, pp. 183–196.

[56] Feige, U., Hajiaghayi, M. and Lee, J.R. (2005) Improved approximation algorithms for minimum-weight vertex separators. *Proc. 37th Annual Symp. Theory of Computing, STOC 2005*, pp. 563–572. ACM Press.

[57] Lagergren, J. (1996) Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, **20**, pp. 20–44.

[58] Reed, B. (1992) Finding approximate separators and computing tree-width quickly. *Proc. of the 24th Annual Symp. Theory of Computing*, pp. 221–228, New York, ACM Press.

[59] Bodlaender, H.L. and Möhring, R.H. (1993) The pathwidth and treewidth of cographs. *SIAM J. Discret. Math.*, **6**, pp. 181–188.

[60] Gavril, F. (1974) The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory Series B*, **16**, pp. 47–56.

[61] Golumbic, M.C. (1980) *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

[62] Gavril, F. (1972) Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, **1**, pp. 180–187.

[63] Rose, D.J. (1970) Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, **32**, pp. 597–609.

[64] Rose, D.J. (1972) A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Reed, R.C. (ed), *Graph Theory and Computing*, Academic Press. pp. 183–217.

[65] Rose, D.J., Tarjan, R.E. and Lueker, G.S. (1976) Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, **5**, pp. 266–283.

[66] Tarjan, R.E. and Yannakakis, M. (1984) Simple linear time algorithms to test chordiality of graphs, test acyclicity of graphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, **13**, pp. 566–579.

[67] Berry, A., Blair, J., Heggernes, P. and Peyton, B. (2004) Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, **39**, pp. 287–298.

[68] Dechter, R. and Pearl, J. (1989) Tree clustering for constraint networks. *Acta Inform.*, **38**, pp. 353–366.

[69] Koster, A.M.C.A., Bodlaender, H.L. and van Hoesel, S.P.M. (2001) Treewidth: Computational experiments. In Broersma, H., Faigle, U., Hurink, J. and Pickl, S. (eds), *Electronic Notes in Discrete Mathematics*, **vol. 8**, pp. 54–57. Elsevier Science Publishers.

[70] van den Broek, J.-W. and Bodlaender, H.L. (2004). TreewidthLIB. http://www.cs.uu.nl/people/hansb/treewidthlib.

[71] Yannakakis, M. (1981) Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discret. Methods*, **2**, pp. 77–79.

[72] Bachoore, E.H. and Bodlaender, H.L. (2005) New upper bound heuristics for treewidth. In Nikoletseas, S.E. (ed), *Proc. 4th Int. Workshop Experimental and Efficient Algorithms WEA 2005*, pp. 217–227. Lecture Notes in Computer Science, **vol. 3503**, Springer-Verlag.

[73] Clautiaux, F., Carlier, J., Moukrim, A. and Négre, S. (2003) New lower and upper bounds for graph treewidth. In Rolim, J.D.P. (ed), *Proc. Int. Workshop on Experimental and Efficient Algorithms, WEA 2003*, pp. 70–80. Lecture Notes in Computer Science, **vol. 2647**, Springer-Verlag.

[74] Clautiaux, F., Moukrim, A., Négre, S. and Carlier, J. (2004) Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Oper. Res.*, **38**, pp. 13–26.

[75] Koster, A.M.C.A. (1999). *Frequency assignment - Models and algorithms*. PhD Thesis, University Maastricht, Maastricht, The Netherlands.

[76] Kjærulff, U. (1992) Optimal decomposition of probabilistic networks by simulated annealing. *Stat. Comput*, **2**, pp. 2–17.

[77] Larrañaga, P., Kuijpers, C.M.H., Poza, M. and Murga, R.H. (1997) Decomposing Bayesian networks: Triangulation of the moral graph with genetic algorithms. *Statistics and Computing (UK)*, **7**, pp. 19–34.

[78] Scheffler, P. (1989) Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme. PhD Thesis, Akademie der Wissenschaften der DDR, Berlin.

[79] Bodlaender, H.L., Koster, A.M.C.A. and Wolle, T. (2006) Contraction and treewidth lower bounds. *J. Graph Algorithms Appl.*, **10**, pp. 5–49.

[80] Koster, A.M.C.A., Wolle, T. and Bodlaender, H.L. (2005) Degree-based treewidth lower bounds. In Nikoletseas, S.E. (ed), *Proc. 4th Int. Workshop on Experimental and Efficient Algorithms WEA 2005*, pp. 101–112. Lecture Notes in Computer Science, **vol. 3503**, Springer-Verlag.

[81] Lucena, B. (2003) A new lower bound for tree-width using maximum cardinality search. *SIAM J. Discret. Math.*, **16**, pp. 345–353.

[82] Bodlaender, H.L. and Koster, A.M.C.A. (2007) On the maximum cardinality search lower bound for treewidth. In Hromkovič, J., Nagl, M. and Westfechtel, B. (eds), *Discret. Appl. Math.*, **155**, pp. 1348–1372.

[83] Bodlaender, H.L. (2003) Necessary edges in *k*-chordalizations of graphs. *J. Comb Optim.*, **7**, pp. 283–290.

[84] Bodlaender, H.L., Grigoriev, A. and Koster, A.M.C.A. (2005) Treewidth lower bounds with brambles. In Brodal, G.S. and Leonardi, S. (eds), *Proc. 13th Annual European Symp. Algorithms, ESA 2005*, pp. 391–402. Lecture Notes in Computer Science, **vol. 3669**, Springer-Verlag.

[85] Seymour, P.D. and Thomas, R. (1993) Graph searching and a minimax theorem for tree-width. *J. Comb. Theory Series B*, **58**, pp. 239–257.

[86] Lipton, R.J. and Tarjan, R.E. (1979) A separator theorem for planar graphs. *SIAM J. Appl. Math.*, **36**, pp. 177–189.

[87] Lipton, R.J. and Tarjan, R.E. (1980). Applications of a planar separator theorem. *SIAM J. Comput.*, **9**, pp. 615–627.

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

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

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

[91] Dorn, F., Penninkx, E., Bodlaender, H.L. and Fomin, F.V. (2005) Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In Brodal, S. and Leonardi, S. (eds), *Proc. 13th Annual European Symp. Algorithms ESA' 2005*, pp. 95–106. Lecture Notes in Computer Science, **vol. 3669**, Springer-Verlag.

[92] Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T. and Niedermeier, R. (2002) Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, **33**, pp. 461–493.

[93] Baker, B.S. (1994) Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, **41**, pp. 153–180.

[94] Demaine, E.D. and Hajiaghayi, M. (2007) The bidimensionality theory and its algorithmic applications. *The Computer Journal*, *this issue*.

[95] Hliněný, P. and Whittle, G. (2006) Matroid tree-width. *European J. Combinatorics*, **27**, pp. 1117–1128.

[96] Gottlob, G., Leone, N. and Scarcello, F. (2002) Hypertree decompositions and tractable queries. *J. Comp. Syst. Sci.*, **64**, pp. 579–627.

[97] Eijkhof, F.v.d., Bodlaender, H.L. and Koster, A.M.C.A. (2007) Safe reduction rules for weighted treewidth. *Algorithmica*, **47**, pp. 138–158.

[98] Bodlaender, H.L. and Fomin, F.V. (2004) Tree decompositions with small cost. *Discret. Appl. Math.*, **145**, pp. 143–154.