Treewidth: Characterizations, Applications, and Computations

Hans L. Bodlaender

Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
hansb@cs.uu.nl

Abstract. This paper gives a short survey on algorithmic aspects of the treewidth of graphs. Some alternative characterizations and some applications of the notion are given. The paper also discusses algorithms to compute the treewidth of given graphs, and how these are based on the different characterizations, with an emphasis on algorithms that have been experimentally tested.

1 Introduction

For approximately a quarter of a century, the notion of treewidth is used in many graph algorithmic and graph theoretic studies. In the 1980's, several researchers invented independently notions that were strongly related, or equivalent: partial k-trees (Arnborg and Proskurowski, e.g., [3,7]), treewidth and tree decompositions (Robertson and Seymour [56]), clique trees (Lauritzen and Spiegelhalter [47]), recursive graph classes (Borie [26,27]), k-terminal recursive graph classes (Wimer [64,65]), decomposition trees (Lautemann [48]), and context-free graph grammars (Lengauer and Wanke [49]). See also [61]. Of these, the notions *treewidth* and *tree decompositions* became the most used (followed by *partial k-trees*).

This short survey discusses some applications of the notion of treewidth, and some recent experimental work on computing the treewidth of a given graph. Some of the algorithms to compute the treewidth are based on a different characterization of the notion. A few of such equivalent notions are also briefly surveyed.

Other surveys on treewidth are e.g., [13,15,16,43,54,55].

2 Definitions and Characterizations

We assume the reader to be familiar with standard notions from graph theory. Throughout this paper, $n = |V|$ denotes the number of vertices of graph $G = (V, E)$. A graph $G = (V, E)$ is *chordal*, if every cycle in G of length at least four has a chord, i.e., there is an edge connecting two non-consecutive vertices in the cycle. A *triangulation* of a graph $G = (V, E)$ is a graph $H = (V, F)$ that contains G as subgraph ($F \subseteq E$) and is chordal. A triangulation H is a *minimal triangulation* of G if there does not exist a triangulation H' of G with H' a

F.V. Fomin (Ed.): WG 2006, LNCS 4271, pp. 1–14, 2006.

proper subgraph of H. A set of vertices S is a *separator*, if $G[V - S]$ is not connected. A separator is an *inclusion minimal separator*, if it does not contain another separator as proper subset.

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 collection of subsets of V, called* bags*, and $T = (I, F)$ a tree, such that for all $v \in V$, there exists an $i \in I$ with $v \in X_i$, for all $\{v, w\} \in E$, there exists 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 subgraph (subtree) of T.*

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

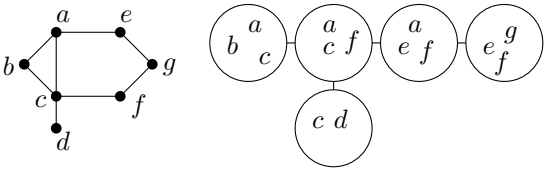

Fig. 1. A Graph and a Tree Decomposition of Width 2

An example of a tree decomposition is shown in Figure 1.

A permutation π of the vertices of a graph is called an *elimination order*. Given an elimination order π of the graph $G = (V, E)$, the *fill-in graph* of G with respect to π is constructed as follows: for $i = 1$ to n, we add an edge between each pair of higher numbered vertices of the i'th vertex in the order. An example is shown in Figure 2. When we 'eliminate" vertices with number 1, 3, 5, 6, or 7, no edges are added. The middle graph is obtained when we eliminate the vertex with number 2; the last one when we eliminate the vertex with number 4.

Elimination orderings and triangulations give alternative characterizations of treewidth. See e.g., [15] for a proof.

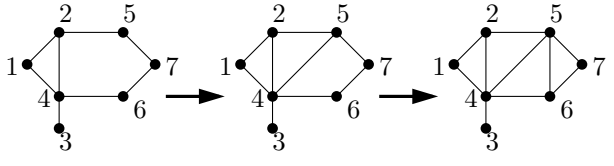

Fig. 2. Obtaining the Fill-in Graph

Theorem 1. *Let $G = (V, E)$ be a graph, and $k < |V|$. The following are equivalent.*

- *G has treewidth at most k.*
- *There is an elimination ordering π of G, such that each vertex has at most k higher numbered neighbors in the fill-in graph with respect to π.*
- *G has a triangulation with maximum clique size at most $k+1$.*
- *G has a minimal triangulation with maximum clique size at most $k+1$.*

Several algorithms to compute the treewidth of a graph exploit these alternative characterizations. There is an interesting connection between alternative characterizations of chordal graphs, and alternative characterizations of treewidth. Compare the following classic results with Theorem 1.

Theorem 2 (See [37,58,39]). *Let G be a graph. The following are equivalent.*

- *G is chordal.*
- *G is the intersection graph of subtrees of a tree, i.e., G has a tree decomposition such that each bag is a clique.*
- *G has a perfect elimination scheme, i.e., a permutation of the vertices such that for each vertex, its higher numbered neighbors form a clique.*

3 Applications

In this section, a number of algorithmic applications of treewidth are discussed.

3.1 Problems Restricted to Graphs of Small Treewidth

Many problems that are NP-hard (and some that are PSPACE-hard or #P-hard) on arbitrary graphs become linear or polynomial time solvable when the inputs are restricted to graphs with some constant upper bound on the treewidth. These include many of the most famous graph problems, like Hamiltonian Circuit, Independent Set, Vertex Cover, etc. Most well known is the result of Courcelle [33] that each problem that can be formulated in *Monadic Second Order Logic* (MSOL) can be solved in linear time on graphs of bounded treewidth. For extensions of this result, see e.g., [5,27]. Also, several problems have tailor-made algorithms that solve them in linear or polynomial time, assuming a constant upper bound on the treewidth of the input graphs. There is a large number of papers with such a result, e.g., [7,40,45,63,66]. Such algorithms, sometimes based on a notion strongly related to treewidth (pathwidth or branchwidth) have also been used successfully in experimental settings: for the Frequency Assignment problem and for constraint satisfaction (Koster [42,45]); for the Traveling Salesman Problem (Cook and Seymour [32]); for problems on planar graphs (Dorn, see [1]); for problems on graphs of small pathwidth (Pönitz and Tittmann, see e.g., [34]).

These algorithms usually use dynamic programming and have the following structure. First, a tree decomposition of small width is constructed. Then, one bag is chosen as root. In a bottom-up order, for each bag of the tree decomposition, a table is computed. Given the table of the root bag, one can find the

answer to the problem quickly. Exploiting that bags are (usually) separators, for computing a table, only some local information on the vertices in the bag, and the tables of the children of the bag are needed. The time to compute a bag typically is exponential on the size of the bag (and its children), but does not depend on the size of the graph. Thus, when the treewidth is bounded by a constant, the algorithm uses linear time. As the algorithm uses time, exponential in the width of the used tree decomposition, this motivates the research for efficient algorithms to compute the treewidth of graphs. See Section 4.

3.2 Probabilistic Networks

Probabilistic networks are the underlying technology of many modern decision support systems. In a probabilistic network, we have a directed acyclic graph, and for each vertex of the graph, a table of conditional probabilities. Each vertex represents a statistical variable, and the (in)dependencies are modeled by the graph structure. A central problem in the use of such networks is *inference*: given values for some *observed* variables, we want to compute the probability distributions for the other variables. This problem is #P-hard. The most used method to solve this problem is as follows. First, the *moralized graph* is build: we add an edge between each pair of vertices that are tail of edges with the same head, and then drop all directions of edges. Then, the algorithm of Lauritzen and Spiegelhalter [47] solves the problem in linear time when the moralized graph has small treewidth. The latter appears to be the case for many probabilistic networks from real-world domains.

3.3 Electrical Networks

In a recent book on graph theory, Bollobás [25] describes the theory of computing the resistance of electrical networks. This theory traces back to a paper by Kirchhoff from 1847 [41]. We have a graph (possibly with parallel edges) with two special vertices, which we will call s and t. Each edge has a resistance: a positive number. Given a potential difference between s and t, we ask how much electrical current will flow through the network, and how much the resistance of the entire network will be.

Three laws that allow to transform networks to smaller, equivalent networks are given. The first law (*series rule*) allows us to remove vertices $\neq s, t$ that have degree 2. The second law (*parallel rule*) allows us to remove parallel edges. The third law (*star-triangle rule* or *Y-Δ rule*) allows us to remove vertices $\neq s, t$ of degree three. See Figure 3. In the first rule, $R = R_1 + R_2$; in the second, we have $\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2}$, and in the third, $S = R_1R_2 + R_1R_3 + R_2R_3$.

When we can apply the rules until we have only a single edge from s to t, then these three rules allow us to compute the resistance of the electrical network. We can use the notion of treewidth to determine for which networks there exists such a series of applications of these three rules that yield a single edge. Note that the order in which we apply rules to vertices matters. Consider the graph shown in Figure 3.3. Two different orders of selecting vertices for reduction

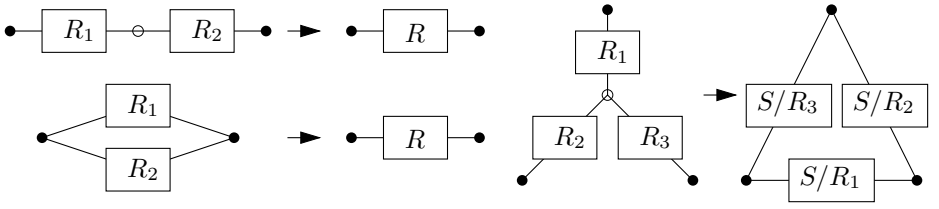

Fig. 3. Series, Parallel, and Star-Triangle Rules for Reduction of Electrical Networks

are used. Removal of parallel edges is assumed and not explicitly shown. If we first eliminate c, then we obtain a clique with five vertices, and no rule applies. However, selecting vertices in the order c, b, d, e yields a single edge $\{s,t\}$. We may assume that the graph $G' = (V, E \cup \{s,t\}$ is a biconnected graph.

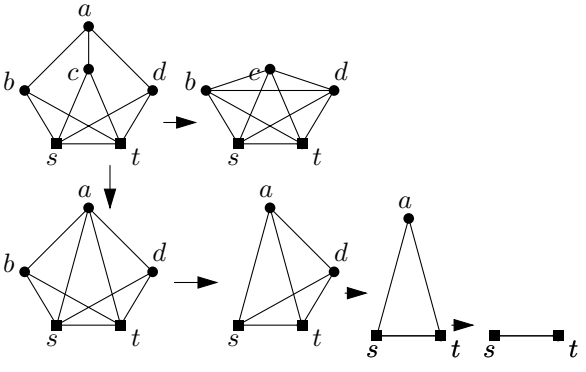

Fig. 4. Different Orders of Applying Reduction Rules of Electrical Networks

(A biconnected component of G' that does not contain both s and t is irrelevant for the computation of the resistance of G.) The next result shows that we can check in linear time (cf. Section 4) if there is an order in which an electrical network can be reduced to a single edge using the rules, and if so, find such an order.

Proposition 1. *Let $G' = (V, E \cup \{s,t\})$ be a biconnected graph. There exists a series of rule applications that reduces G to the single edge $\{s,t\}$, if and only if G' has treewidth at most three.*

Proof. Suppose we have an series of rule applications that reduces G to a single edge $\{s,t\}$. Now, take the elimination ordering of G, that puts vertices in the order in which they are removed, and then ends with s, t. For each vertex $v \notin \{s,t\}$, its neighbors (two or three) at the moment it is removed are higher numbered neighbors in the fill-in graph, and thus we have an elimination ordering

of G (and of G') such that each vertex has at most three higher numbered neighbors. So G and G' have treewidth at most three.

Suppose G' has treewidth at most three. G' hence is a subgraph of a chordal graph H with maximum clique size four (see Section 2). Repeat the following step. As H is chordal, H has two non-adjacent simplicial vertices (see [39]. A vertex is simplicial if its neighborhood is a clique). As $\{s, t\}$ is an edge in G' and hence in H, there is a vertex $v \notin \{s, t\}$ that is simplicial. As v with its neighbors forms a clique, v has degree at most three. As G' is biconnected, v has degree two or three. So, we can apply the series or star-triangle rule to v, and then remove possibly created parallel edges. Repeat on $H[V - \{v\}]$ until we have the single edge $\{s, t\}$. □

4 Computations

As discussed in Section 3.1, there are many algorithms that first find a tree decomposition, and then use it to solve the problem at hand. As the second step usually is exponential in the width of the tree decomposition, there is a need for efficient algorithms to compute the treewidth of a given graph, and to find tree decompositions with optimal or close to optimal width. In this section, such algorithms will be discussed. Many papers have been written on this topic. Here we focus on algorithms that have been experimentally evaluated.

As for any graph parameter, we can classify the algorithms to compute treewidth into *exact algorithms*, *upper bound algorithms*, and *lower bound algorithms*. In addition, *preprocessing* is an important technique, which is in several cases of great help.

4.1 Upper Bounds

In this section, we discuss a number of algorithms that give upper bounds on the treewidth of the input graphs. Some algorithms have a guaranteed approximation ratio (and hence, can be seen to be lower bound algorithms as well). A typical example are the algorithms by Amir [2]. (See e.g., also [10].) Other heuristics do not have such a guarantee, but often give tree decompositions with close to optimal width. *Construction heuristics* take a graph, and build a tree decomposition (or, a different representation, e.g., an elimination ordering.) *Improvement heuristics* take a tree decomposition (or elimination ordering), and stepwise try to change it to one with smaller and smaller width.

Construction Heuristics. Some construction heuristics, like the algorithms of Amir [2] use a technique, known as *nested dissection*. Here, repeatedly separators are constructed in specific subgraphs of the input graph. Other construction heuristics build an elimination ordering of the graph. As discussed in Section 2, a permutation of the vertices give us an upper bound on the treewidth. Thus, we can use any algorithm or heuristic to build a permutation of the vertices as construction heuristic. Often used heuristics are the *Minimum Degree* and

Minimum Fill-In heuristics (explained below), and algorithms that build elimination orderings that are used for the recognition of chordal graphs: Maximum Cardinality Search, Lexicographic Breadth First Search [62,11,58].

The *Minimum Degree* heuristic is here explained in terms of tree decompositions. It can also be seen as a heuristic to obtain an elimination ordering. A short proof of the following folklore fact can e.g., be found in [24].

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

The *Minimum Degree* heuristic works as follows. If $|V| = 1$, we take a trivial tree decomposition with one bag. Otherwise, choose a vertex $v \in V$ with minimum degree. Let G' be the graph, obtained by turning the set of neighbors of v into a clique, and then removing v. Recursively, build a tree decomposition of G'. This tree decomposition must contain a bag i which contains all neighbors of v (as this set is a clique in G'). Now, add a new bag which consists of v and its neighbors and make it adjacent to i. This is a tree decomposition of G.

In the *Minimum Degree* heuristic, we have chosen a vertex of minimum degree: this makes that the bag with this vertex is as small as possible. Other, related heuristics, make different choices for v. The *Minimum Fill-In* heuristic chooses a vertex v, such that the number of edges added when turning v's neighborhood into a clique is as small as possible. In [8,31], some alternative manners to choose v are investigated.

Improvement Heuristics. The *Minimum Separating Vertex Sets* heuristic of Koster [42] starts with a trivial tree decomposition: one bag containing all vertices, and refines this stepwise, using minimum separators. Other currently proposed improvement heuristics for treewidth use a form of local search. Clautiaux et al. [31] use tabu search. As solution space, they take the set of elimination orderings. Two solutions are neighboring, if they represent different triangulations (compare Theorem 1), and are obtained by moving one vertex to a different position in the ordering. Graph theoretic arguments provide a fast test to see if the triangulation changes. Recently, Koster, Marchal and van Hoesel [44] investigated local search algorithms based on 'flipping' edges in triangulations.

The representation by triangulations can also be used for a *postprocessing* step in combination with many heuristics. Take a tree decomposition of the input graph G, obtained by some heuristic. Transform this into a triangulation H of G. If this is not a minimal triangulation, we can obtain a subgraph H' of H that is a minimal triangulation of G, e.g., with the algorithm of [12]. The corresponding tree decomposition never has a larger treewidth compared to the first one, but sometimes has a smaller treewidth.

4.2 Lower Bound Heuristics

Approximation algorithms with a guaranteed approximation ratio give both an upper bound and a lower bound. There are also a number of heuristics that

provide only lower bounds. Lower bound heuristics have several uses: if a lower bound matches the upper bound provided by a heuristic, we know we have the exact treewidth; if we have a very large lower bound for the treewidth, we know an approach using tree decompositions and dynamic programming as discussed in Section 3.1 will yield slow algorithms; lower bounds can be used to stop branches in a branch and bound algorithm (see Section 4.3.)

If G has treewidth k, then it has a vertex of degree at most k (consider the first vertex of an elimination order). So, the minimum degree is a simple lower bound on the treewidth. It can be improved using the following observation: the treewidth of a subgraph of G is at most the treewidth of G. Thus, the following algorithm, which computes the *degeneracy* of a graph, yields a lower bound on the treewidth: Set $k = 0$. While G is not empty, select a vertex v of minimum degree in G. Set k to the maximum of k and the degree of v, then remove v and its incident edges from G and repeat.

The *contraction* of an edge also does not increase the treewidth: if G' is obtained from G by contracting edge $\{v, w\}$ to x, then a tree decomposition of G can be transformed to one of G' by replacing each occurrence of v or w in a bag by x. Thus, instead of deleting a vertex, we contract it to one of its neighbors. Two strategies perform well here: contracting to a neighbor of minimum degree, or contracting to a neighbor that has few common neighbors. See [23].

Instead of using the minimum degree, one can use different bounds, and possibly also combine these with vertex deletion or contraction [46]: the one-but-smallest degree; if G is not complete, the minimum over all pairs v, w of non-adjacent vertices of the maximum degree of v and w (Ramachandramurthi [52,53]); the maximum number of visited neighbors of a vertex when it is visited by the Maximum Cardinality Search algorithm (Lucena [50], see also [19]). Also, one can run an exact algorithm on a graph, obtained by contracting edges.

A clever method to improve lower bounds, based on adding edges to G was found by Clautiaux et al. [30]. Combining these LBN and LBP-methods with contraction gives further improvement to the lower bounds for many cases [22].

Seymour and Thomas gave a 'min-max' characterization of treewidth, using the notion of *brambles* [59]. This was used in [18] to obtain a new type of lower bound method for treewidth. This method appears to work well in particular for graphs that are planar or 'close to planar'.

4.3 Exact Algorithms

In theory, for each fixed k, there is a linear time algorithm that tests if a given graph G has treewidth at most k, and if so, finds a tree decomposition of width at most k [14]. However, an experimental evaluation has shown that this algorithm is much too slow in practice [57]. Fortunately, some other algorithms compute for many practical cases the treewidth exactly of (not too large) graphs.

One approach is to build an elimination ordering with a branch and bound algorithm. At each branching step of the algorithm, the next vertex in the elimination ordering is chosen. Several rules, e.g., lower bounds, are used to cut off

some branches of the decision tree. See [38,9]. Note that the different representation of treewidth by elimination orderings is again of great use.

Recently, a dynamic programming of the style of the classic Held-Karp algorithm for the Traveling Salesman Problem has been given [17]. Let for a set $S \subseteq V$ of vertices $TWDP(S)$ be the minimum over all elimination orderings that start with the vertices in S in some order, the minimum over all vertices in S of their number of higher numbered neighbors in the fill-in graph. Clearly, by Theorem 1, the treewidth of G equals $TWDP(V)$. One can show that for all $V \subseteq S, V \neq \emptyset, TWDP(S) = \min_{v \in S} \max(TWDP(S - \{v\}), |\{w \in V - S \mid$ there is a path from v to w using only vertices in $\{v\} \cup (V - S)\}|$. Using this in a dynamic programming algorithm with a few addition optimizations leads to an algorithm that can compute the treewidth for graphs with 30 – 60 vertices.

Shoikhet and Geiger [60] has shown that an algorithm of Arnborg et al. [4] can be used to compute the treewidth. This algorithm builds a tree decomposition, and also uses a form of dynamic programming.

If $k \leq 3$, then there are relatively simple and extremely fast algorithms to test if the treewidth is at most k, and if so, find the corresponding tree decompositions. These are based on reduction. Arnborg and Proskurowski [6] give six rules, illustrated in Figure 5. A graph G has treewidth at most three, if and only if it is reduced to the empty graph by repeated application of these rules. This gives a linear time algorithm to test if a graph has treewidth at most three, see [51]. It is also possible to construct a tree decomposition of width at most three if existing. Also, the order in which the vertices are eliminated by the rules gives an elimination ordering where each vertex has three higher numbered neighbors in the fill-in graph; see also [21].

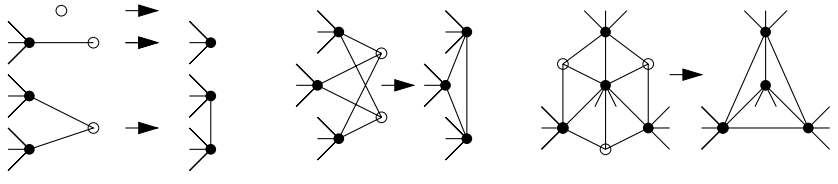

Fig. 5. The Six Reduction Rules for Treewidth Three

4.4 Preprocessing

For many real-world problems, preprocessing is an extremely important technique. For treewidth, two approaches have been used: reduction (or simplification), and splitting (the divide step of divide and conquer). With these approaches, often significant reductions in problem size can be obtained.

Reduction. In [21], the six rules of Arnborg and Proskurowski for treewidth three are used and generalized for preprocessing graphs for computing treewidth.

Besides a graph G, the algorithm maintains a variable *low*, that gives a lower bound on the treewidth of the original input graph. Each rule modifies G and possibly *low*, such that the maximum of *low* and the treewidth of G does not change. Each rule also decreases the size of G. For many graphs from applications, these rules give significant reductions in the size of G. After the preprocessing step, another (e.g., an exact algorithm from Section 4.3) algorithm is used to compute the treewidth of G. The hope is that the size reduction by the preprocessing helps to significantly reduce the time used by this exact algorithm. This technique is also known as *simplification*. Generalizations of the rules were given in [35].

Splitting. A different form of preprocessing is obtained by using *safe separators*, i.e., splitting the graph in different parts. As in a divide and conquer algorithm, the treewidth of each part can be computed separately, and the treewidth of the original graph is the maximum of the treewidth of the parts. Let for a graph G' and vertex set S, $G' + clique(S)$ be the graph, obtained by making S a clique in G', i.e., adding an edge between each pair of non-adjacent vertices in S. In [20], a separator $S \subseteq V$ is safe (for treewidth) in a graph $G = (V, E)$ if the treewidth of G equals the maximum over all connected components W of $G[V - S]$ of the treewidth of $G[W \cup S] + clique(S)$. Thus, if we have a safe separator S, we have as parts all graphs $G[W \cup S] + clique(S)$, for all connected components W of $G[V - S]$. In [20], it is shown that the following sets are safe separators, and can be found, if existing, in polynomial time.

- Separators of size at most one.
- Inclusion minimal separators of size two.
- Inclusion minimal separators S of size at least three, such that no component of $G[V - S]$ contains at least $|V| - 4$ vertices.
- Separators that are a clique.
- Inclusion minimal separators S for which there is a $v \in S$ with $S - \{v\}$ a clique.

5 Conclusions

In this survey, different characterizations and applications of treewidth, and methods to compute the treewidth of a graph were discussed. Experimental and theoretical research benefit from each other. Many of the experimentally tested algorithms are based on interesting combinatorial insights, and the experimental algorithms give rise to new and interesting theoretical questions.

I believe that for testing algorithms experimentally, it is of great importance to make a good selection of the graphs on which to test the algorithms. As random graphs have in general properties that do not need to hold for the graphs encountered in real life applications, I believe one should not rely on only testing the algorithms on randomly generated graphs.

There is a growing number of examples of the fact, that in several cases, the treewidth of graphs can be used to practically solve real-life problems. Of course,

small treewidth of the graphs at hand is needed, but appears that in several problem domains, graphs of small treewidth are sufficiently often encountered.

These is much room for additional work. One intriguing question is whether the algorithm of Bouchitté and Todinca [28,29], or its form of Fomin et al. [36] can be used in a practical setting to compute the treewidth of a given graph.

Acknowledgment

I am very grateful to many colleagues for discussions, help, and cooperations. In particular, I thank Arie Koster without whom most of my work reported here would not have been possible.

References

1. J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Disc. Appl. Math.*, 145:210–219, 2005.
2. E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15, 2001.
3. S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.
4. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8:277–284, 1987.
5. S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.
6. S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.
7. S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.
8. E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 217–227. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.
9. E. H. Bachoore and H. L. Bodlaender. A branch and bound algorithm for exact, upper, and lower bounds on treewidth. In S.-W. Cheng and C. K. Poon, editors, *Proceedings 2nd International Conference on Algorithmic Aspects in Information and Management, AAIM 2006, Lecture Notes in Computer Science, vol. 4041*, pages 255–266, 2006.
10. A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.
11. A. Berry, J. R. S. Blair, and P. Heggernes. Maximum cardinality search for computing minimal triangulations. In L. Kučera, editor, *Proceedings 28th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'02*, pages 1–12. Springer Verlag, Lecture Notes in Computer Science, vol. 2573, 2002.
12. J. R. S. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comp. Sc.*, 250:125–141, 2001.

13. H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.
14. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.
15. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.
16. H. L. Bodlaender. Discovering treewidth. In P. Vojtáš, M. Bieliková, and B. Charron-Bost, editors, *SOFSEM 2005: Theory and Practive of Computer Science: 31st Conference on Current Trends in Theory and Practive of Computer Science*, pages 1–16. Springer-Verlag, Lecture Notes in Computer Science 3381, 2005.
17. H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. In Y. Azar and T. Erlebach, editors, *Proceedings 14th Annual European Symposium on Algorithms ESA 2006*, pages 672–683. Springer Verlag, Lecture Notes in Computer Science, vol. 4168, 2006.
18. H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth lower bounds with brambles. In G. S. Brodal and S. Leonardi, editors, *Proceedings 13th Annual European Symposium on Algorithms, ESA2005*, pages 391–402. Springer-Verlag, Lecture Notes in Computer Science, vol. 3669, 2005.
19. H. L. Bodlaender and A. M. C. A. Koster. On the Maximum Cardinality Search lower bound for treewidth. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Proc. 30th International Workshop on Graph-Theoretic Concepts in Computer Science WG 2004*, pages 81–92. Springer-Verlag, Lecture Notes in Computer Science 3353, 2004.
20. H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. *Disc. Math.*, 306:337–350, 2006.
21. H. L. Bodlaender, A. M. C. A. Koster, and F. v. d. Eijkhof. Pre-processing rules for triangulation of probabilistic networks. *Computational Intelligence*, 21(3):286–305, 2005.
22. H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. In S. Albers and T. Radzik, editors, *Proceedings 12th Annual European Symposium on Algorithms, ESA2004*, pages 628–639. Springer, Lecture Notes in Computer Science, vol. 3221, 2004.
23. H. L. Bodlaender, A. M. C. A. Koster, and T. Wolle. Contraction and treewidth lower bounds. *Journal of Graph Algorithms and Applications*, 10:5–49, 2006.
24. H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Math.*, 6:181–188, 1993.
25. B. Bollobás. *Modern Graph Theory.* Graduate Texts in Mathematics, Springer, New York, 1998.
26. R. B. Borie. *Recursively Constructed Graph Families.* PhD thesis, School of Information and Computer Science, Georgia Institute of Technology, 1988.
27. R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–581, 1992.
28. V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.
29. V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theor. Comp. Sc.*, 276:17–32, 2002.

30. F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J. D. P. Rolim, editor, *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003*, pages 70–80. Springer Verlag, Lecture Notes in Computer Science, vol. 2647, 2003.
31. F. Clautiaux, A. Moukrim, S. Négre, and J. Carlier. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Operations Research*, 38:13–26, 2004.
32. W. Cook and P. D. Seymour. Tour merging via branch-decomposition. *INFORMS J. on Computing*, 15(3):233–248, 2003.
33. B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.
34. K. Dohmen, A. Pönitz, and P. Tittmann. A new two-variable generalization of the chromatic polynomial. *Disc. Math. and Theor. Comp. Sc.*, 6:69–90, 2004.
35. F. v. d. Eijkhof, H. L. Bodlaender, and A. M. C. A. Koster. Safe reduction rules for weighted treewidth. To appear in Algorithmica.
36. F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, ICALP 2004*, pages 568–580, 2004.
37. F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Theory Series B*, 16:47–56, 1974.
38. V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence UAI-04*, pages 201–208, Arlington, Virginia, USA, 2004. AUAI Press.
39. M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
40. K. Jansen and P. Scheffler. Generalized coloring for tree-like graphs. In *Proceedings 18th International Workshop on Graph-Theoretic Concepts in Computer Science WG'92*, pages 50–59, Berlin, 1993. Springer Verlag, Lecture Notes in Computer Science, vol. 657.
41. G. Kirchhoff. Über die Auflösung der Gleichugen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird. *Ann. Phys. Chem.*, 71:497–508, 1847.
42. A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms*. PhD thesis, Univ. Maastricht, Maastricht, The Netherlands, 1999.
43. A. M. C. A. Koster, H. L. Bodlaender, and S. P. M. van Hoesel. Treewidth: Computational experiments. In H. Broersma, U. Faigle, J. Hurink, and S. Pickl, editors, *Electronic Notes in Discrete Mathematics*, volume 8, pages 54–57. Elsevier Science Publishers, 2001.
44. A. M. C. A. Koster, B. Marchal, and C. P. M. van Hoesel. Local search algorithms for treewidth. Work in progress, 2006.
45. A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.
46. A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 101–112. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.
47. S. J. Lauritzen and D. J. Spiegelhalter. 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.

48. C. Lautemann. Decomposition trees: structured graph representation and efficient algorithms. In *Proceedings CAAP'88*, pages 28–39. Springer Verlag, Lecture Notes in Computer Science, vol. 299, 1988.
49. T. Lengauer and E. Wanke. Efficient analysis of graph properties on context-free graph languages. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming, ICALP'88*, pages 379–393. Springer Verlag, Lecture Notes in Computer Science, vol. 317, 1988.
50. B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM J. Disc. Math.*, 16:345–353, 2003.
51. J. Matoušek and R. Thomas. Algorithms for finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.
52. S. Ramachandramurthi. A lower bound for treewidth and its consequences. In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, *Proceedings 20th International Workshop on Graph Theoretic Concepts in Computer Science WG'94*, pages 14–25. Springer Verlag, Lecture Notes in Computer Science, vol. 903, 1995.
53. S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Disc. Math.*, 10:146–157, 1997.
54. B. A. Reed. *Tree width and tangles, a new measure of connectivity and some applications*, volume 241 of *LMS Lecture Note Series*, pages 87–162. Cambridge University Press, Cambridge, UK, 1997.
55. B. A. Reed. *Algorithmic aspects of tree width*, pages 85–107. CMS Books Math. / Ouvrages Math. SMC, 11. Springer, New York, 2003.
56. N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7:309–322, 1986.
57. H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.
58. D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.
59. P. D. Seymour and R. Thomas. Graph searching and a minimax theorem for tree-width. *J. Comb. Theory Series B*, 58:239–257, 1993.
60. K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 185–190. Morgan Kaufmann, 1997.
61. M. M. Sysło. NP-complete problems on some tree-structured graphs: A review. In M. Nagl and J. Perl, editors, *Proc. WG'83 International Workshop on Graph Theoretic Concepts in Computer Science*, pages 342–353, Linz, West Germany, 1983. University Verlag Rudolf Trauner.
62. R. E. Tarjan and M. Yannakakis. Simple linear time algorithms to test chordiality of graphs, test acyclicity of graphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13:566–579, 1984.
63. J. A. Telle and A. Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Disc. Math.*, 10:529 – 550, 1997.
64. T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Dept. of Computer Science, Clemson University, 1987.
65. T. V. Wimer, S. T. Hedetniemi, and R. Laskar. A methodology for constructing linear graph algorithms. *Congressus Numerantium*, 50:43–60, 1985.
66. X. Zhou, K. Fuse, and T. Nishizeki. A linear algorithm for finding $[g, f]$-colorings of partial k-trees. *Algorithmica*, 27:227–243, 2000.