An introduction to treewidth

D. Bronner[1], B. Ries [2]

Abstract

The notion of treewidth has seen to be a powerful vehicle for many graph algorithmic studies. A lot of problems, which are in general NP-complete, can be solved in polynomial time for graphs with small treewidth. In the first part of this paper we present three different definitions of treewidth and we show their equivalence. The second part consists of an overview of the complexity status of the treewidth problem as well as of algorithmics results. We also discuss a polynomial time algorithm for computing the treewidth of circle graphs.

Keywords

treewidth, chordal graphs, triangulation, k-trees, circle graphs.

(March 10, 2006)

[1]IMA - EPFL, Lausanne (Switzerland), email: daniela.bronner@epfl.ch
[2]IMA - EPFL, Lausanne (Switzerland), email: bernard.ries@epfl.ch

1 Introduction

The notion of treewidth is playing a central role in many recent investigations in algorithmic graph theory, mainly because many classical NP-hard problems become polynomial and even linear when restricted to graphs with small treewidth.

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 actual applied settings. Amongst other interesting examples of this practical use of treewidth, we can say that it is used in expert system [42], telecommunication network design [26], VLSI-design, Choleski factorisation and natural language processing [39].

The treewidth of a graph is, in general, NP-hard to compute. Much research has been carried out to determine the treewidth for special classes of graphs (see e.g. [19], [9], [37], [38], [31]). There exist many heuristics as well as approximation algorithms for the treewidth of a graph. Moreover, for fixed k, linear time algorithms to determine whether a given graph has treewidth k have been developped.

In this paper, we present some known results related to the notion of treewidth. In section 2 some basic definitions are given. In section 3 we focus on the definition of treewidth and we prove the equivalence of three different definitions that we have chosen. Section 4 deals with some complexity results concerning the determination of the treewidth of a graph and we give an overview of existing heuristics, approximation algorithms and exact algorithms. We also consider the case of fixed parameter and introduce the notions of preprocessing and postprocessing. Finally in section 5 we present a polynomial time algorithm to determine the treewidth of circle graphs.

2 Preliminaries

Throughout this paper we consider simple, finite, undirected graphs. We shall start by giving some definitions which will be used frequently. For all graph theoretical terms not defined here, the reader is referred to [4].

The class of undirected graphs known as **k-trees** is defined recursively as follows: A k-tree on k vertices consists of a clique on k vertices (k-clique); given any k-tree T_n on n vertices, we construct a k-tree on $n+1$ vertices by adding a new vertex v_{n+1} to T_n, which is made adjacent to each vertex of some k-clique of T_n and nonadjacent to the remaining $n-k$ vertices. A graph G is called a **partial k-tree** if G is a subgraph of a k-tree.

A graph G is a **chordal graph** (or **triangulated graph**) if every cycle of length greater or equal to 4 has a chord, that is, an edge connecting two vertices that are not consecutive on the cycle.

A vertex v is called **simplicial** if its neighborhood $N(v) = \{w \in V | [v, w] \in E\}$ is a clique, that is, every pair of neighbors of v are connected by an edge of the graph.

Let $\sigma = [v_1, ..., v_n]$ be an ordering of the vertices and let G_i denote the subgraph remaining after deleting $\{v_1, ..., v_{i-1}\}$ from G. We define σ to be a **perfect elimination ordering (peo)** if v_i is a simplicial vertex in the graph G_i, for all i.

Given a graph $G = (V, E)$, a **triangulation** of G is a chordal graph $H = (V, F)$, with $E \subseteq F$. We will say that G can be triangulated into H.

We will denote by $\omega(G)$ the size of a largest clique in a graph G. It will be called the **clique number**.

3 Treewidth

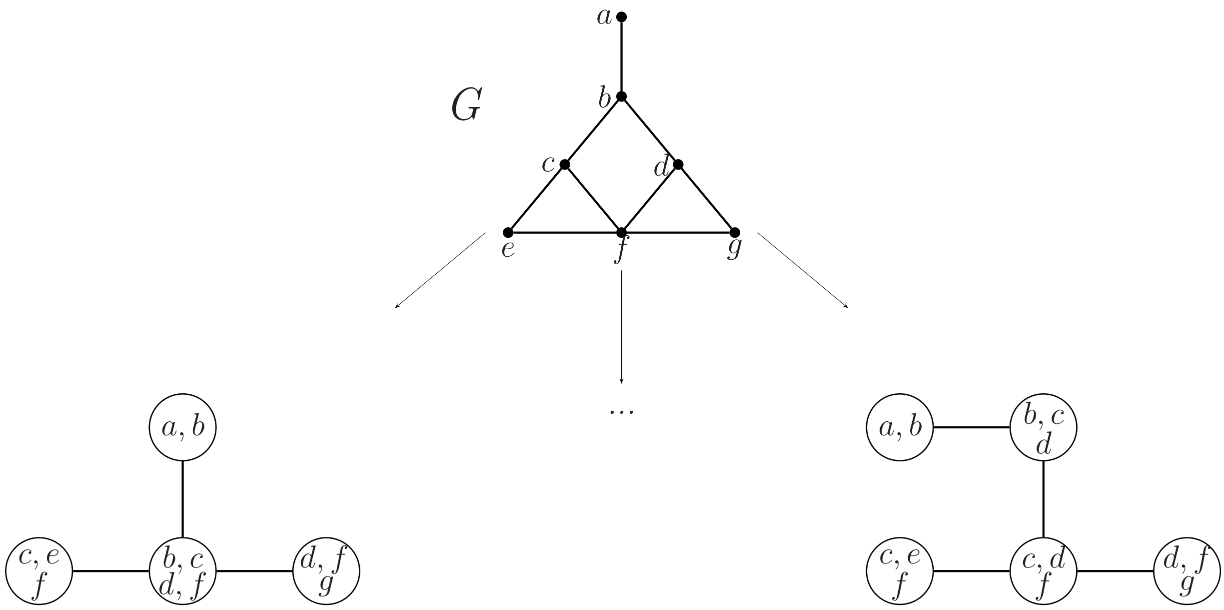
The notion of *treewidth* was introduced by Robertson and Seymour in their work on graph minors [47]. Today there exist a lot of different definitions of treewidth. For an overview of graph theoretic notions equivalent to treewidth, the reader is referred to [15].

In this section we choose three different definitions of treewidth and we show their equivalences. We shall start with the definition given by Robertson and Seymour. It is based on *tree decomposition*.

Definition 3.1 *A **tree decomposition** of a graph $G = (V, E)$ is a tree $T = (I, F)$ where each vertex $i \in I$ has a label $X_i \subseteq V$ such that:*

1. $\bigcup_{i \in I} X_i = V$;
2. *for any edge $[v, w]$ in G, there exists an $i \in I$ with $v, w \in X_i$;*
3. *for any $v \in V$, the vertices containing v in their label form a connected subtree of T.*

Example 3.1 *A graph G and two possible tree decompositions.*

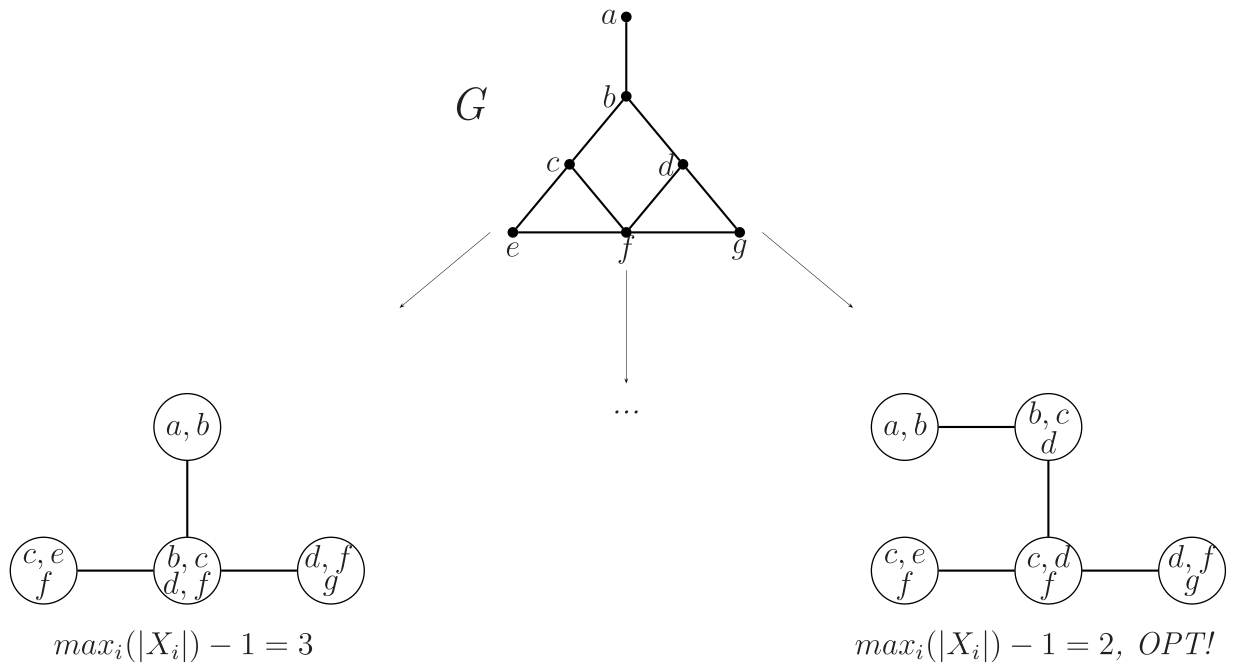


Definition 3.2 *Given a tree decomposition $T = (I, F)$, the **width** of the decomposition is* $\max_{i \in I}(|X_i|) - 1$.

Using tree decomposition, we get our first definition of treewidth.

Definition 3.3 *The **treewidth** of a graph G is the minimum k such that G has a tree decomposition of width k.*

Example 3.2 *A graph G and an 'optimal' tree decomposition.*



Remark 3.1 *Let $G = (V, E)$ be a graph and $H = (V, F)$ a triangulation of G. Then $tw(G) \leq tw(H)$ since every tree decomposition of H is also a tree decomposition of G.*

In the following theorem we state the equivalences mentioned above.

Theorem 3.1 *Let $G = (V, E)$ be an undirected graph. The following properties are equivalent:*

(i) G has treewidth k;

(ii) G is a partial k-tree, k minimal;

(iii) $\min_H(\omega(H) - 1) = k$, where H is a triangulation of G.

The rest of this section will be devoted to the proof of this main theorem. We will need several results related to chordal graphs and tree decompostion. We shall start by introducing the *Helly property.*

Definition 3.4 *[30] A family $\{T_i\}_{i \in I}$ of subsets of a set T is said to satisfy the* ***Helly property*** *if $J \subseteq I$ and $T_i \cap T_j \neq \emptyset$ for all $i, j \in J$ implies that $\bigcap_{j \in J} T_j \neq \emptyset$.*

Theorem 3.2 *[30] A family $\{T_i\}_{i \in I}$ of subtrees of a tree T satisfies the Helly property.*

Proof: Suppose that $T_i \cap T_j \neq \emptyset$ for all $i, j \in J$. Consider three vertices a, b, c in T. Let $S = \{s|\ a, b \in T_s\ or\ b, c \in T_s\ or\ a, c \in T_s\}$ be the set of indices s such that T_s contains at least two of these three vertices. Denote by P_{ab}, P_{bc}, P_{ac} the simple paths in T connecting a with b, b with c and a with c, respectively. Notice that since T is a tree, $P_{ab} \cap P_{bc} \cap P_{ac} \neq \emptyset$. Furthermore each T_s, $s \in S$, contains at least one of these paths. Therefore, $\bigcap_{s \in S} T_s \supseteq P_{ab} \cap P_{bc} \cap P_{ac} \neq \emptyset$.
We will prove the result by induction. Let us assume that $[T_i \cap T_j \neq \emptyset$ for all $i, j \in J] \rightarrow \bigcap_{j \in J} T_j \neq \emptyset$ for all index sets J of size $\leq k$. Notice that this is certainly true for $k = 2$. We consider now a family of subtrees $\{T_{i_1}, ..., T_{i_{k+1}}\}$. By induction hypothesis, there exist three vertices a, b, c in T such that $a \in \bigcap_{j=1}^{k} T_{i_j}$, $b \in \bigcap_{j=2}^{k+1} T_{i_j}$, $c \in T_{i_1} \cap T_{i_{k+1}}$. Moreover, every T_{i_j} contains at least two of the vertices a, b, c. Hence, using the facts mentioned above, we conclude that $\bigcap_{j=1}^{k+1} T_{i_j} \neq \emptyset$. □

Lemma 3.1 *[18] Let $G = (V, E)$ be a graph and let $T = (I, F)$ be a tree decomposition where each $i \in I$ has a label $X_i \subseteq V$. Let $W \subseteq V$ be a clique in G. Then $\exists\ i \in I$ such that $W \subseteq X_i$.*

Proof: Let $T_v = \{i \in I|\ v \in X_i\}$. $\{T_v|\ v \in W\}$ is a family of subtrees of a tree. By the previous theorem we have that $\exists\ i \in I$ such that $\forall v \in W$, $i \in T_v$. Hence $\exists\ i \in I$ such that $W \subseteq X_i$. □

The following corollary is an immediate consequence of the previous lemma:

Corollary 3.1 *Let $G = (V, E)$ be a graph. Then $tw(G) \geq \omega(G) - 1$.*

The next theorem is due to Dirac and will be given without any proof.

Theorem 3.3 *G is a chordal graph if and only if G has a perfect elimination ordering.*

Lemma 3.2 *[36] If G is a chordal graph with at least $k+1$ vertices and has a clique number at most $k+1$, then G can be triangulated into a k-tree.*

Proof: Let $\sigma = [v_1, ..., v_n]$ be a perfect elimination ordering of G (σ exists because of Theorem 3.3). We make an embedding of G in a k-tree recursively as follows. First we add edges such that the subgraph induced by $\{v_{n-k}, ..., v_n\}$ becomes a $k+1$-clique. For the induction step, assume the subgraph with vertices $\{v_{i+1}, ..., v_n\}$ has been triangulated into a k-tree T_{i+1}. In G vertex v_i is adjacent to a clique C with at most k vertices in $\{v_{i+1}, ..., v_n\}$. In the k-tree T_{i+1}, C is contained in a k-clique C'. We make v_i adjacent to all vertices of C' and we obtain a k-tree T_i which is a triangulation of G. □

Theorem 3.4 *[5] The following statements are equivalent:*

(i) G is a partial k-tree.

(ii) G is a subgraph of a chordal graph G' with $\omega(G') \leq k+1$.

(iii) G is a subgraph of a chordal graph G' with $\omega(G') = k+1$.

Proof: $i) \Rightarrow ii)$ follows immediately from the definition of a partial k-tree, since a k-tree is chordal and the maximum clique has size at most $k+1$.
$ii) \Rightarrow iii)$ is trivial, for if G' has $\omega(G') < k+1$, then we add edges to G' until the maximum clique has size $k+1$.
$iii) \Rightarrow i)$ holds because of Lemma 3.2, since we can add edges to G' to obtain a k-tree, thus any subgraph of G' is a partial k-tree. □

Lemma 3.3 *[5] Let G be a chordal graph. Then it has a tree decompostion of width $\omega(G)-1$.*

Proof: Let $v_n, ..., v_1$ be a perfect elimination ordering of G. By induction on i, we will build a tree decompostion T_i for G such that for all $j \leq i$ there exists a vertex in T_i whose label contains all vertices in $\{v_j\} \cup Pred(v_j)$, where $Pred(v_j)$ is the set of neigbours v_l of v_j with $l \leq j$.
Let $k = \omega(G) - 1$. The base case occurs for $i = k+1$. We add all vertices $v_1, ...v_{k+1}$ into one vertex of the tree decomposition. Clearly this satisfies all conditions.
Now we suppose that $i > k+1$. Let T_{i-1} be a tree decomposition for the graph induced by the vertices $v_1, ..., v_{i-1}$ such that for all $j \leq i-1$ there exists a vertex in T_{i-1} whose label contains all vertices in $\{v_j\} \cup Pred(v_j)$. Consider vertex v_i. Let v_h be the vertex such that $h = \max_{v_j \in Pred(v_i)}(j)$.
By induction we know that there exists a vertex $a \in T_{i-1}$ that contains all vertices in $\{v_h\} \cup Pred(v_h)$. Add a new vertex b to the tree decompostion T_{i-1} and make it adjacent to a. Give it label $\{v_i\} \cup Pred(v_i)$. We obtain the tree decomposition T_i for the graph induced by the vertices $v_1, ..., v_i$.
The label size of vertex b is $1 + |Pred(v_i)| \leq \omega(G)$. Thus T_i has width at most $k = \omega(G) - 1$. Clearly it covers all vertices and edges and for all $j \leq i$ there exists a vertex in T_i whose label contains all vertices in $\{v_j\} \cup Pred(v_j)$. Let us now verify that the connectivity condition still holds. Since v_h is the vertex such that $h = \max_{v_j \in Pred(v_i)}(j)$, we have that $Pred(v_i) \subseteq Pred(v_h) \cup \{v_h\}$. Therefore, the only vertices for which we added a new vertex containing them are v_i (for which b is the only vertex containing it, so this is connected), or a vertex in $Pred(v_i)$, for which a also contains it, so its subtree continues to be connected. □

From Corollary 3.1 and Lemma 3.3 we obtain the following result:

Corollary 3.2 *Let $G = (V, E)$ be a chordal graph. Then $tw(G) = \omega(G) - 1$.*

The following three propositions shall proof theorem 3.1 that we stated at the beginning of this section.

Proposition 3.1 *[32] Let $G = (V, E)$ be a graph with treewidth k. Then G is partial k-tree.((i) $\Rightarrow$ (ii))*

Proof: Let $G = (V, E)$ be a graph with $|V| \geq k + 1$, and let $T = (I, F)$ be a tree decomposition of G of width at most k. We will prove, with induction on $|V|$, that there is a k-tree $H = (V, E')$ such that for every $i \in I$, $G[X_i]$, representing the subgraph of G induced by X_i, is a subgraph of a clique of H with $k + 1$ vertices. If $|V| = k + 1$ then we are done. Otherwise, take a leaf vertex $l \in I$ and let $j \in I$ be the only neighbor of $l \in T$. If $X_l \subseteq X_j$, then we can remove l from T and continue with the remaining tree decomposition. Let v be the only vertex in $X_l \setminus X_j$ (otherwise X_l can be divided into several tree vertices such that the resulting new leaf vertex satisfies this requirement). Now it is important to note the following things: since v does not appear in any X_i for $i \neq l$, all neighbors of v must appear in X_l. Since $|X_l| \leq k + 1$, v has at most k neighbors. Also, because v is the only vertex in $X_l \setminus X_j$, all neighbors of v must also appear in X_j. Suppose that the induction hypothesis on $G[V \setminus \{v\}]$ with tree decomposition $T = (I \setminus \{l\}, F')$ results in a k-tree H'. Remember that v has at most k neighbors in G and they all belong to X_j. Therefore, the neighbors of v induce a subgraph of a k-clique C of H' in G. Now, we can add v with edges to all vertices of C to H' (which will make a $(k + 1)$-clique), and get the desired k-tree H. □

Proposition 3.2 *Let $G = (V, E)$ be a partial k-tree with k minimal. Then $\min_H(\omega(H) - 1) = k$, where H is a triangulation of G.((ii) $\Rightarrow$ (iii))*

Proof: Suppose G is a partial k-tree with k minimal. By Theorem 3.4, we know that G is a subgraph of a chordal graph G' with $\omega(G') = k + 1$. Thus there is a triangulation G' of G with $\omega(G') = k + 1$. Let us show now that there is no triangulation H of G with $\omega(H) < k + 1$. Suppose that there is such a triangulation H with $\omega(H) = k'' < k + 1$. As H is chordal and $\omega(H) = k''$, we can add edges to H such that the resulting graph is a $(k''-1)$-tree (see Lemma 3.2). Then G would be a partial $(k''-1)$-tree, with $k''-1 < k$. This is in contradiction with our hypothesis that k is minimal. Thus there is no triangulation H of G with $\omega(H) < k + 1$ and so $\min_H(\omega(H) - 1) = k$ over all triangulations H of G. □

Proposition 3.3 *Let $G = (V, E)$ be a graph and $\min_H(\omega(H) - 1) = k$, where H is a triangulation of G. Then G has treewidth k. ((iii) $\Rightarrow$ (i))*

Proof: Let H be the triangulation of G with $\omega(H) = k + 1$. As H is chordal, $tw(H) = \omega(H) - 1 = k$ (see Corollary 3.2). Furthermore, since G is a partial graph of H, $tw(G) \leq tw(H) = k$ (see Remark 3.1). Now suppose that $tw(G) < k$, that is, $tw(G) = k' < k$. Then G is a partial k'-tree (Proposition 3.1), k' minimal, and thus $\min_H(\omega(H) - 1) = k' < k$, where H is a triangulation of G. This is a contradiction. We conclude that G has treewidth k. □

4 Complexity results

In this section we give a overview of the complexity status of the following problem:

Treewidth Problem
Instance: An undirected graph $G = (V, E)$ and an integer k.
Question: Is the treewidth of G at most k?

This problem was proven to be NP-complete in [1]. As with a lot of NP-complete problems there exist several special classes of graphs for which the treewidth can be computed in polynomial time. The following table gives an overview of the complexity status for some classes of graphs (see e.g. [19], [9], [37], [38], [31]). Notice that for planar graphs, the complexity status is still open.

Class	**Complexity**
Tree/Forest	Const.
Series-parallel graph	Const.
Outerplanar graph	Const.
k-Outerplanar graph	Const.
Halin graph	Const.
Chordal graph	Poly.
Starlike chordal graph	Poly.
k-Starlike chordal graph	Poly.
Co-chordal graph	Poly.
Split graph	Poly.
Permutation graph	Poly.
Circular permutation graph	Poly.
Cograph	Poly.
Chordal bipartite graph	Poly.
Interval graph	Poly.
Circular arc graph	Poly.
Circle graph	Poly.
Distance hereditary graph	Poly.
Bounded degree	NP-comp.
Bipartite graph	NP-comp.
Cocomparability graph	NP-comp.
Planar graphs	open

Table 1: *Complexity status for some classes of graphs.*

Recently Bouchitté and Todinca (see [21]) found an algorithm to determine the treewidth of a graph in time, polynomial in the number of its minimal separators. Many graph classes have the property that each graph in the class has a polynomial number of minimal separators, for instance permutation graphs or weakly chordal graphs.

In the following we give an overview of existing algorithms for computing the treewidth of a graph. We will distinguish between exponential time algorithms, approximation algorithms, upper and lower bound heuristics. We will also study the case where k is a constant and introduce the notions of *preprocessing* and *postprocessing*. This overview is based on [8] where more details and references may be found. Also for non defined notions here, the reader is referred to [8] or to the indicated references.

4.1 Exponential time algorithms

In [28] an exact algorithm for treewidth is given that runs in $O^*(1.9601^n)$ time. For small graphs, the treewidth can be computed in practice using branch and bound (see [29]). This algorithm can also be used as a heuristic, by stopping it at a specific time and reporting the best solution found so far.

4.2 Approximation algorithms

There exist many different algorithms approximating the treewidth of a graph. We start by considering algorithms which are polynomial, even when k is not bounded. The first such approximation algorithm for treewidth was given in [17]. It gives tree decompositions with width at most $O(log(n))$ times the optimal treewidth. This result was recently improved in [20], where a polynomial time approximation algorithm with ratio $O(log(k))$ was given, i.e., the output is a tree decomposition of width $O(klog(k))$ when the treewidth of the input graph is k. It is still an open problem whether there exists a polynomial time approximation algorithm for treewidth with a constant performance ratio.

There exist also some approximation algorithms for treewidth which run in time, exponential in k. They either give a tree decomposition of width at most ck (where c is a constant), or they tell that the treewidth is more than k.

Approximation algorithms have also been invented for special classes of graphs: a ratio of 2 can be obtained for AT-free graphs (see [22]) and a constant ratio can be obtained for graphs with bounded asteroidal number (see [20]).

4.3 Upper bound heuristics

Many heuristics have been invented to find tree decompositions of small width. They do not have a guarantee on their performance, but many among them perform very well. A large class of these heuristics is based upon the same principle: first an ordering of the vertices is built according to some criterion; then a fill-in procedure is used to triangulate the graph and finally a corresponding tree decomposition can be obtained. Depending on the criterion used to build the ordering one gets different heuristics, e.g. the minimum degree heuristic (the vertex with the minimum number of unselected neighbors is selected as the next vertex in the ordering) or the minimum fill-in heuristic (the vertex which gives the minimum number of added edges is selected as the next vertex in the ordering). More complicated rules to select the next vertex in the ordering are given in [3], [24] and [25]. Sometimes orderings generated by algorithms which recognize chordal graphs were used. For some instances these orderings give better bounds.

A different type of heuristic was proposed in [40]. The idea is to start with any tree decomposition and then to select a vertex of that tree decomposion and to split it into several vertices, maintaining the properties of tree decomposition.

Several algorithms which find a minimal triangulation of a given graph can be used as treewidth heuristics. In [33] an algorithm which runs in $o(n^{2.376})$ time is presented.

In [35], [25] and [41] local search methods like simulated annealing, tabu search and genetic algorithms are used to solve the treewidth problem. The running time of this meta-heuristics is higher, but they often give good results.

4.4 Lower bound heuristics

Lower bounds on the treewidth may be interesting, e.g. to inform about the quality of upper bounds. Obviously the minimum degree of a vertex in a graph G and the clique number of G are lower bounds for the treewidth. But very often these bounds are not good and furthermore the clique number is NP-hard to compute. These bounds can be improved using the *degeneracy* (maximum over all subgraphs G' of G of the minimum vertex degree) which can easily be computed (see [3], [29]).

In [43] *Maximum Cardinality Search (MCS)* is used to get a different lower bound. It is shown that when MCS visits a vertex that has at that point k visited neighbors, then the treewidth is at least k. It is easy to see that this bound is at least the degeneracy.

Other lower bounds were found based on the degree. In [46] it was shown that for all graphs that are not complete, the minimum over all non-adjacent pairs of vertices v and w of the maximum of the degree of v and w is a lower bound for the treewidth of G.

An interesting technique to obtain better lower bounds was introduced in [24]. It uses the *path improved graph* of G, which is the graph obtained by adding edges between all pairs of vertices with at least $k+2$ common neighbors. The method gives significant increases of the lower bounds for many graphs, but also needs much time. In [11] a heuristic is proposed which uses edge contraction and the path improvement graph alternatively. It works well on small instances but seems to need too much time on larger instances.

Unfortunately these degree-based heuristics seem not to work very well on some graphs like planar graphs or graphs similar to planar graphs. A different technique which seems to work well on these graphs is based on *brambles* (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). In [49] it is shown that the maximum order of a bramble (smallest size of a set S of vertices such that each set in the bramble contains at least one vertex from S) of a graph G is exactly one larger than the treewidth of G. In [7], heuristics are given using brambles.

4.5 Preprocessing

Before running an algorithm for computing the treewidth of a graph, one may try to reduce the size of the input graph in order to fasten the running time of the algorithm. With *preprocessing*, we hope to decrease the size of the graph and thus the algorithm runs on a smaller instance and can be much faster. For example, one may preprocess a graph and then run an exact algorithm on the reduced instance.

There exist several methods of preprocessing. In [14], *reduction rules* are given which allow to decrease the input graph. It is shown that if we take the maximum of the treewidth of a smaller graph (obtained by using the reduction rules) and an integer variable which is updated at each reduction step, we get the treewidth of the original graph. Generalizations of the rules are given in [27].

Another method of preprocessing is based on *separators* (S is a separator of a connected graph G if $G - S$ contains more than one connected component). Using some special separators, the input graph is split into several parts and the treewidth is computed separately for each part. It can be shown that the treewidth of the input graph G equals the maximum of the treewidth of these different parts. This method is used in [10].

Modular or *join decomposition* of a graph are other preprocessing methods (see [12]). G_1 and G_2 are a join decomposition of G if G is obtained by taking the disjoint union of G_1 and G_2 and adding all edges between a vertex in G_1 and a vertex in G_2. Modular decomposition is a generalization of join decomposition and it can be shown that the treewidth can be computed using a modular decomposition. Unfortunately, it is unlikely that in many practical instances, this is of benefit.

4.6 Postprocessing

Suppose now that our algorithm gave us a tree decomposition of the input graph. It might be possible to modify this tree decomposition such that we obtain another one with smaller width. This is called *postprocessing*. In fact, many heuristics give a tree decompostion of the input graph G whose corresponding triangulation H is not always minimal. Thus, we can try to find a minimal triangulation H' (which is a subgraph of H) and then turn this minimal triangulation back into a tree decomposition with smaller width. Algorithms to find such minimal triangulations are presented in [6] and [34].

4.7 Fixed parameter cases

We will consider now the case where k is a constant. Our problem then becomes polynomially solvable. The first polynomial time algorithm for this problem was given in [1] and it runs in $O(n^{k+2})$. The first result showing that the treewidth problem is solvable in $O(n^c)$ time for some constant c, for fixed k, was obtained by Robertson and Seymour. Later results improved their method and finally in [16] a linear time algorithm was given that checks whether the treewidth is at most k, and if so, outputs the corresponding tree decompostion. It uses $O(k^3)$ calls to a dynamic programming algorithm, but the constant hidden in the O-notation is very big, even for small values of k, e.g. $k = 4$. A variant using $O(k^2)$ calls to that same dynamic programming algorithm was given in [45].

For very small values of k, there exist special algorithms.

(a) Testing if the treewidth is one is trivially linear since the graph must be a forest.

(b) A graph has treewidth at most two if and only if each biconnected component is a series parallel graph (see [13]); testing whether a graph is series parallel can be done in linear time (see [50]).

(c) In [2] a set of six reduction rules are given, 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. The rules can be implemented such that the algorithm runs in linear time (see [44]).

(d) A more complicated linear time algorithm is given in [48] for testing if a graph has treewidth at most 4.

5 Treewidth of circle graphs

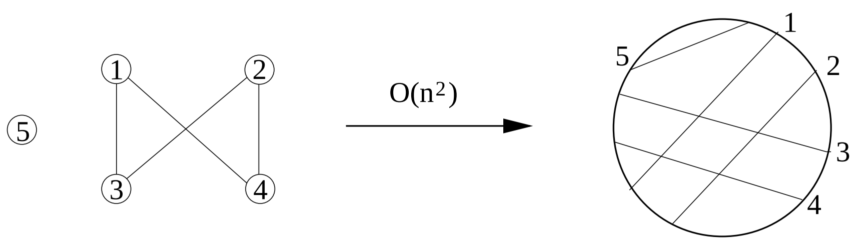
In this section we focus on a special class of graphs which are the circle graphs. We present a polynomial time algorithm to determine the treewidth of such graphs, which was given by T. Kloks in [37]. The idea of the algorithm is as follows: consider the circle model of the graph. Go around the circle in clockwise order and place a new vertex between every two consecutive end vertices of chords. Let Z be the set of these new vertices. Consider the

polygon P with vertex set Z, and let T be a triangulation of this polygon. For a triangle in this triangulation, define the weight as the number of chords in the circle model that cross the triangle. The weight of T is the maximum weight of the triangles. The treewidth of the circle graph is the minimum weight minus 1 over all triangulations of that polygon P. Using dynamic programming, the treewidth can be computed in $O(n^3)$ time.
We shall start by giving some definitions and results needed to justify the algorithm mentioned above.

Definition 5.1 *A **circle graph** is a graph for which one can associate for each vertex a chord of a circle such that two vertices are adjacent if and only if the corresponding chords have a non empty intersection. The set of chords of a circle is called the **circle model**.*

Notice that we can assume w.l.o.g. that no two chords share an end vertex. If the circle model is not given, it can be found in $O(n^2)$ time [23].

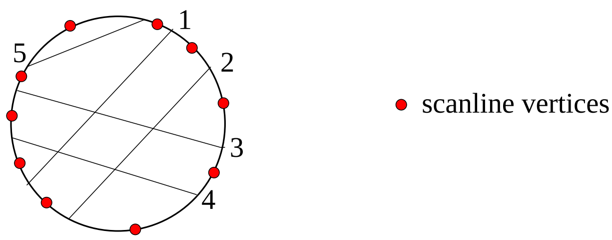
Example 5.1 *A circle graph G and the corresponding circle model.*



Definition 5.2 *Consider a circle model. Place new points on the circle as follows: go around the circle in clockwise order and place a new vertex between every 2 consecutive end vertices of chords. These new vertices are called **scanline vertices**.*

Notice that there are $2n$ scanline vertices, where n is the number of chords in the circle model. We denote the set of scanline vertices by Z and the polygon of vertex set Z by $P(Z)$.

Example 5.2 *A circle model and the scanline vertices.*



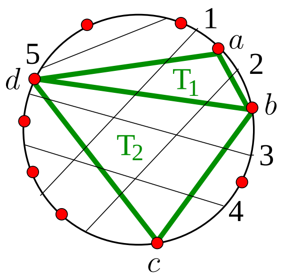
Definition 5.3 *A* ***scanline*** *is a chord of the circle of which the end vertices are scanline vertices.*

Let us give now some definitions concerning triangulations of a polygon.

Definition 5.4 *Let P be a polygon with m vertices. A* ***triangulation*** *of P is a set of $m-3$ non crossing diagonals in P that divide the interior of P in $m-2$ triangles.*

Definition 5.5 *Let Y be a set of at least three scanline vertices. Consider a triangulation T of $P(Y)$. The* ***weight*** *of a triangle in T is the number of chords in the circle model that have a non empty intersection with the triangle. The* ***weight of the triangulation*** *T, denoted by $w(T)$, is the maximum weight of a triangle.*

Example 5.3 *A circle model and the weight of two triangles T_1, T_2.*



$\omega(T_1) = 2$
$\omega(T_2) = 4$ $\Rightarrow$ $\omega(T) = 4$ with $Y = \{a, b, c, d\}$

We will now see the relation between a triangulation T of the polygon $P(Z)$ and a triangulation H of the circle graph $G = (V, E)$. We define the graph $H(T) = (V, F)$ with the same vertex set as G as follows: two vertices are adjacent in $H(T)$ if there is a triangle in T such that the chords corresponding with these two vertices intersect the triangle. Notice that G is a partial graph of $H(T)$. Then we have the following result:

Proposition 5.1 *The graph $H(T) = (V, F)$ is a triangulation of $G = (V, E)$.*

Proof: Since we already noticed that G is a partial graph of $H(T)$, we just need to prove that $H(T)$ is chordal. To do so, we will show that $H(T)$ has a perfect elimination ordering and using Theorem 3.3 we can conclude.
Consider a vertex z of $P(Z)$ which is not incident to a diagonal of T. This vertex z is incident to exactly one triangle Q. Consider a vertex x in $H(T)$ of which the corresponding chord intersects the triangle Q but no other triangle of T. Then the neighborhood of x, denoted by $N(x)$, is a clique and hence x is a simplicial vertex. Remove x from $H(T)$ and the corresponding chord from the circle model. If there is no chord left in the circle model which intersects Q but no other triangle, then remove z from Z. Repeating this process will give us a perfect elimination ordering and hence $H(T)$ is a chordal graph. □

Remark 5.1 *Notice that the number of vertices in a clique $\{x\} \cup N(x)$ of $H(T)$ is equal to the weight of the triangle Q which the corresponding chords intersect. Thus we conclude that the number of vertices in a maximum clique in $H(T)$ is equal to the weight of the triangulation T. This justifies the fact that in the algorithm which we will present, we determine a triangulation of $P(Z)$ with minimum weight. This will give us a triangulation $H(T)$ of G with a minimum clique number. By Theorem 3.1 we know that this clique number minus one is equal to the treewidth of the graph G.*

We will now present the algorithm which determines the treewidth of a circle graph in $O(n^3)$ time. First let us define some notions which will be used in the algorithm. We denote by $s_0, ..., s_{l-1}$ the scanline vertices ordered clockwise and by $c(i, j)$ the number of chords that cross the scanline $[s_i, s_j]$. $P(i, t)$ represents the polygon defined by $s_i, ..., s_{i+t-1}$, where indices are to be taken modulo l. We define $w(i, t)$ as the minimum weight of a triangulation of the polygon $P(i, t)$. Finally $F(i, j) = \frac{c(i,i+j-1)+c(i+j-1,i+t-1)+c(i,i+t-1)}{2}$ represents the number of chords crossing the triangle $(s_i, s_{i+j-1}, s_{i+t-1})$.

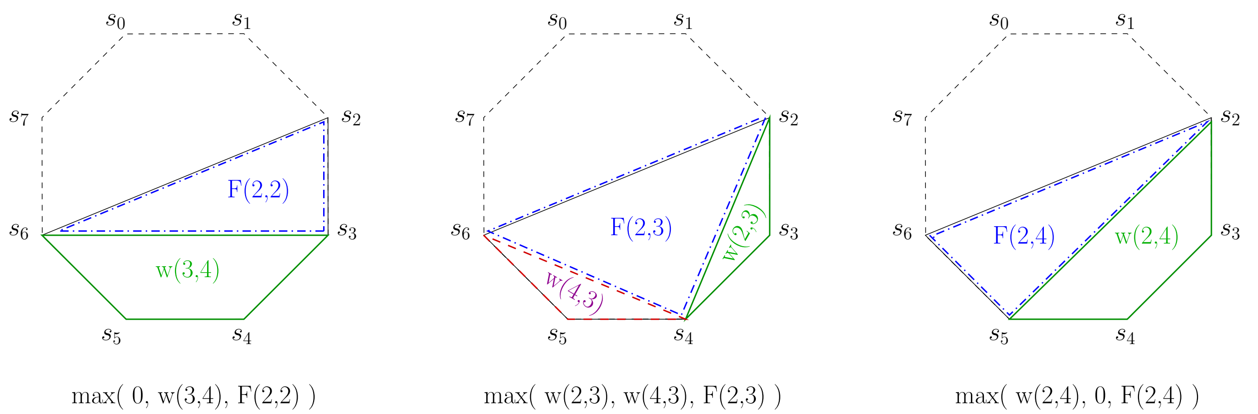
Algorithm 1 Finds the treewidth of a circle graph $G = (V, E)$

Require: Let $G = (V, E)$ be a circle graph
  compute a circle model
  determine the set Z of scanline vertices and define the polygon $P(Z)$
  for each scanline $[s_i, s_j]$ **do**
    compute $c(i, j)$
  end for
  set all $w(i, 2)$ equal to 0
  for $t = 3, ..., l$ and for all i **do**
    $w(i, t) = \min_{2 \leq j < t}(\max(w(i, j), w(i + j - 1, t - j + 1), F(i, j)))$
  end for
  return $tw(G) = w(0, l) - 1$

Since $w(0, l)$ is the minimum weight of a triangulation of the polygon $P(0, l) = P(Z)$, we clearly have that $w(0, l) - 1 = tw(G)$ (see Remark 5.1). The algorithm needs $O(n^3)$ time. In fact, as already mentioned at the beginning of this section, a circle model, given the circle graph G, can be computed in $O(n^2)$ time. Computing $c(i, j)$ costs $O(n^3)$ time since there are $O(n^2)$ scanlines and the test if a scanline and a chord cross can be performed in $O(1)$ time. Finally observe that the $w(i, t)$'s can be calculated in $O(n^3)$ time since there are $O(n^2)$ $w(i, t)$'s and at each step we need to find the minimum among at most $O(n)$ values. So we conclude that the algorithms gives us the treewidth of a circle graph $G = (V, E)$ in $O(n^3)$ time.

We will now illustrate the algorithm by an example which will "show" the correctness of the formula of $w(i, t)$.

Example 5.4 *Computing the weight $w(2, 5)$.*



Remark 5.2 *Since the class of permutation graphs is properly contained in the class of circle graphs, this result generalizes some results in [19] where an $O(nk)$ time algorithm is given for the treewidth of permutation graphs.*

6 Conclusion

In this paper we studied the notion of treewidth, which seems to be a graph parameter with several interesting theoretical and practical applications. We gave three different definitions of treewidth and showed their equivalence. More equivalent definitions of treewidth can be found in [15]. We also presented an overview of the complexity status as well as of algorithmics results related to the determination of the treewidth of a graph. In [8], Bodlaender gives a complete survey of recent algorithmic results concerning treewidth. Finally we discussed a polynomial time algorithm for circle graphs which was given in [37].

The notion of treewidth has been studied a lot, but two interesting questions remain still open: Is there an approximation algorithm for treewidth with a constant performance ratio? Does there exist a polynomial time algorithm for computing the treewidth of planar graphs, or is this problem NP-hard?

References

[1] S. Arnborg, D. Corneil and A. Prokurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Alg. Disc. Meth.* **8, No. 2** (1987) 277–284.

[2] S. Arnborg and A. Proskurowski, Characterization and recognition of partial 3-trees, *SIAM J. Alg. Disc. Meth.* **7, No. 2** (1986) 305–314.

[3] E. Bachoore and H.L. Bodlaender, *New upper bound heuristics for treewidth*, (Technical Report, Utrecht University, 2004)

[4] C. Berge, *Graphs and Hypergraphs* (North-Holland, Amsterdam, 1976).

[5] T. Biedl, Graph-Theoretic Algorithms, *Lecture Notes, University of Waterloo, Canada* (2005).

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

[7] H.L. Bodlaender, A. Grigoriev and A.M.C.A. Koster, *Treewidth lower bounds with brambles* , (Technical Report, Utrecht University, 2005)

[8] H.L. Bodlaender, *Discovering treewidth*, (Technical Report, Utrecht University, 2005)

[9] H.L. Bodlaender and T. Kloks, *Only few graphs have bounded treewidth*, (Technical Report, Utrecht University, 1992)

[10] H.L. Bodlaender and A.M.C.A. Koster, Safe separators for treewidth, *Proceedings 6th Workshop on Algorithm Engeneering and Experiments ALENEX04* (2004) 70–78.

[11] H.L. Bodlaender, A.M.C.A. Koster and T. Wolle, Contraction and treewidth lower bounds, *Proceedings 12th Annual European Symposium on Algorithms, ESA2004, Springer-Verlag, Lecture Notes in Computer Science* **3221** (2004) 628–639.

[12] H.L. Bodlaender and U. Rotics, Computing the treewidth and the minimum fill-in with the modular decomposition, *Algorithmica* **36** (2003) 543–559.

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

[14] H.L. Bodlaender, A.M.C.A. Koster, F.v.d. Eijkhof and L.C. van der Gaag, Preprocessing for triangulation of probabilistic networks, *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence, San Francisco, Morgan Kaufmann* **3221** (2001) 32–39.

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

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

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

[18] H.L. Bodlaender and R.H. Möhring, The pathwidth and treewidth of cographs, *SIAM J. Disc. Meth.* **6** (1993) 81–88.

[19] H.L. Bodlaender, T. Kloks and D. Kratsch, *Treewidth and pathwidth of permutation graphs*, (Technical Report, Utrecht University, 1992)

[20] V. Bouchitté, D. Kratsch, H. Muller and I. Todinca, On treewidth approximations, *Disc. Appl. Math.* **136** (2004) 183–196.

[21] V. Bouchitté and I. Todinca, Listing all potential maximal cliques, *Theor. Comp. Sc.* **276** (2002) 17–32.

[22] V. Bouchitté and I. Todinca, Approximating the treewidth of AT-free graphs, *Disc. Appl. Math.* **131** (2003) 11–37.

[23] A. Brandstädt, Special graph classes - a survey, *Schriftenreihe des Fachbereichs Mathematik, SM-DU-199, Universität Duisburg Gesamthochschule* (1991)

[24] F. Clautiaux, A. Moukrim and S. Négre, New lower and upper bounds for graph treewidth *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003, Springer-Verlag, Lecture Notes in Computer Science* **2647** (2003) 70–80.

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

[26] W. Cook and P.D. Seymour, *An algorithm for the ring-routing problem, Bellcore technical memorandum, Bellcore* (1993)

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

[28] F.V. Fomin, D. Kratsch and I. Todinca, Exact (exponential) algorithms for treewidth and minimum fill-in, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming* (2004) 568–580.

[29] V. Gogate and R. Dechter, A complete anytime algorithm for treewidth, *Proceedings UAI2004, Uncertainty in Artificial IntelligenPartial curriculum in INF334-Advanced algorithmical techniques, Department of Informatics, University of Bergen, Norway, 2005* (2004)

[30] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, (2nd edition, Annals of Discrete Mathematics 57, Elsevier, 2004).

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

[32] P. Heggernes, Treewidth, partial k-trees, and chordal graphs, *Partial curriculum in INF334-Advanced algorithmical techniques, Department of Informatics, University of Bergen, Norway, 2005.*

[33] P. Heggernes, J.A. Telle and Y. Villanger, Computing minimal triangulations in time $O(n^{\alpha} log(n)) = o(n^{2.376})$, *SIAM J. Disc. Math.* **19(4)** (2005) 900–913.

[34] P. Heggernes and Y. Villanger, Efficient implementation of a minimal triangulation algorithm, *Proceedings of the 10th Annual European Symposium on Algorithms, ESA'2002, Springer Verlag, Lecture Notes in Computer Science* **2461** (2002) 550–561.

[35] U. Kjaerulff, Optimal decomposition of probalistic networks by simulated annealing, *Statistics and Computing* **2** (1992) 7–17.

[36] T. Kloks, *Treewidth: Computations and Approximations*, (Lecture Notes in Computer Science, Springer-Verlag, 1994)

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

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

[39] A. Kornai and Z. Tuza, Narrowness, pathwidth, and their application in natural language processing, *Disc. Appl. Math.* **36** (1992) 87–92.

[40] A.M.C.A. Koster, Frequency assignment - Models and Algorithms, *PhD Thesis, University of Maastricht, Netherlands* (1999).

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

[42] S.J. lauritzen and D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, *The Jorunal of the Royal Statistical Society, Series B (Methodological)* **50** (1988) 157–224.

[43] B. Lucena, A new lower bound for treewidth using maximum cardinality search, *SIAM J. Disc. Math.* **16(3)** (2003) 345–353.

[44] J. Matousek and R. Thomas, Algorithms for finding tree-decomposition of graphs, *J. Algorithms* **12** (1991) 1–22.

[45] L. Perkovic and B. Reed, An improved algorithm for finding tree decompositions of small width, *Proceedings 25th Int. Workshop on Graph Theoretic Concepts in Computer Science, WG'99, Springer-Verlag, Lecture Notes in Computer Science* **1665** (1999) 148–154.

[46] S. Ramachandramurthi, A lower bound for treewidth and its consequences, *Proceedings 20th International Workshop on Graph theoretic Concepts in Computer Science WG94, Springer-Verlag, Lecture Notes in Computer Science* **903** (1995) 14–25.

[47] N. Robertson and P.D. Seymour, Graph minors II. Algorithmic aspects of treewidth, *J. Algorithms* **7** (1986) 309–322.

[48] D.P. Sanders, On linear recognition of treewidth at most four, *SIAM J. Disc. Math.* **9(1)** (1996) 101–117.

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

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