Discovering Treewidth

Hans L. Bodlaender

institute of information and computing sciences, utrecht university

technical report UU-CS-2005-018

www.cs.uu.nl

Discovering Treewidth*

Hans L. Bodlaender†

Abstract

Treewidth is a graph parameter with several interesting theoretical and practical applications. This survey reviews algorithmic results on determining the treewidth of a given graph, and finding a tree decomposition of small width. Both theoretical results, establishing the asymptotic computational complexity of the problem, as experimental work on heuristics (both for upper bounds as for lower bounds), preprocessing, exact algorithms, and postprocessing are discussed.

1 Introduction

About a quarter of a century, the notion of treewidth has now played a role in many investigations in algorithmic graph theory. While for a long time, the use of treewidth was limited to theoretical investigations, and it sometimes was believed that it could not play a role in practical applications, nowadays there is a growing tendency to use it in an actual applied setting.

An interesting example of this practical use of treewidth can be found in the work by Koster, van Hoesel, and Kolen [80], where tree decompositions are used to solve frequency assignment instances from the CALMA project, and other partial constraint satisfaction problems. The most frequent used algorithm to solve the inference problem for probabilistic, or Bayesian belief networks (often used in decision support systems) uses tree decompositions [85]. See e.g., also [2, 47].

Graphs of bounded treewidth appear in many different contexts. For an overview of graph theoretic notions that are equivalent to treewidth, or from which bounded treewidth can be derived, see [17]. Many probabilistic networks appear to have small treewidth in practice. Yamagucki, Aoki, and Mamitsuka [114] have computed the treewidth of 9712 chemical compounds from the LIGAND database, and discovered that all but one had treewidth at most three; the one exception had treewidth four. Thorup [109] showed that the control flow graph of goto-free programs, written in one of a number of common imperative programming languages (like C, Pascal) have treewidth bounded by small constants.

*This is a revised version of a paper that appeared in the proceedings of SOFSEM 2005, Springer-Verlag Lecture Notes in Computer Science, vol. 3381.

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

See also [61]. Treewidth also has been used in graph drawing research, see e.g. [113]. Another interesting application of treewidth is for solving or approximating optimisation problems on planar graphs, or applications of planar graphs, see e.g., [1, 7, 48, 50, 49].

Many problems can be solved in linear or polynomial time when the treewidth of the input graph is bounded. (See e.g., [73, Section 10.4].) Usually, the first step of such an algorithm is to find a tree decomposition of small width. In this paper, we give an overview of algorithms for finding such tree decompositions. Nowadays, much work has been done on this topic, and we now have a rich theory, and intriguing experimental approaches.

After some preliminaries in Section 2, we survey exact algorithms (Section 3), approximation algorithms and upper bound heuristics (Section 4), lower bound heuristics (Section 5), and preprocessing and postprocessing methods (Section 6).

Other overviews of related topics can be found in [10, 12, 13, 16, 17, 42, 43, 51, 52, 54, 56, 74, 89, 90, 97, 98].

2 Preliminaries

The notion of treewidth was introduced by Robertson and Seymour in their work on graph minors [99]. Equivalent notions were invented independently, e.g., a graph has treewidth at most k, if and only if it is a partial k-tree. See [17] for an overview of notions equivalent to or related to treewidth. In this paper, we assume graphs to be undirected and simple. Many results also hold for directed graphs, and often they can be generalised to hypergraphs. $n = |V|$ denotes the number of vertices of graph $G = (V, E)$, $m = |E|$ its number of edges.

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

1. $\bigcup_{i \in I} X_i = V$.
2. For all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$.
3. For all $v \in V$, $T_v = \{i \in I \mid v \in X_i\}$ forms a connected subtree of T.

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

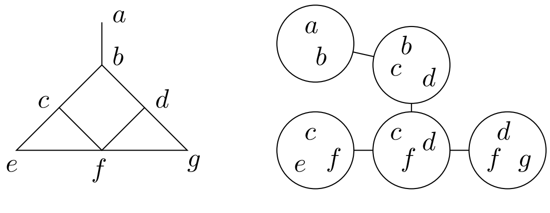

Figure 1: A graph and a tree decomposition of width 2

Having a tree decomposition of small width in general implies that the graph has many separators of small size. E.g., consider a tree decomposition $(\{X_i, i \in I\}, T = (I, F))$, and choose a node $r \in I$ as root of T. Consider some node $i \in I$, and let G_i be the subgraph of G induced by the set V_i of vertices in sets X_j with $j = i$ or j is a descendant of i. Then, the definition of tree decomposition implies that all vertices in G_i that have a neighbour in G that does not belong to G_i belong to X_i. Hence, X_i separates all vertices in $V_i - X_i$ from all vertices in $V - V_i$. This property amongst others enables many dynamic programming algorithms on graphs of bounded treewidth. A useful lemma on tree decompositions (which can be seen as a rephrasing of the Helly property for trees, see [60, 28] is the following.

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

There are several equivalent definitions of the notion of treewidth. The various algorithms for determining the treewidth and finding tree decompositions are based upon different such notions. We review here those that we use later in this paper.

A graph $G = (V, E)$ is *chordal*, if and only if each cycle in G of length at least four has a chord, i.e., an edge between non-successive vertices in the cycle. There are two equivalent definitions of chordality that we will use. 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 $v_i \in V$, its higher numbered neighbours form a clique, i.e., for $j_1 > i$, $j_2 > i$, if $\{v_i, v_{j_1}\} \in E$ and $\{v_i, v_{j_2}\} \in E$, then $\{v_{j_1}, v_{j_2}\} \in E$. A graph $G = (V, E)$ is the *intersection graph of subtrees of a tree*, if and only if there is a tree $T = (I, F)$ and for each vertex $v \in V$ a subtree T_v of T, such that for all $v, w \in V$, $v \neq w$: $\{v, w\} \in E$, if and only if the trees T_v and T_w have at least one vertex in common.

Theorem 2 (See [58, 60]) *Let $G = (V, E)$ be a graph, The following statements are equivalent.*

1. *G is a chordal graph.*
2. *G has a perfect elimination scheme.*
3. *G is the intersection graph of subtrees of a tree.*

A *triangulation* of a graph $G = (V, E)$ is a chordal graph $H = (V, F)$ that contains G as a subgraph: $E \subseteq F$. A triangulation $H = (V, F)$ is *a minimal triangulation*, when there does not exist a triangulation $H' = (V, F')$ with $E \subseteq F' \subset F$ $(F' \neq F)$.

Given a tree decomposition $(\{X_i, i \in I\}, T = (I, F))$ of G, we can build corresponding triangulation $H = (V, F)$: add to G an edge between each non-adjacent pair of vertices $\{v, w\}$ such that there is an $i \in I$ with $v, w \in X_i$. I.e., each bag X_i is turned into a clique. The graph H thus obtained is the intersection graph of subtrees $T_v = T[\{i \in I \mid v \in X_i\}]$ of T, thus chordal. The maximum cliquesize of H is exactly one larger than the width of the tree decomposition (compare with Lemma 1.)

Lemma 3 *The treewidth of a graph G equals the minimum over all triangulations H of G of the maximum clique size of H minus one.*

We can also build a tree decomposition from an ordering $v_1, \ldots, v_n$ of the vertices of the graph G. We first construct a triangulation H of G by the following *fill-in* procedure: initially, $H = G$, and then for $i = 1$ to n, we add to H, edges between yet non-adjacent higher numbered neighbours of v_i. After having done this, $v_1, \ldots, v_n$ is a perfect elimination scheme of H; the model of H as intersection graph of subtrees of a tree can be easily transformed to a tree decomposition of H and of G. Its width equals the maximum over all vertices of its higher numbered neighbours in the ordering in H.

3 Exact algorithms

The Treewidth problem: given a graph G, and an integer k, decide if the treewidth of G is at most k, is NP-complete [4]. This unsettling fact does not prevent us from wanting to compute the treewidth of graphs, and fortunately, in many cases, there are methods to effectively obtain the treewidth of given graphs.

3.1 Special graph classes

There are many results on the complexity of treewidth when restricted to special graph classes. We mention here a few of these. A highly interesting recent result was obtained by Bouchitté and Todinca, who found an algorithm to determine the treewidth of a graph in time, polynomial in the number of its minimal separators [35, 34]. Many graph classes have the property that each graph in the class has a polynomial number of minimal separators, e.g., permutation graphs, weakly chordal graphs.

Other polynomial time algorithms for treewidth for special graph classes can be found in e.g., [23, 29, 37, 38, 46, 76, 75, 77]. NP-completeness results appear amongst others in [30, 62]. See also [92]. Other older results are surveyed in [13].

3.2 Exponential time algorithms

Based upon the results from Bouchitté and Todinca [35, 34], Fomin et al. [57] found an exact algorithm for treewidth that runs in time $O^*(1.9601^n)$ time. (See [112] for the O^* notation and an introduction to exponential time algorithms.)

Algorithms with a running time of $O^*(2^n)$ are easier to obtain: one can show that the algorithm of [4] has this time, or build a dynamic programming algorithm following a technique first established for TSP by Held and Karp [65]; experiments with this dynamic programming algorithm are currently carried out [18].

For small graphs, the treewidth can be computed in practice using *branch and bound*. Experiments have been published by Gogate and Dechter [59]. The algorithm searches for an ordering of the vertices that corresponds to a tree decomposition of small width, see Section 2, i.e., at each step, we select the next vertex in the ordering. Gogate and Dechter establish several rules to cut off branches during branch and bound. The algorithm can also be used as a heuristic, by stopping the branch and bound algorithm at a specific time and reporting the best solution found so far.

3.3 Fixed parameter cases

As we often want to use a tree decomposition for running a dynamic programming algorithm that is exponential in the width, we often want to test if the treewidth is smaller than some given constant k. Much work has been done of this fixed parameter case of treewidth. Here we let k denote the constant for which we want to test if the treewidth is at most k.

The first polynomial time algorithm for the problem was given by Arnborg, Corneil, and Proskurowski [4]. Their algorithm runs in $O(n^{k+2})$ time. A modification of this algorithm has been proposed and successfully experimentally evaluated by Shoikhet and Geiger [106].

Downey and Fellows introduced the theory of *fixed parameter tractability.* A problem with input parameter k and input size n is fixed parameter tractable, when there is a function f and a constant c, such that there is an algorithm that solves the problem in $f(k) \cdot n^c$ time, (in contrast to algorithms using $\Omega(n^{g(k)})$ time for some increasing function g). See [54]. The first result that showed that treewidth is fixed parameter tractable, i.e., solvable in $O(n^c)$ time for some constant c, for fixed treewidth k, was obtained by Robertson and Seymour [99, 100]. This result was fully non-constructive: from the deep results of their graph minor theory, one gets a non-constructive proof that there exists a characterisation that can be tested in $O(n^2)$ time. Later results, by Lagergren [82], Reed [96], Lagergren and Arnborg [83], Bodlaender and Kloks [22], and Bodlaender [14] improved upon either the constructivity or the running time. Finally, in [15], a linear time algorithm was given that checks if the treewidth is at most k, and if so, outputs the corresponding tree decomposition. That algorithm uses about $O(k^3)$ calls to the dynamic programming algorithm from [22], but the hidden constant in the 'O'-notation of this algorithm is horrendous, even for small values of k. Röhrig [101] has experimentally evaluated the linear time algorithm from [15]. Unfortunately, this evaluation shows that the algorithm uses too much time even for very small values of k (e.g., when $k = 4$.) A parallel variant of the algorithm from [15] was given by Bodlaender and Hagerup [21]. A variant with $O(k^2)$ calls to the algorithm of [22] was given by Perković and Reed [93].

The linear time algorithm for fixed k is attractive from a theoretical point of view: in many cases, an algorithm exploiting small treewidth would use the algorithm as a first step. From a practical point of view, the algorithm is useless however due to its large constant factor hidden in the O-notation, and the quest remains for algorithms that are efficient from the implementation viewpoint. Fortunately, several heuristics appear to perform well in practice, as we will see in the next section.

Also, for very small values of k, there are special algorithms. Testing if the treewidth is one is trivially linear (the graph must be a forest), a graph has treewidth at most two, if and only if each biconnected component is a series parallel graph (see e.g., [31]), and testing if a graph is series parallel can be done in linear time by the algorithm of Valdes, Tarjan, and Lawler [111]. Arnborg and Proskurowski [5] give a set of six reduction rules, such that a graph has treewidth at most three, if and only if it can be reduced to the empty graph by means of these rules. These rules can be implemented such that the algorithm runs in linear time, see also [88]. Experiments show that these algorithms run very fast in practice (see e.g. [26]). A more complicated linear time algorithm for testing if the treewidth of a graph is at most 4 has been given by Sanders [103]. As far as I know, this algorithm has not yet been tried out in practice.

4 Approximation algorithms and upper bound heuristics

There are many algorithms that approximate the treewidth. We can distinguish a number of different types, depending on whether the algorithm is polynomial for all values of k, and whether the algorithm has a guaranteed performance.

4.1 Polynomial time approximation algorithms with a performance ratio

We first look at algorithms that are polynomial, even when k is not bounded, and that give a guarantee on the quality of the output. The first such approximation algorithm for treewidth was given in [19]. This algorithm gives tree decompositions with width at most $O(\log n)$ times the optimal treewidth. (See also [74].) It builds a tree decomposition by repeatedly finding balanced separators in the graph and subgraphs of it, see also below. Bouchitté et al. [33] and Amir [3] recently improved upon this result, giving polynomial time approximation algorithms with ratio $O(\log k)$, i.e., the algorithms output a tree decomposition of width $O(k \log k)$ when the treewidth of the input graph is k. It is a long standing and apparently very hard open problem whether there exist a polynomial time approximation algorithm for treewidth with a constant performance ratio.

4.2 Fixed Parameter Approximation Algorithms

There are also several approximation algorithms for treewidth that run in time, exponential in k. They either give a tree decomposition of width at most ck (for some constant c), or tell that the treewidth is more than k. See [3, 8, 53, 82, 96, 100]. See also [73, Section 10.5]. These algorithms more or less share a common structure. Consider the following recursive procedure.

PROCEDURE BuildTD (Graph $G = (V, E)$, vertex set $X \subseteq V$, integer k)
Find a separator S in G such that each component of $G[V - S]$ contains 'few' vertices of X.
If $|S|$ is 'too large', then we *reject* as the treewidth of G is larger than k.
For each connected component $W \subseteq V$ of $G[V - S]$, run BuildTD ($G[W \cup S]$, $S \cup (W \cap X)$).

If at least one of these calls *reject*, then also *reject*.
Otherwise, build a tree decomposition of G such that the root bag contains all vertices of X, using the tree decompositions from the recursive calls.

The step to build the tree decomposition can be done as follows: we take one new bag that contains $S \cup W$, take the disjoint union of the tree decompositions obtained from all calls BuildTD $(G[W \cup S], S \cup (W \cap X)$ for all components W of $G[V - S]$, and make each root bag of such a tree decomposition adjacent to the new bag. One can verify that this gives a tree decomposition.

By using different algorithms to find the separators in the first step of the procedure and different measures of when S is too large, the algorithms from [3, 8, 53, 82, 96, 100] achieve different running times and bounds on the treewidth. A similar algorithmic method is also used in [3, 19, 33, 74] for the approximation algorithms whose running time does not exponentially depend on k.

4.3 Upper Bound Heuristics

Many of the heuristics that have been proposed and are used to find tree decompositions of small width do not have a guarantee on their performance. However, amongst these, there are many that appear to perform very well in many cases.

A large class of these heuristics is based upon the same principle. As discussed in Section 2, a tree decomposition can be build from a linear ordering of the vertices. Thus, we can build in some way a linear ordering of the vertices, run the fill-in procedure, and turn the triangulation into a tree decomposition. Often, one already adds fill-in edges during the construction of the linear order, and so we get the following type of algorithm:

$H = G$;
for $i = 1$ **to** n **do**
  Select a vertex v from H according to some criterion.
  Let v be the ith vertex in vertex ordering π.
  Add an edge between each pair of non-adjacent neighbors of v.
  Remove v and its incident edges from H.
Build the tree decomposition corresponding to π.

Different criteria in this procedure give different treewidth heuristics. The edges added during the procedure are called *fill-in* edges. A very simple heuristic of this type is the Minimum Degree heuristic: we repeatedly select the vertex v with the minimum number of unselected neighbours as the next vertex in the ordering. The Minimum Fill-in heuristic is similar, but now we select a vertex which gives the minimum number of added fill-in edges for the current step. More complicated rules for selecting next vertices have been proposed by Bachoore and Bodlaender [6], and by Clautiaux et al. [39, 40]. In some cases, improvements are thus made upon the simpler Minimum Degree or Minimum Fill-in heuristics.

Also, sometimes orderings generated by algorithms originally invented for chordal graph recognition ([102, 108, 9] have been used as linear ordering to generate the tree decomposition from. These tend to give tree decompositions with larger width, but give for some instances better bounds. See [79] for an experimental evaluation of several of these heuristics.

These heuristics can also be described using tree decompositions only: Select some vertex v (according to the criteria at hand, e.g., the vertex of minimum degree). Build the graph G', by turning the set of neighbours $N(v)$ of v into a clique, and then removing v. Recursively, compute a tree decomposition of G'. By Lemma 1, there must be a bag i^* with $N(v) \subseteq X_i$. Now, add a new node i_v to G, with $X_{i_v} = \{v\} \cup N(v)$, and make i_v adjacent to i^* in the tree. One can check that this gives a tree decomposition of G.

A different type of heuristic was proposed by Koster [78]. The main idea of the heuristic is to start with any tree decomposition, e.g., the trivial one where all vertices belong to the same bag, and then stepwise refine the heuristics, i.e., the heuristic selects a bag and splits it into smaller bags, maintaining the properties of tree decomposition.

There are several algorithms that, given a graph G, make a minimal triangulation of G. While not targeted at treewidth, such algorithms can be used as treewidth heuristic. Recently, Heggernes, Telle, and Villanger [63] found an algorithm for this problem that uses $o(n^{2.376})$ time; many other algorithms use $O(nm)$ time.

See [110] for an online database with some experimental results.

4.4 Heuristics with local search methods

Some work has been done on using stochastic local search methods to solve the treewidth problem or related problems. Kjærulff [72] uses simulated annealing to solve a problem related to treewidth. Genetic algorithms have been used by Larrañaga et al. [84]. Clautiaux et al. [40] use tabu search for the treewidth problem. The running times of these meta heuristics is significantly higher, but good results are often obtained.

4.5 Approximation algorithms for special graph classes

Also, approximation algorithms have been invented with a guarantee on the performance for special graph classes, e.g., a ratio of 2 can be obtained for AT-free graphs [36], and a constant ratio can be obtained for graphs with bounded asteroidal number [33].

5 Lower bound heuristics

It is for several reasons interesting to have good lower bound heuristics for treewidth. They can be used in a subroutines in a branch and bound algorithm (as, e.g., is done in [59]), or in an upper bound heuristic (e.g., as part of the rule to select the next vertex of the vertex ordering [40]), and inform us about the quality of upper bound heuristics. Also, when a lower bound for the treewidth is too high, it may tells us that it is not useful to aim at a dynamic programming algorithm solving a problem with tree decompositions.

5.1 Degree based methods and contraction

It is easy to see that the minimum degree of a vertex in G, and the maximum clique size of G are lower bounds for the treewidth. These bounds are often not very good, and the maximum clique size is NP-hard to compute. An improvement to these bounds is made with the *degeneracy*: the maximum over all subgraphs G' of G of the minimum vertex degree of G' [107, 86]. The degeneracy can be easily computed: repeatedly remove a vertex of minimum degree from the graph, and then report the maximum over the degrees of the vertices when they were removed.

An improvement to the degeneracy can be obtained by instead of removing a vertex, contracting it to one of its neighbours. This idea was found independently by Bodlaender, Koster, and Wolle [27], and by Gogate and Dechter [59]. The MMD+ heuristic thus works as follows: set $\ell = 0$, then repeat until G is empty: find a vertex v of minimum degree d in G; set $\ell = \max(\ell, d)$; contract v to a neighbour (or remove v if v is isolated). In [27], different rules to select the vertex to contract to are explored. The heuristic to select the neighbour of v of smallest degree performs reasonably well, but the heuristic to select the neighbour w of v such that v and w have the smallest number of common neighbours usually gives better lower bounds. (When v and w have a common neighbour x, then contracting v and w causes the two edges $\{v, x\}$ and $\{w, x\}$ to become the same edge. The rule thus tries to keep the graph as dense as possible.)

In [27], the related graph parameter of *contraction degeneracy*: the maximum over all minors G' of G of the minimum vertex degree of G' is introduced and studied. Computing the contraction degeneracy is NP-hard [27], but it can be computed in polynomial time on cographs [32].

A different lower bound rule, based on the Maximum Cardinality Search algorithm has been invented by Lucena [87]. Maximum Cardinality Search (originally invented as a chordal graph recognition algorithm by Tarjan and Yannakakis [108]) works as follows. The vertices of the graph are visited one by one. MCS starts at an arbitrary vertex, and then repeatedly visits an unvisited vertex which has the maximum number of visited neighbours. Lucena showed that when MCS visits a vertex that has at that point k visited neighbours, then the treewidth is at least k.

So, we can get a treewidth lower bound by constructing an MCS ordering of the vertices of G, and then reporting the maximum over all vertices of the number of its visited neighbours when it was visited. This bound is always at least the degeneracy (if G has a subgraph G' with minimum vertex degree k, then when the last vertex from G' is visited, it has at least k visited neighbours). A theoretical and experimental analysis of this lower bound was made by Bodlaender and Koster [24]. E.g., it is NP-hard to find an MCS ordering that maximises the yielded lower bound.

Other lower bounds based on the degree are also possible. Ramachandramurthi [94, 95] showed that for all graphs that are not complete, the minimum over all non-adjacent pairs of vertex v and w of the maximum of the degree of v and w is a lower bound for the treewidth of G. (This bound can be shown as follows. Consider a tree decomposition of G, and repeatedly remove leaf nodes i from T with neighbour j in T with $X_i \subseteq X_j$. If we remain with a tree decomposition with one bag, the claim clearly holds. Otherwise, T has at least two leaf nodes, and each bag of a leaf node contains a vertex whose neighbours are all in the same leaf bag.) This lower bound usually is not very high, but when combined with contraction, it can give small improvements to the MMD+ lower bound. An investigation of this method, and other methods combining degree based lower bounds with contraction is made in [81].

5.2 Improved graphs

An interesting technique to obtain better lower bounds was introduced by Clautiaux et al. in [39]. It uses the following result.

Lemma 4 *Let v, w be two vertices in G, and let v and w have at least $k+2$ disjoint neighbours (vertex disjoint paths between them). Then G has treewidth at most k, if and only if $G + \{v, w\}$ has treewidth at most k.*

The *neighbour* or *path improved graph* of G is the graph obtained by adding edges between all pairs of vertices with at least $k+2$ common neighbours (or vertex disjoint paths). The method of [39] now can be described as follows. Set ℓ to some lower bound on the treewidth of input graph G. Compute the (neighbour or path) improved graph G' of G (with $k = \ell$). Run some treewidth lower bound algorithm on G'. If this algorithm gives a lower bound larger than ℓ, then the treewidth of G is at least $\ell + 1$, and we add one to ℓ, and repeat, until no increase to ℓ is obtained. In [39], the degeneracy is used as lower bound subroutine, but any other lower bound can be used. Experimental results of this type can be found in [27]. The method gives significant increases to the lower bounds for many graphs, but also costs much time; the version where we use the neighbour improved graph gives smaller bounds but uses also less time when compared to the path improved graph. In [27], a heuristic is proposed, where edge contraction steps are alternated with improvement steps. This algorithm works well for small instances, but appears to use (too) much time on larger instances.

5.3 Brambles

For many instances, the lower bound methods described above give good lower bounds. However, there are also instances where these 'degree-based' techniques do not work well, e.g., for planar graphs and graphs that are similar to planar graphs. For instance, as each planar graph has a vertex of degree at most five, the contraction degeneracy of a planar graph is at most five. A different lower bound technique that appears to work well on planar graphs and graphs that are 'close to being planar' is based on the use of brambles. A bramble of a graph is a collection of mutually touching connected subsets of vertices;

two subsets are said to touch if they intersect or there is an edge between two vertices, one from each set. The order of a bramble is the smallest size of a set S of vertices such that each set in the bramble contains at least one vertex from S. Seymour and Thomas [104] have shown that the maximum order of a bramble of a graph G is exactly one larger than the treewidth of G. In [20], heuristics are given and evaluated that construct brambles in given graphs with large orders, thus giving an alternative method for treewidth lower bounds.

6 Preprocessing and postprocessing

6.1 Preprocessing

There are several methods for preprocessing a graph before running an algorithm for treewidth on it. With preprocessing, we hope to decrease the size of the input graph. The algorithm for treewidth thus often runs on a smaller instance, and hence can be much faster. E.g., we first preprocess the graph, and then run a slow exact algorithm on the reduced instance.

6.1.1 Reduction rules

Bodlaender et al. [26] give several reduction rules that are *safe* for treewidth. Besides a graph (initially the input graph), we maintain an integer variable *low* that is a lower bound for the treewidth of the input graph. We have that initially $low \leq tw(G)$, (e.g., *low*$= 0$. Each reduction rule takes G and *low*, and rewrites this to a smaller graph G', with possibly an updated value of *low*. A rule is safe, if, whenever we can rewrite a graph G with variable *low* to G' and *low*', we have $\max(tw(G), low) = \max(tw(G'), low')$. It follows that when G'' and *low*" are obtained from G with a series of applications of safe rules, then the treewidth of G equals $\max(tw(G''), low'')$. The rules in [26] are taken from the algorithm from [5] to recognise graphs of treewidth at most three, or generalisations of these. Two of these rules are the *simplicial rule*: remove a vertex of degree d whose neighbours form a clique, and set *low* to $\max(d, low)$, and the *almost simplicial rule*: when v is a vertex of degree $d \leq low$ whose neighbourhood contains a clique of size $d - 1$, then add edges between non-adjacent neighbours of v and remove v. Experiments show that in many instances from practical problems, significant reductions can be obtained with these reduction rules [26]. Generalisations of the rules were given by van den Eijkhof and Bodlaender [55].

6.1.2 Safe separators

A set of vertices $S \subseteq V$ is a *separator* in a graph $G = (V, E)$, if $G[V - S]$ contains more than one connected component. A separator is *inclusion minimal*, when it does not contain another separator of G as proper subset. A separator S in G is *safe for treewidth*, when the treewidth of G equals the maximum over all connected components W of $G[V - S]$ of the treewidth of the graph $G[W \cup S] + clique(S)$ (i.e., the graph obtained with vertices $V \cup S$, and edges between adjacent vertices in G, and each pair of vertices in S).

Thus, when we have a safe (for treewidth) separator S in G, we can split G into the parts of the form $G[W \cup S] + clique(S)$ for all connected components W of $G[V - S]$, and compute for each such part the treewidth separately. Hence, safe separators can be used for preprocessing for treewidth.

There are several types of safe separators that can be found quickly. For instance, every separator that is a clique is safe (see [91]), and clique separators can be found in $O(nm)$ time. Other safe separators are given in [25], e.g., inclusion minimal separators of size r that contain a clique of size $r-1$; all inclusion separators of size two; minimum size separators S of size three such that at least two connected components of $G[V - S]$ contain at least two vertices. See also [25] for an experimental evaluation of the use of safe separators.

6.1.3 Modular and join decompositions

A different form of preprocessing is possible by using modular or join decompositions of graphs. We say that $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are a join decomposition of $G = (V, E)$ if $V = V_1 \cup V_2$ and $E = E_1 \cup E_2 \cup \{\{v, w\} \mid v \in V_1, w \in V_2\}$, i.e., G is obtained by taking the disjoint union of G_1 and G_2 and adding all edges between a vertex in V_1 and a vertex in V_2. If G_1 and G_2 are a join decomposition of G, then $\tau(G) = \min(\tau(G_1) + |V_2|, \tau(G_2) + |V_1|)$, so we can compute the treewidth of G easily from the treewidth of G_1 and G_2. Finding a join decomposition can be done in quadratic time: look whether the complement of G has more than one connected component. In experiments, we found only a few cases where preprocessing by using the join decomposition lead to a size reduction of the graph; in most cases, this was because the graph had a universal vertex (a vertex adjacent to all other vertices); removing a universal vertex decreases the treewidth by exactly one.

A generalisation of the join decomposition is to use the modular decomposition. In [29], it is shown that the treewidth can be computed using a modular decomposition of the graph. In theory, this can be used for preprocessing, but it is unlikely that in many practical instances, this is of benefit.

6.2 Postprocessing

Once we have found a tree decomposition of G, it sometimes is possible to modify the tree decomposition slightly to obtain one with a smaller width. This can be best explained by looking at the triangulation of G that corresponds to the tree decomposition.

Many heuristics yield tree decompositions whose corresponding triangulations are not always minimal triangulations, e.g., the minimum degree heuristic. (A few heuristics guarantee that the triangulation is always minimal.)

There are several algorithms, that, given a graph $G = (V, E)$, and a triangulation $H = (V, F)$ of G, find a minimal triangulation $H' = (V, F')$ of G that is a subgraph of

H: $E \subseteq F' \subseteq F$ [11, 45, 64]. So, we can use the following postprocessing step when given a tree decomposition: build the corresponding triangulation, find a minimal triangulation (e.g., with an algorithm from [11, 64]) and then turn this minimal triangulation back into a tree decomposition.

7 Conclusions

There are several interesting notions that are related to treewidth, and that obtained also much attention in the past years, e.g., pathwidth, cliquewidth (see e.g. [44]). Very closely related to treewidth is the notion of branchwidth (treewidth and branchwidth differ approximately by at most a factor of 1.5). The branchwidth of planar graphs can be computed in polynomial time [105], and thus it is intriguing that the corresponding problem for treewidth is still open. Interesting experimental work on branchwidth has been done by Hicks [67, 68, 69, 70, 71, 66]. Cook and Seymour [41] used branch decompositions for solving the travelling salesman problem.

The many theoretic and experimental results on the treewidth problem show that finding a tree decomposition of small width is far from hopeless, even while the problem itself is NP-hard. Upper and lower bound heuristics appear to give good results in many practical cases, which can be further improved by postprocessing; preprocessing combined with cleverly designed exact algorithms can solve many small instances exactly. There still are several challenges. Two theoretical questions are open for a long time, and appear to be very hard: Is there an approximation algorithm for treewidth with a constant performance ratio (assuming $P \neq NP$)? Does there exist a polynomial time algorithm for computing the treewidth of planar graphs, or is this problem NP-hard? Also, the quest for better upper and lower bound heuristics, more effective preprocessing methods, etc. remains.

Acknowledgement

I want to express my gratitude to the many colleagues who collaborated with me on the research on treewidth and other topics, helped me with so many things, and from who I learned so much in the past years. I apologise to all whose work should have been included in this overview but was inadvertingly omitted by me.

References

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

[2] 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, 2004.

[3] E. Amir. Efficient approximations for triangulation of minimum treewidth. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 7–15, 2001.

[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 and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Alg. Disc. Meth.*, 7:305–314, 1986.

[6] E. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. Technical Report UU-CS-2004-036, Institute for Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

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

[8] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artificial Intelligence*, 125:3–17, 2001.

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

[10] D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science*, 5:33–49, 1991.

[11] J. R. S. Blair, P. Heggernes, and J. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comp. Sc.*, 250:125–141, 2001.

[12] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. In A. George, J. R. Gilbert, and J. H. U. Liu, editors, *Graph Theory and Sparse Matrix Computations*, pages 1–29, New York, 1993. Springer-Verlag.

[13] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[14] H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. *Disc. Appl. Math.*, 54:101–115, 1994.

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

[16] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In I. Privara and P. Ruzicka, editors, *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97, Lecture Notes in Computer Science, volume 1295*, pages 19–36, Berlin, 1997. Springer-Verlag.

[17] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.

[18] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos, 2005. Work in progress.

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

[20] H. L. Bodlaender, A. Grigoriev, and A. M. C. A. Koster. Treewidth lower bounds with brambles. Submitted for publication, 2005.

[21] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27:1725–1746, 1998.

[22] H. L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21:358–402, 1996.

[23] H. L. Bodlaender, T. Kloks, D. Kratsch, and H. Mueller. Treewidth and minimum fill-in on d-trapezoid graphs. *J. Graph Algorithms and Applications*, 2(5):1–23, 1998.

[24] 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.

[25] H. L. Bodlaender and A. M. C. A. Koster. Safe separators for treewidth. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments ALENEX04*, pages 70–78, 2004.

[26] H. L. Bodlaender, A. M. C. A. Koster, F. v. d. Eijkhof, and L. C. van der Gaag. Pre-processing for triangulation of probabilistic networks. In J. Breese and D. Koller, editors, *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 32–39, San Francisco, 2001. Morgan Kaufmann.

[27] 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.

[28] H. L. Bodlaender and R. H. Möhring. The pathwidth and treewidth of cographs. *SIAM J. Disc. Math.*, 6:181–188, 1993.

[29] H. L. Bodlaender and U. Rotics. Computing the treewidth and the minimum fill-in with the modular decomposition. *Algorithmica*, 36:375–408, 2003.

[30] H. L. Bodlaender and D. M. Thilikos. Treewidth for graphs with small chordality. *Disc. Appl. Math.*, 79:45–61, 1997.

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

[32] H. L. Bodlaender and T. Wolle. Contraction degeneracy on cographs. Technical Report UU-CS-2004-031, Institute for Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, 2004.

[33] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Disc. Appl. Math.*, 136:183–196, 2004.

[34] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31:212–232, 2001.

[35] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theor. Comp. Sc.*, 276:17–32, 2002.

[36] V. Bouchitté and I. Todinca. Approximating the treewidth of at-free graphs. *Disc. Appl. Math.*, 131:11–37, 2003.

[37] H. Broersma, E. Dahlhaus, and T. Kloks. A linear time algorithm for minimum fill in and tree width for distance hereditary graphs. *Disc. Appl. Math.*, 99:367–400, 2000.

[38] H. Broersma, T. Kloks, D. Kratsch, and H. Müller. A generalization of AT-free graphs and a generic algorithm for solving triangulation problems. *Algorithmica*, 32:594–610, 2002.

[39] 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.

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

[41] W. Cook and P. D. Seymour. Tour merging via branch-decomposition. *Informs J. on Computing*, 15(3):233–248, 2003.

[42] B. Courcelle. Graph rewriting: an algebraic and logical approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, volume B*, pages 192–242, Amsterdam, 1990. North Holland Publ. Comp.

[43] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In N. Robertson and P. Seymour, editors, *Graph Structure Theory, Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference, Seattle WA, June*

1991, pages 565 – 590, Providence, RI, 1993. American Math. Soc. Contemp. Math. 147.

[44] B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique width. *Theor. Comp. Sc.*, 33:125–150, 2000.

[45] E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In *Proceedings 23rd International Workshop on Graph-Theoretic Concepts in Computer Science WG'97*, pages 132–143. Springer Verlag, Lecture Notes in Computer Science, vol. 1335, 1997.

[46] E. Dahlhaus. Minimum fill-in and treewidth for graphs modularly decomposable into chordal graphs. In *Proceedings 24th International Workshop on Graph-Theoretic Concepts in Computer Science WG'98*, pages 351–358. Springer Verlag, Lecture Notes in Computer Science, vol. 1517, 1998.

[47] R. Dechter. Bucket elimination: a unifying framework for reasoning. *Acta Informatica*, 113:41–85, 1999.

[48] E. Demaine, M. Hajiaghayi, N. Nishimura, P. Ragde, and D. M. Thilikos. Approximation algorithms for classes of graphs excluding single-crossing graphs as minors. *J. Comp. Syst. Sc.*, 69:166–195, 2004.

[49] E. D. Demaine and M. Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA2005*, pages 590–601, 2005.

[50] E. D. Demaine and M. Hajiaghayi. Graphs excluding a fixed minor have grids as large as treewidth, with combinatorial and algorithmic applications through bidimensionality. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA2005*, pages 682–689, 2005.

[51] R. Diestel. *Graph Decompositions. A Study in Infinite Graph Theory.* Oxford Science Publications, Clarendon Press, Oxford, 1990.

[52] R. Diestel. *Graph theory.* Springer-Verlag, 2000. 2nd edition.

[53] R. Diestel, T. R. Jensen, K. Y. Gorbunov, and C. Thomassen. Highly connected sets and the excluded grid theorem. *J. Comb. Theory Series B*, 75:61–73, 1999.

[54] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1998.

[55] F. v. d. Eijkhof and H. L. Bodlaender. Safe reduction rules for weighted treewidth. In L. Kučera, editor, *Proceedings 28th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'02*, pages 176–185. Springer Verlag, Lecture Notes in Computer Science, vol. 2573, 2002.

[56] M. R. Fellows. The Robertson-Seymour theorems: A survey of applications. *Contemporary Mathematics*, 89:1–18, 1989.

[57] 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*, pages 568–580, 2004.

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

[59] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In proceedings UAI'04, Uncertainty in Artificial Intelligence, 2004.

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

[61] J. Gustedt, O. A. Mæhle, and J. A. Telle. The treewidth of Java programs. In D. M. Mount and C. Stein, editors, *Proceedings 4th International Workshop on Algorithm Engineering and Experiments*, pages 86–97. Springer Verlag, Lecture Notes in Computer Science, vol. 2409, 2002.

[62] M. Habib and R. H. Möhring. Treewidth of cocomparability graphs and a new order-theoretic parameter. *ORDER*, 1:47–60, 1994.

[63] P. Heggernes, J. A. Telle, and Y. Villanger. Computing minimal triangulations in time $O(n^\alpha \log n) = o(n^{2.376})$. To appear in proceedings SODA'05, 2005.

[64] P. Heggernes and Y. Villanger. Efficient implementation of a minimal triangulation algorithm. In R. Möhring and R. Raman, editors, *Proceedings of the 10th Annual European Symposium on Algorithms, ESA'2002*, pages 550–561. Springer Verlag, Lecture Notes in Computer Science, vol. 2461, 2002.

[65] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *J. SIAM*, 10:196–210, 1962.

[66] I. V. Hicks. Graphs, branchwidth, and tangles! Oh my! Working paper. http://ie.tamu.edu/People/faculty/Hicks/default.htm.

[67] I. V. Hicks. *Branch Decompositions and their Applications.* Ph. d. thesis, Rice University, Houston, Texas, 2000.

[68] I. V. Hicks. Branchwidth heuristics. *Congressus Numerantium*, 159:31–50, 2002.

[69] I. V. Hicks. Branch decompositions and minor containment. *Networks*, 43:1–9, 2004.

[70] I. V. Hicks. Planar branch decompositions I: The ratcatcher. INFORMS Journal on Computing (to appear), 2005.

[71] I. V. Hicks. Planar branch decompositions II: The cycle method. INFORMS Journal on Computing (to appear), 2005.

[72] U. Kjærulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, 2:2–17, 1992.

[73] J. Kleinberg and Éva Tardos. *Algorithm Design.* Addison-Wesley, Boston, 2005.

[74] T. Kloks. *Treewidth. Computations and Approximations*. Lecture Notes in Computer Science, Vol. 842. Springer-Verlag, Berlin, 1994.

[75] T. Kloks. Treewidth of circle graphs. *Int. J. Found. Computer Science*, 7:111–120, 1996.

[76] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. *J. Algorithms*, 19:266–281, 1995.

[77] T. Kloks, D. Kratsch, and J. Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theor. Comp. Sc.*, 175:309–335, 1997.

[78] A. M. C. A. Koster. *Frequency Assignment - Models and Algorithms.* PhD thesis, Univ. Maastricht, Maastricht, The Netherlands, 1999.

[79] 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. Elsevier Science Publishers, 2001.

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

[81] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. To appear in Proceedings WEA 2005: 4th International Workshop on Efficient and Experimental Algorithms, 2005.

[82] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20:20–44, 1996.

[83] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 532–543. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

[84] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. *Statistics and Computing (UK)*, 7(1):19–34, 1997.

[85] 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.

[86] D. R. Lick and A. T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22:1082–1096, 1970.

[87] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM J. Disc. Math.*, 16:345–353, 2003.

[88] J. Matoušek and R. Thomas. Algorithms for finding tree-decompositions of graphs. *J. Algorithms*, 12:1–22, 1991.

[89] R. H. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and Order*, pages 41–101, Dordrecht, 1985. Reidel.

[90] R. Niedermeier. *Invitation to fixed-parameter algorithms.* Universität Tübingen, 2002. Habilitation Thesis.

[91] K. G. Olesen and A. L. Madsen. Maximal prime subgraph decomposition of Bayesian networks. *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 32:21–31, 2002.

[92] A. Parra and P. Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Disc. Appl. Math.*, 79:171–188, 1997.

[93] L. Perković and B. Reed. An improved algorithm for finding tree decompositions of small width. In P. Widmayer, editor, *Proceedings 25th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'99*, pages 148–154. Springer Verlag, Lecture Notes in Computer Science, vol. 1665, 1999.

[94] 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.

[95] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Disc. Math.*, 10:146–157, 1997.

[96] B. Reed. Finding approximate separators and computing tree-width quickly. In *Proceedings of the 24th Annual Symposium on Theory of Computing*, pages 221–228, New York, 1992. ACM Press.

[97] 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.

[98] B. A. Reed. *Algorithmic aspects of tree width*, pages 85–107. CMS Books Math. / Ouvrages Math. SMC, 11. Springer, New York, 2003.

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

[100] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory Series B*, 63:65–110, 1995.

[101] H. Röhrig. Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

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

[103] D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Disc. Math.*, 9(1):101–117, 1996.

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

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

[106] 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.

[107] G. Szekeres and H. S. Wilf. An inequality for the chromatic number of a graph. *J. Comb. Theory*, 4:1–3, 1968.

[108] 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.

[109] M. Thorup. Structured programs have small tree-width and good register allocation. *Information and Computation*, 142:159–181, 1998.

[110] Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib, 2004-03-31.

[111] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11:298–313, 1982.

[112] G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization: "Eureka, you shrink"*, pages 185–207, Berlin, 2003. Springer Lecture Notes in Computer Science, vol. 2570.

[113] D. R. Wood. Queue layouts, tree-width, and three-dimensional graph drawing. In *Proc. of 22nd Foundations of Software Technology and Theoretical Computer Science FSTTCS'02*, pages 348–359. Springer Verlag, Lecture Notes in Computer Science, vol. 2556, 2002.

[114] A. Yamaguchi, K. F. Aoki, and H. Mamitsuka. Graph complexity of chemical compounds in biological pathways. *Genome Informatics*, 14:376–377, 2003.