On exact algorithms for treewidth

Hans L. Bodlaender

Fedor V. Fomin

Arie M. C. A. Koster

Dieter Kratsch

Dimitrios M. Thilikos

Department of Information and Computing Sciences,
Utrecht University

Technical Report UU-CS-2006-032

www.cs.uu.nl

ISSN: 0924-3275

On exact algorithms for treewidth[*]

Hans L. Bodlaender[†] Fedor V. Fomin[‡] Arie M.C.A. Koster[§]
Dieter Kratsch[¶] Dimitrios M. Thilikos[||]

Abstract

We give experimental and theoretical results on the problem of computing the treewidth of a graph by exact exponential time algorithms using exponential space or using only polynomial space. We first report on an implementation of a dynamic programming algorithm for computing the treewidth of a graph with running time $O^*(2^n)$. This algorithm is based on the old dynamic programming method introduced by Held and Karp for the TRAVELING SALESMAN problem. We use some optimizations that do not affect the worst case running time but improve on the running time on actual instances and can be seen to be practical for small instances. However, our experiments show that the space used by the algorithm is an important factor to what input sizes the algorithm is effective. For this purpose, we settle the problem of computing treewidth under the restriction that the space used is only polynomial. In this direction we give a simple $O^*(4^n)$ algorithm that requires *polynomial* space. We also show that with a more complicated algorithm, using balanced separators, TREEWIDTH can be computed in $O^*(2.9512^n)$ time and polynomial space.

1 Introduction

The use of treewidth in several application areas requires efficient algorithms for computing the treewidth and optimal width tree decompositions of given graphs. In the past years,

[*]This research was partially supported by the project *Treewidth and Combinatorial Optimization* with a grant from the Netherlands Organization for Scientific Research NWO and by the Research Council of Norway and by the DFG research group "Algorithms, Structure, Randomness" (Grant number GR 883/9-3, GR 883/9-4). The research of the last author was supported by the Spanish CICYT project TIN-2004-07925 (GRAMMARS).

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

[‡]Department of Informatics, University of Bergen, N-5020 Bergen, Norway. fomin@ii.uib.no

[§]Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin, Germany. koster@zib.de

[¶]LITA, Université de Metz, F-507045 Metz Cedex 01, France. kratsch@sciences.univ-metz.fr

[||]Department of Mathematics, National and Capodistrian University of Athens, Panepistimioupolis, GR-15784, Athens, Greece. sedthilk@math.uoa.gr a large number of papers appeared studying the problem to determine the treewidth of a graph, including both theoretical and experimental results, see e.g., [4] for an overview. Since the problem is NP complete [1], there is a little hope in finding an algorithm which can determine the treewidth of a graph in polynomial time. There are several exponential time (exact) algorithms known in the literature for the treewidth problem. (See the surveys [11, 26] for an introduction to the area of exponential algorithms.) Arnborg et al. [1] gave an algorithm that tests in $O(n^{k+2})$ time if a given graph has treewidth at most k. It is not hard to observe that the algorithm runs for variable k in $O^*(2^n)$ time[1]. See also [22]. In 2004, Fomin et al. [12] presented an $O(1.9601^n)$ algorithm to compute the treewidth based on minimal separators and potential maximal cliques of graphs, using the paradigms introduced by Bouchitté and Todinca [7, 6]. The analysis of the algorithm of Fomin et al. from [12] was improved by Villanger [25], who showed that the treewidth of a graph can be computed in $O(1.8899^n)$ time. While the algorithms from [12, 25] provide the best known running time, they are based on computations of potential maximal cliques and are difficult to implement.

In this paper we try another approach to compute the treewidth, which seems to be much more suitable for implementations. While Treewidth is usually formulated as the problem to find a tree decomposition of minimum width, it is possible to formulate it as a problem to find a linear ordering of (the vertices of) the graph such that a specific cost measure of the ordering is as small as possible. Several existing algorithms and heuristics for treewidth are based on this linear ordering characterization of treewidth, see e.g., [2, 8, 15]. In this paper, we exploit this characterization again, and a lesser known property of the characterization. Thus, we can show that an old dynamic programming method, introduced by Held and Karp for the Traveling Salesman problem [18] in 1962 can be adapted and used to compute the treewidth of given graphs. Suppressing polynomial factors, time and space bounds of the algorithm for treewidth is the same as that of the algorithm of Held and Karp for TSP: $O^*(2^n)$ running time and $O^*(2^n)$ space. The Held-Karp algorithm tabulates some information for pairs (S, v), where S is a subset of the vertices, and v is a vertex (from S); a small variation of the scheme allows us to save a factor $O(n)$ on the space for the problems considered in this paper: we tabulate information for all subsets $S \subseteq V$ of vertices.

We have carried out experiments that show that the method works well to compute the treewidth of graphs of size up to around forty to fifty. For larger graphs, the space requirements of the algorithm appear to be the bottleneck. Thus, this raises the question: are there polynomially space algorithms to compute the treewidth having running time of the form $O^*(c^n)$ for some constant c? In this paper we answer this question in the affirmative. We show that there is an algorithm to compute the treewidth that uses $O^*(4^n)$ time and only polynomial space. This algorithm uses a simple recursive divide-and-conquer technique and is similar to the polynomial space algorithm of Gurevich and Shelah [17] for Hamiltonian Path.

[1]We sometimes use O^*-notation which is a modified O-notation introduced by Woeginger [26] suppressing all polynomially bounded factors.

Finally, we further provide theoretical results improving upon the running time for the polynomial space algorithm for treewidth. Using balanced separators, we obtain an algorithm for TREEWIDTH that uses $O^*(2.9512^n)$ time and polynomial space. It should be noted that this result is only theoretical: the algorithm must consider many subsets of a specific size of the set of vertices. Thus, we did not carry out an experimental evaluation of the polynomial space algorithms.

2 Preliminaries

2.1 Definitions

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

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

- *For all $v \in V$, there exists an $i \in I$ with $v \in X_i$.*
- *For all $\{v, w\} \in E$, there exists an $i \in I$ with $v, w \in X_i$.*
- *For all $v \in V$, the set $I_v = \{i \in I \mid v \in X_i\}$ forms a connected subgraph (subtree) of T.*

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

The following alternative characterization of treewidth is well known, see e.g., [3].

Proposition 2 *Let $G = (V, E)$ be a graph, k an integer. The following are equivalent.*

1. *G has treewidth at most k.*
2. *G has a triangulation $H = (V, F)$ with the maximum size of a clique in H at most $k + 1$.*
3. *G has a minimal triangulation $H = (V, F)$ with the maximum size of a clique in H at most $k + 1$.*

2.2 Treewidth as a Linear Ordering Problem

It is well known that treewidth can be formulated as a linear ordering problem, and this is exploited in several algorithms for determining the treewidth, see e.g., [2, 8, 9, 15].

A *linear ordering* of a graph $G = (V, E)$ is a bijection $\pi : V \rightarrow \{1, 2, \ldots, |V|\}$. For a linear ordering π and $v \in V$, we denote by $\pi_{<,v}$ the set of vertices that appear before v in the ordering: $\pi_{<,v} = \{w \in V \mid \pi(w) < \pi(v)\}$. Likewise, we define $\pi_{\leq,v}$, $\pi_{>,v}$, and $\pi_{\geq,v}$. A linear ordering π of G is a *perfect elimination scheme*, if for each vertex, its higher numbered neighbors form a clique, i.e., for each $i \in \{1, 2, \ldots, |V|\}$, the set $\{\pi^{-1}(j) \mid \{\pi^{-1}(i), \pi^{-1}(j)\} \in E \wedge j > i\}$ is a clique. It is well-known that a graph has a perfect elimination scheme, if and only if it is chordal, see [16, Chapter 4].

For arbitrary graphs G, a linear ordering π defines a triangulation H of G that has π as perfect elimination scheme. The *triangulation with respect to* π of G is built as follows: first, set $G_0 = G$, and then for $i = 1$ to n, G_i is obtained from G_{i-1} by adding an edge between each pair of non adjacent higher numbered neighbors of $\pi^{-1}(i)$. One can observe that the resulting graph $H = G_n$ is chordal, has π as perfect elimination scheme, and contains G as subgraph.

For our algorithms, we want to avoid working with the triangulation explicitly. The following predicate allows us to 'hide' the triangulation. For a linear ordering π, and two vertices $v, w \in V$, we say $P_\pi(v, w)$ holds, if and only if there is a path $v, x_1, x_2, \ldots, x_r, w$ from v to w in G, such that for each i, $1 \leq i \leq r$, $\pi(x_i) < \pi(v)$, and $\pi(x_i) < \pi(w)$. In other words, $P_\pi(v, w)$ is true, if and only if there is a path from v to w such that all internal vertices are before v and w in the ordering π. Note that the definition implies that $P_\pi(v, w)$ is always true when $v = w$ or when $\{v, w\} \in E$.

With $R_\pi(v)$, we denote the number of higher numbered vertices $w \in V$ for which $P_\pi(v, w)$ holds, i.e., $R_\pi(v) = |\{w \in V \mid \pi(w) > \pi(v) \wedge P_\pi(v, w)\}|$. The proof of the following proposition is an immediate consequence of a lemma of Rose et al. [21]. (See also [3, 8, 9].)

Proposition 3 *Let $G = (V, E)$ be a graph, and k a non-negative integer. The treewidth of G is at most k iff there is a linear ordering π of G, such that for each $v \in V$, $R_\pi(v) \leq k$.*

Proof: We use the following result from [21]. For a given graph $G = (V, E)$, and a linear ordering π, we have for each pair of disjoint vertices $v, w \in E$: $\{v, w\}$ is an edge in the triangulation $H = (V, E_H)$ with respect to π, if and only if $P_\pi(v, w)$ is true. Also, we use the result of Fulkerson and Gross [13] that if π is a perfect elimination scheme of chordal graph $H = (V, E_H)$, then the maximum clique size of H is one larger than the maximum over all $v \in V$ of the number of higher numbered neighbors $|\{v, w\} \in E_H \mid \pi(w) > \pi(v)\}|$.

Now, the treewidth of G is at most k, if and only if there is a triangulation $H = (V, E_H)$ of G with maximum clique size at most k (Proposition 3), if and only if for a perfect elimination scheme π of triangulation H, we have that for each $v \in V$:

$$\begin{aligned} k + 1 &\leq |\{v, w\} \in E_H \mid \pi(w) > \pi(v)\}| \\ &= |\{w \in V \mid P_\pi(v, w) \wedge \pi(w) > \pi(v)\}| \\ &= R_\pi(v) \end{aligned}$$

□

Let $\Pi(S)$ be the set of all permutations of a set S. So, $\Pi(V)$ is the set of all linear orderings of G. Write $\Pi(S, R)$ for the collection of permutations of S, that end with vertices in R, i.e., with the property that for each $v \in R$: $\pi(v) \geq |S| - |R| + 1$.

For a graph $G = (V, E)$, a set of vertices $S \subseteq V$ and a vertex $v \in V - S$, we define

$$Q_G(S, v) = \\ |\{w \in V - S - \{v\} \mid \text{there is a path from } v \text{ to } w \text{ in } G[S \cup \{v, w\}]\}|$$

If G is clear from the context, we drop the subscript G. Let us note that $Q(S, v)$ can be computed in $O(n + m)$ time by checking for each $w \in V - S - \{v\}$ whether w has a neighbor in the component of $G[S \cup \{v\}]$ containing v. Also note that $R_\pi(v) = |Q(\pi_{<,v}, v)|$ for any $v \in V$, and any linear ordering $\pi \in \Pi(V)$.

3 A dynamic programming algorithm for treewidth

The results of this section are based on the observation that the value $R_\pi(v)$ only depends on v, G, and the set of vertices left of v in π.

We define

$$TW_G(S) = \min_{\pi \in \Pi(V)} \max_{v \in S} |Q_G(\pi_{<,v}, v)|.$$

Again, usually G is clear from the context, and dropped as subscript. The main idea of the algorithm in this section is that we compute $TW_G(S)$ for all subsets $S \subseteq V$ using dynamic programming. The next lemma shows that this solves the treewidth problem.

Lemma 4 *For each graph $G = (V, E)$, the treewidth of G equals $TW(V)$.*

Proof: Using Proposition 3, we have

$$\begin{aligned} \mathbf{tw}(G) &= \min_{\pi \in \Pi(V)} \max_{v \in V} R_\pi(v) \\ &= \min_{\pi \in \Pi(V)} \max_{v \in V} |Q(\pi_{<,v}, v)| \\ &= TW(V). \end{aligned}$$

□

The following lemma gives the recursive formulation that allows us to compute the values $TW(S)$ with dynamic programming.

Lemma 5 *For any graph $G = (V, E)$, and any set of vertices $S \subseteq V$, $S \neq \emptyset$,*

$$TW(S) = \min_{v \in S} \max\{TW(S - \{v\}), |Q(S - \{v\}, v)|\}$$

Proof: Let $\pi \in \Pi(V)$ be a permutation with $TW(S) = \max_{w \in S} |Q(\pi_{<,w}, w)|$. Suppose v is the vertex from S with the largest index in π, i.e., the vertex with $S \subseteq \pi_{\leq,v}$.

From the definition of TW, it directly follows that $TW(S) \geq TW(S - \{v\})$. Also, as $S \subseteq \pi_{\leq,v}$, we have $|Q(S - \{v\}, v)| \leq |Q(\pi_{<,v}, v)|$. Hence

$$\begin{aligned} TW(S) &\geq \max\left\{TW(S - \{v\}), \max_{w \in S} |Q(\pi_{<,w}, w)|\right\} \\ &\geq \max\{TW(S - \{v\}), |Q(\pi_{<,v}, v)|\} \\ &\geq \max\{TW(S - \{v\}), |Q(S - \{v\}, v)|\} \end{aligned}$$

Thus,

$$TW(S) \geq \min_{v \in S} \max\{TW(S - \{v\}), |Q(S - \{v\}, v)|\}$$

For the other direction, let v be an arbitrary vertex from S. Suppose $\pi \in \Pi(V)$ is a permutation with $TW(S - \{v\}) = \max_{w \in S} |Q(\pi_{<,w}, w)|$. Let $\pi' \in \Pi(V)$ be a permutation, obtained by first taking the vertex in $S - \{v\}$ in the order as they appear in π, then taking v, and then taking the vertices in $V - S$ in an arbitrary order. Note that we have for all $w \in S - \{v\}$, $\pi'_{<,w} \subseteq \pi_{<,w}$, and that $\pi'_{<,v} = S - \{v\}$. Now

$$\begin{aligned} TW(S) &\leq \max_{w \in S} |Q(\pi'_{<,w}, w)| \\ &= \max\left\{\max_{w \in S - \{v\}} |Q(\pi'_{<,w}, w)|, |Q(\pi'_{<,v}, v)|\right\} \\ &\leq \max\left\{\max_{w \in S - \{v\}} |Q(\pi_{<,w}, w)|, |Q(S - \{v\}, v)|\right\} \\ &= \max\{TW(S - \{v\}), |Q(S - \{v\}, v)|\} \end{aligned}$$

□

This gives us the following relatively simple algorithm for TREEWIDTH with $O^*(2^n)$ worst case running time and space.

Theorem 6 *The treewidth of a graph on n vertices can be determined in $O^*(2^n)$ time and $O^*(2^n)$ space.*

Proof: By Lemma 5, we almost directly obtain a Held-Karp-like dynamic programming algorithm for the problem. In order of increasing sizes, we compute for each $S \subseteq V$, $TW(S)$ using Lemma 5. Below, we give pseudo-code for a simple form of the algorithm Dynamic-Programming-Treewidth.

The algorithm uses $O^*(2^n)$ time, as we do polynomially many steps per subset of V. The algorithm also keeps all subsets of V and thus uses $O^*(2^n)$ space. □

Algorithm 1 Dynamic-Programming-Treewidth(Graph $G = (V, E)$)

```
Set $TW(\emptyset) = -\infty$.
for i = 1 to $n$ do
  for all sets $n \subset V$ with $|S| = n$ do
    Set $TW(S) = \min_{v \in S} \max\{TW(S - \{v\}), |Q(S - \{v\}, v)|\}$
  end for
end for
return $TW(V)$
```

4 A recursive algorithm for treewidth

For vertex subsets $L, S \subseteq V, S \cap L = \emptyset$ of a graph $G = (V, E)$ we define

$$TWR(L, S) = \min_{\pi \in \Pi(V)} \max_{v \in S} |Q(L \cup \pi_{<,v}, v)|$$

The intuition behind $TWR(L, S)$ is as follows: we investigate the resulting cost of the 'best' ordering of the vertices in S, assuming that all vertices in L are left of all vertices in S, and all vertices in $V - (L \cup S)$ are right of all vertices in S. We observe that if $S = \{v\}$, then $TWR(L, S) = |Q(L, v)|$. Also, by definition, $TWR(\emptyset, S) = TW(S)$ and therefore $\mathbf{tw}(G) = TWR(\emptyset, V)$.

Lemma 7 *Let $G = (V, E)$ be a graph, let $S \subseteq V$, $|S| \geq 2$, $L \subseteq V$, $L \cap S = \emptyset$, $1 \leq k < |S|$. Then*

$$TWR(L, S) = \min_{S' \subseteq S, |S'| = k} \max\{TWR(L, S'), TWR(L \cup S', S - S')\}$$

Proof: Suppose $\pi \in \Pi(V)$ fulfills $TWR(L, S) = \max_{v \in S} |Q(L \cup \pi_{<,v}, v)|$. Let S' be the first k vertices in S that appear in π, i.e., all vertices in $S - S'$ have a higher index in π than any element in S'. Now,

$$\begin{aligned} TWR(L, S) &= \max_{v \in S} |Q(L \cup \pi_{<,v}, v)| \\ &= \max\left\{\max_{v \in S'} |Q(L \cup \pi_{<,v}, v)|, \max_{v \in S - S'} |Q(L \cup \pi_{<,v}, v)|\right\} \\ &\geq \max\left\{TWR(L, S'), \max_{v \in S - S'} |Q(L \cup S' \cup \pi_{<,v}, v)|\right\} \\ &\geq \max\{TWR(L, S'), TWR(L \cup S', S - S')\} \end{aligned}$$

For the other direction, suppose that $S' \subseteq S$ fulfills

$$\begin{gathered} \max\{TWR(L, S'), TWR(L \cup S', S - S')\} = \\ \min_{S'' \subseteq S, |S'| = k} \max\{TWR(L, S''), TWR(L \cup S', S - S'')\} \end{gathered}$$

Let $\pi' \in \Pi(V)$ be a permutation with $TWR(L, S') = \max_{v \in S'} |Q(L \cup \pi'_{<,v}, v)|$. Let $\pi'' \in \Pi(V)$ be a permutation with $TWR(L \cup S', S - S') = \max_{v \in S - S'} |Q(L \cup S' \cup \pi'_{<,v}, v)|$. We now build a permutation $\pi \in \Pi(V)$ in the following way. First, we take the elements in L, in some arbitrary order. Then, we take the elements in S', in the order as they appear in π'. I.e., for $v, w \in S'$, we have that v has a smaller index than w in π, if and only if v has a smaller index than w in π'. Then, we take the elements in $S - S'$, in the order as they appear in π''. I.e., for $v, w \in S - S'$, we have that v has a smaller index than w in π, if and only if v has a smaller index than w in π''. Also, for all $v \in S'$, $w \in S - S'$, v has a smaller index than w in π. We end permutation π by taking the elements in $V - S - L$ in some arbitrary order. For this order π, we have

$$\begin{aligned}
TWR(L, S) &\leq \max_{v \in S} |Q(L \cup \pi_{<,v}, v)| \\
&= \max \left\{ \max_{v \in S} |Q(L \cup \pi_{<,v}, v)|, \max_{v \in S - S'} |Q(L \cup \pi_{<,v}, v)| \right\} \\
&\leq \max \left\{ \max_{v \in S} |Q(L \cup \pi'_{<,v}, v)|, \max_{v \in S - S'} |Q(L \cup S' \cup \pi'_{<,v}, v)| \right\} \\
&= \max \{TWR(L, S'), TWR(L \cup S', S - S')\}
\end{aligned}$$

This proves the result. □

By making use of Lemma 7 with $k = \lfloor |S|/2 \rfloor$, we obtain the following result.

Theorem 8 *The treewidth of a graph on n vertices can be determined in $O^*(4^n)$ time and polynomial space.*

Proof: Lemma 7 is used to obtain Algorithm 2. This algorithm computes $TWR(L, S)$ recursively. Algorithm 2 computes the treewidth of the graph G when calling Recursive-Treewidth(G,$\emptyset$,V). Since $\mathbf{tw}(G) = TWR(\emptyset, V)$, this gives the answer to the problem.

The algorithm clearly uses polynomial space: recursion depth is $O(\log n)$, and per recursive step, only polynomial space is used. To estimate the running time, suppose that Recursive-Treewidth(G,L,S) costs $T(n, r)$ time with n the number of vertices of G and $r = |S|$. All work, except the time of recursive calls, has its time bounded by a polynomial in n, $p(n)$. As we make less than 2^{r+1} recursive calls, each with a set S' with $|S'| \leq \lceil |S|/2 \rceil$, we have

$$T(n, r) \leq 2^{r+1} \cdot T(n, \lceil r/2 \rceil) + p(n) \qquad (1)$$

From this, it follows that there is a polynomial $p'(n)$, such that

$$T(n, r) \leq 4^r \cdot p'(n) \qquad (2)$$

As the algorithm is called with $|S| = n$, it uses $O^*(4^n)$ time. □

In Section 5, we report on an implementation of the $O^*(2^n)$ algorithm for TREEWIDTH (with additional improvements to decrease the time for actual instances). So, while the $O^*(2^n)$ algorithm does not give a theoretical improvement, it can be seen to be of practical use. In Section 6, we improve upon the running time for the case of polynomial space.

Algorithm 2 Recursive-Treewidth(Graph G, Vertex Set L, Vertex Set S)

```
if $|S|=1$ then
    Suppose $S = \{v\}$.
    return $Q(L, v)$
end if
Set Opt $= \infty$.
for all sets $S' \subseteq S$, $|S'| = \lfloor |S|/2 \rfloor$ do
    Compute $v1 =$ Recursive-Treewidth$(G, L, S')$;
    Compute $v2 =$ Recursive-Treewidth$(G, L \cup S', S - S')$;
    Set Opt $= \min\{\text{Opt}, \max\{v1, v2\}\}$;
end for
return Opt
```

5 Experimental results

In this section, we comment on the experiments we have carried out for the dynamic programming algorithm for computing the treewidth of a given graph.

For practical considerations, we use a scheme that is slightly different than that of Theorem 6. We can note that it is not useful to perform computations with sets S for which $TW(S)$ is larger than or equal to a known upper bound up on the treewidth of G: these cannot lead to a smaller bound on the treewidth of G. Thus, in order to save time and space in practice, we avoid handling some of such S. We compute collections $TW_1, TW_2, \ldots, TW_n$. Each collection TW_i ($1 \leq i \leq n$) contains pairs $(S, TW(S))$ with $|S| = i$. The collection for sets of size $i > 1$ is built as follows: for each pair $(S, r) \in TW_{i-1}$ and each $x \in V - S$, we compute $r' = \max\{r, |Q(S, x)|\}$. If $r' < up$, then we check if there is a pair $(S \cup \{x\}, t)$ in TW_i for some t, and if so, replace it by $(S \cup \{x\}, \min(t, r'))$. If $r' < up$, but there is no such pair $(S \cup \{x\}, t)$ for some t is in TW_i, then we insert $(S \cup \{x\}, r')$ in TW_i.

The scheme is shown in Algorithm 3.

In our implementation, we use two additional optimizations that appeared to give significant savings in time and space consumption. The following simple lemma gives the first idea.

Lemma 9 *Let $G = (V, E)$ be a graph, and let $S \subseteq V$. The treewidth of G is at most* $\max\{TW(S), n - |S| - 1\}$.

Proof: Take $\pi \in \Pi(V)$ with $TW(S) = \max_{v \in S} |Q_G(\pi_{<,v}, v)|$. Now, take a linear ordering π' of G that starts with the vertices in S in the same order as these are in π, and then the vertices in $V - S$ in some arbitrary order. Now Now we claim that

$$\mathbf{tw}(G) \leq \max_{v \in V} |Q_G(\pi'_{<,v}, v)| \leq \max\{TW(S), n - |S| - 1\}$$

For $v \in S$, $|Q_G(\pi'_{<,v}, v)| = |Q_G(\pi_{<,v}, v)| \leq TW(S)$. If $v \in V - S$, $|Q_G(\pi'_{<,v}, v)| \leq |V - S - \{v\}| \leq n - |S| - 1$. $\square$

Algorithm 3 Algorithm TWDP (Graph $G = (V, E)$)

```
$n = |V|$.
Compute some initial upper bound $up$ on the treewidth of $G$. (E.g., set $up = n - 1$.)
Let $TW_0$ be the set, containing the pair $(\emptyset, -\infty)$.
for $i = 1$ to $n$ do
   Set $TW_i$ to be an empty set.
   for each pair $(S, r)$ in $TW_{i-1}$ do
      for each vertex $x \in V - S$ do
         Compute $q = |\{w \in V - S - \{x\} \mid w \sim_S x\}|$.
         Set $r' = \min\{r, q\}$.
         if $r' < up$ then
            if There is a pair $(S \cup \{x\}, t)$ in $TW_i$ for some $t$ then
               Replace the pair $(S \cup \{x\}, t)$ in $TW_i$ by $(S \cup \{x\}, \min(t, r'))$.
            else
               Insert the pair $(S \cup \{x\}, r')$ in $TW_i$.
            end if
         end if
      end for
   end for
end for
if $TW_n$ contains a pair $(V, r)$ for some $r$ then
   return $r$
else
   return $up$
end if
```

Lemma 9 shows correctness of the following rule that was used in the implementation: we keep an upper bound *up* for the treewidth of G, initially set by the user or set to $n-1$. Each time, we get a pair (S, r) in a collection TW_i, either by insertion, or by replacement of an existing pair, we set the upper bound *up* to the minimum of *up* and $n-|S|-1 = n-i-1$. Moreover, when handling a pair (S, r) from TW_{i-1}, it is first checked if r is smaller than *up*; if not, then this pair cannot contribute to an improvement of the upper bound, and hence is skipped. Our second optimization is stated in Lemma 11. We use the following basic fact on chordal graphs and cliques.

Lemma 10 *Let $H = (V, E_H)$ be a chordal graph and let $C \subseteq V$ induce a clique in G. Then H has a perfect elimination scheme π that ends with C, i.e., such that for each $v \in C$: $\pi(v) \geq |V| - |C| + 1$.*

Proof: Consider (for instance) the Maximum Cardinality Search (MCS) algorithm from [23]. When given a chordal graph H, it produces a perfect elimination scheme of H. It is easy to see that MCS can produce an ordering with the vertices of C at the highest numbered positions. □

Lemma 11 *Let $C \subseteq V$ induce a clique in graph $G = (V, E)$. The treewidth of G equals* $\max\{TW(V - C), |C| - 1\}$.

Proof: Using the proof method of Proposition 3 and Proposition 10, we obtain that the treewidth of G is at most some non-negative integer k, if and only if there is a linear ordering $\pi \in \Pi(V, C)$ of G, (i.e., with π ending with the vertices in C), such that for each $v \in V$, $R_\pi(v) \leq k$. In a similar, slightly more complicated way as for the proof of Lemma 4, we have

$$\begin{aligned}
\mathbf{tw}(G) &= \min_{\pi \in \Pi(V,C)} \max_{v \in V} R_\pi(v) \\
&= \min_{\pi \in \Pi(V,C)} \max \left\{ \max_{v \in V-C} R_\pi(v), \max_{v \in C} R_\pi(v) \right\} \\
&= \min_{\pi \in \Pi(V,C)} \max \left\{ \max_{v \in V-C} R_\pi(v), |C| - 1 \right\} \\
&= \max \left\{ |C| - 1, \min_{\pi \in \Pi(V,C)} R_\pi(v) \right\} \\
&= \max\{|C| - 1, TW(V - C)\}
\end{aligned}$$

□

By Lemma 11, we can restrict the sets S to elements from $V - C$ for some clique C; in particular for the maximum clique. Although it is NP-hard to compute the maximum clique in a graph, it can be computed extremely fast for the graphs considered. In our program, we use a simple combinatorial branch-and-bound is used to compute all maximum cliques. It recursively extends a clique by all candidate vertices once.

The algorithm was implemented in C++, using the Boost graph library, as part of the Treewidth Optimization Library TOL, a package of algorithms for the treewidth of graphs. The package includes preprocessing, upper bound, and lower bound algorithms for treewidth. Experiments were carried out on a number of graphs taken from applications; several were used in other experiments. See [24] for the used graphs, information on the graphs, and other results of experiments to compute the treewidth. The experiments were carried out on a Sun computer with 4 AMD Dualcore Opteron 875, 2.2 GHz processor and at most 20 GB of internal memory available. The program did not use parallelism.

In Table 1 the results of our experiments on a number of graphs are reported. Besides instance name, number of vertices, number of edges, and the computed treewidth, we report on the CPU time in seconds and the maximum number of sets (S, r), considered at once, $\max |TW| = \max_{i=0,\ldots,n} |TW_i|$ in a number of cases. First, we report on the CPU time and maximum number of sets for the case that no initial upper bound *up* is exploited. Next, we report on the case where we use an initial upper bound, displayed in the column *up*. The last two columns report on the experiments in which the algorithm is advanced by both an initial upper bound *up* and a maximum clique C of size ω. In several instances reported in [24], the best bound obtained from a few upper bound heuristics, and the lower bound obtained by the LBP+(MMD+) heuristic match, and then we have obtained in a relatively fast way an exact bound on the treewidth of the instance graph. In other cases, these bounds do not match. Then, when the graph is not too large, the dynamic programming algorithm can be of good use.

A nice example is the celar03 graph. This graph has 200 vertices and 721 edges. A combination of different preprocessing techniques yield an equivalent instance celar03-pp-001 which has 38 vertices and 238 edges. Existing upper bound heuristics gave a best upper bound of 15, while the lower bound of the LBP+(MMD+) heuristic was 13. With the dynamic programming algorithm with 15 as input for an upper bound, we obtained the exact treewidth of 14 for this graph, and hence also for celar03.

				no *up*, no C			with *up*, no C			with *up*, w C	
instance	$\|V\|$	$\|E\|$	**tw**	CPU	max $\|TW\|$	*up*	CPU	max $\|TW\|$	ω	CPU	max $\|TW\|$
myciel3	11	20	5	0.00	240	5	0.00	35	2	0.00	21
myciel4	23	71	10	7.64	296835	10	0.14	4422	2	0.12	4064
queen5-5	25	160	18	0.15	18220	18	0.02	944	5	0.02	392
queen6-6	36	290	25	36.43	2031716	26	1.16	18872	6	0.36	6994
queen7-7	49	476	35	-	-	37	1012.12	96517095	7	248.03	24410915
pathfinder-pp	12	43	6	0.00	107	6	0.00	1	6	0.00	1
oesoca+-pp	14	75	11	0.00	48	11	0.00	5	9	0.00	5
fungiuk	15	36	4	0.07	4713	4	0.00	4	5	0.00	4
weeduk	15	49	7	0.02	2906	7	0.00	35	8	0.00	35
munin-kgo-pp	16	41	5	0.11	6892	5	0.00	2	4	0.00	2
wilson	21	27	3	14.44	350573	3	0.08	2412	3	0.06	2342
water-pp	22	96	9	1.60	77286	10	0.04	816	6	0.01	475
oow-trad-pp	23	54	6	42.91	1065120	6	0.09	2953	4	0.05	1895
barley-pp	26	78	7	349.31	6110572	7	0.61	13597	5	0.29	7971
oow-bas	27	54	4	1579.38	19937301	4	0.01	303	4	0.00	111
oow-solo-pp	27	63	6	1059.50	17048070	6	0.91	22484	4	0.30	9426
ship-ship-pp	30	77	8	-	-	9	291.20	3062863	4	50.75	820910
water	32	123	9	-	-	10	12.59	127545	6	1.53	25874
oow-trad	33	72	6	-	-	6	129.55	1162650	4	14.55	178846
mildew	35	80	4	-	-	4	2.98	33045	4	0.35	5431
mainuk	48	198	7	-	-	8	-	-	8	2251.97	11748147
celar03-pp-001	38	238	14	-	-	15	121.29	911918	8	4.36	55504

Table 1: Experimental results for some DIMACS vertex coloring graphs, some probabilistic networks and frequency assignment graph celar03-pp-001

The algorithm can also be used as a lower bound heuristic: give the algorithm as 'upper bound' a conjectured lower bound ℓ: when it terminates, it either has found the exact treewidth, or we know that ℓ is indeed a lower bound for the treewidth of the input graph. In a few cases, we could thus increase the lower bound for the treewidth of considered instances, e.g., for the treewidth of the queen8-8 graph (the graph modeling the possible moves of a queen on an 8 by 8 chessboard) the lower bound could be improved from 27 to 35.

For larger graphs, the above idea can be combined by an idea exploited earlier in various papers. Given a graph G and a minor G' of G, $\mathbf{tw}(G') \leq \mathbf{tw}(G)$. In [5, 15, 20], a lower bound on $\mathbf{tw}(G')$ is computed to obtain a lower bound for G. With Algorithm 3, we can compute $\mathbf{tw}(G')$ exactly to obtain a lower bound for $\mathbf{tw}(G)$. For the 1024 vertices graph pignet2-pp, we have generated a sequence of minors by repeatedly contracting a minimum degree vertex with a neighbor with least number of common neighbors (see [5]).

Figure 1 shows the treewidth (right y-scale) for the minors with 70 to 79 vertices. Moreover, the maximum number of sets for three different upper bounds is reported (left y-scale, logarithmic). If the used upper bound is less than or equal to the treewidth, no feasible solution is found in the end. The best known lower bound for pignet2-pp is increased from 48 to 59 by the treewidth of the 79 vertex-minor. Figure 1 shows one more time the impact of the upper bound on the memory consumption (and time consumption) of the algorithm.

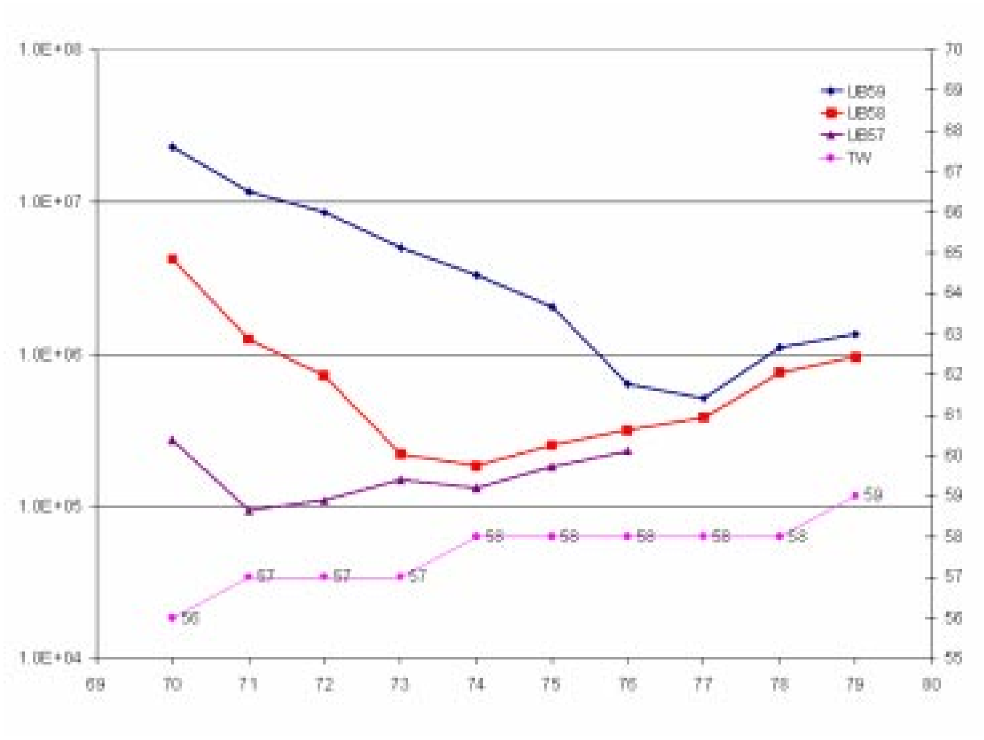

Figure 1: Maximum number of subsets S during algorithm for different upper bounds.

6 Improved polynomial space algorithms for treewidth

In this section, we give a faster exponential time algorithm with polynomial space for Treewidth. The algorithm is based on results of earlier sections combined with techniques based upon balanced separators. We now derive a number of necessary lemmas.

Lemma 12 *Suppose π is a linear ordering of $G = (V, E)$ with*

$$\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$$

Let $0 \leq i < |V|$, and $S = \{v \in V \mid \pi(v) > i\}$ be the set with the $|V| - i$ highest numbered vertices. Then
$$\mathbf{tw}(G) = \max\{TW(V - S), TWR(V - S, S)\}$$

Proof: Recall that $TWR(\emptyset, S) = TW(S)$, for all $S \subseteq V$. By Lemma 7,
$$\mathbf{tw}(G) = TWR(\emptyset, V) \leq \max\{TWR(\emptyset, V - S), TWR(V - S, S)\}$$
Clearly $TW(V - S) \leq TW(V) \leq \mathbf{tw}(G)$. Observing that for $v \in S$: $\pi_{<,v} = V - S \cup \pi_{<,v}$, we have
$$\begin{aligned} TWR(V - S, S) &\leq \max_{v \in S} |Q(V - S \cup \pi_{<,v}, v)| \\ &= \max_{v \in S} R_\pi(v) \leq \mathbf{tw}(G) \end{aligned}$$
□

Lemma 13 *Let $G = (V, E)$ be a graph. Let $S \subseteq V$ be a set of vertices, such that the treewidth of G is equal to the treewidth of the graph $G' = (V, E \cup \{\{v, w\} \mid v, w \in S, v \neq w\})$ obtained from G by turning S into a clique. Then there is a linear ordering $\pi \in \Pi(V, S)$ (i.e., π ends with the vertices in S), such that $\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$.*

Proof: Suppose $H = (V, E_H)$ is a triangulation of G', such that the maximum clique size of H equals $\mathbf{tw}(G') + 1 = \mathbf{tw}(G) + 1$. H is also a triangulation of G, and S is a clique in H. By Lemma 10, there is a perfect elimination scheme π of H that ends with the vertices in S, i.e., with $\pi \in \Pi(V, S)$. For this ordering π, we have that $tw(G) = \max_{v \in V} R_\pi(v)$, as we have for each $v \in V$ that $\{v\} \cup Q(\pi_{<,v}, v)$ is a clique in H, and hence $R_\pi(v) \leq \mathbf{tw}(G)$. □

The following lemma is a small variant on a folklore result. Its proof follows mostly the folklore proof.

Lemma 14 *Let $G = (V, E)$ be a graph with treewidth at most k. There is a set $S \subseteq V$ with*

- *$|S| = k + 1$.*
- *Each connected component of $G[V - S]$ contains at most $(|V| - k)/2$ vertices.*
- *The graph $G' = (V, E \cup \{\{v, w\} \mid v, w \in S, v \neq w\})$ obtained from G by turning S into a clique has treewidth at most k.*

Proof: It is well known that if the treewidth of G is at most k, then G has a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ such that

- For all $i \in I$: $|X_i| = k + 1$.
- For all $(i, j) \in F$: $|X_i - X_j| \leq 1$.

Take such a tree decomposition. Now, for each $i \in I$, consider the trees obtained when removing i from T. For each such tree, consider the union of the sets $X_j - X_i$ with j in this tree. Each connected component W of $G[V - X_i]$ has all its vertices in one such set, i.e., for one subtree of $T - i$. Suppose that for $i \in I$, there is at least one component of $G[V - X_i]$ that contains more than $(|V| - k)/2$ vertices. Let i' be the neighbor of i that belongs to the subtree that contains the vertices in W. Now, direct an arc from i to i'. In this way, each node in I has at most one outgoing arc.

Suppose first that there are two neighboring nodes i_1 and i_2, with i_1 having an arc to i_2, and i_2 having an arc to i_1. Let W_1 be the connected component of $G[V - X_{i_1}]$ that contains more than $(|V| - k)/2$ vertices. Let W_2 be the connected component of $G[V - X_{i_2}]$ that contains more than $(|V| - k)/2$ vertices.

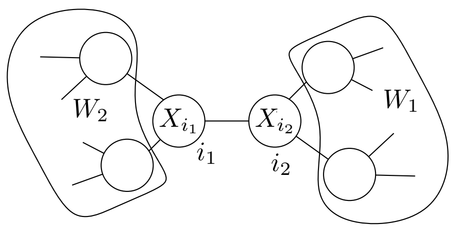

Figure 2: Illustration to the proof of Lemma 14

Now, W_1 and W_2 are disjoint sets. (See Figure 2. Note that if $v \in W_1 \cap W_2$, then v must belong to a bag in the part of the tree, marked with W_1, and a part of the tree marked with W_2. Then also $w \in X_{i_1}$, and this is a contradiction as W is a connected component of $G[V - X_{i_1}]$.)

Also, $W_1 \cap X_{i_1} = \emptyset$. As $W_2 \cap X_{i_2} = \emptyset$, $W_1 \cap X_{i_2} \subseteq X_{i_1} - X_{i_2}$. Now, X_{i_1}, W_1, and $W_2 - (X_{i_1} - X_{i_2})$ are disjoint sets, which contain together at least $(k+1) + ((|V| - k)/2 + 1) + ((|V| - k)/2 + 1 - 1) > |V|$ vertices, contradiction.

Now, as there are no two neighboring nodes i_1 and i_2, with i_1 having an arc to i_2, and i_2 having an arc to i_1, there must be a there is a node i_0 in T without outgoing arcs. (Start at any tree node, and follow arcs. As the tree is finite and loopless, we end in a node i_0 without outgoing arcs.) Now taking $S = X_{i_0}$ gives the required set: $(\{X_i \mid i \in I\}, T = (I, F))$ is also a tree decomposition of G', so G' has treewidth k, and as i_0 has no outgoing arcs, each connected component of $G[V - X_{i_0}]$ has at most $(|V| - k)/2$ vertices. □

Lemma 15 *Let $G = (V, E)$ be a graph with treewidth at most k. Let $k + 1 \leq r \leq n$. There is a set $W \subseteq V$, with*

- *$|W| = r$.*

- *Each connected component of $G[V - W]$ contains at most $(|V| - r + 1)/2$ vertices.*

- $\mathbf{tw}(G) = \max\{TWR(\emptyset, V - W), TWR(V - W, W)\}$.

Proof: First, let S be the set, implied by Lemma 14. Let $\pi \in \Pi(V, S)$ be the linear ordering with $\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$, see Lemma 13. If $k + 1 = r$, then we can take $W = S$, and we are done by Lemma 12.

If $k + 1 < r$, then we construct W and an ordering π as follows. We start by setting $W = S$, but will add later more vertices to W. Repeat the following steps, until $|W| = r$: compute the connected components of $G[V - W]$. Suppose Z is the set of vertices of the connected component of $G[V - W]$ with the largest number of vertices. Let $z \in Z$ be the vertex in Z with the largest index in π: $\pi(z) = \max_{v \in Z} \pi(v)$. Now, we do the following.

- Change the position of z in π as follows: move z to the first position before an element in W, i.e., set $\pi(z) = |V| - |Q|$. All other elements keep their relative position. Now, note that sets $Q(\pi_{<,v}, v)$ do not change, for all $v \in V$. So, for the new ordering π, we still have that $\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$.

- Add z to W. Note that we still have that π ends with the vertices in W.

This procedure keeps as invariants that $\pi \in \Pi(V, W)$, i.e., π ends with W, that $\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$, and that each connected component of $G[V - W]$ contains at most $(|V| - |W| + 1)/2$ vertices. (This can be seen as follows. The component that contained z became one smaller, while the term $(|V| - |W| - 1)/2$ decreases with 1/2. All but the largest component of $G[V - W]$ contain at most $(|V| - |W|)/2$ vertices, which means that they still are of sufficiently small size when $|W|$ increases by one.)

By the Lemma 12, we known that the third condition holds for W, thus the set W obtained by the procedure fulfills the conditions stated in the lemma. □

For a graph $G = (V, E)$, and a set W, let $G^+[W]$ be the fill-in graph, obtained by eliminating the vertices in $V - W$, i.e., $G^+[W] = (W, F)$, with for all $v, w \in W$, $v \neq w$, we have that $\{v, w\} \in F$, if and only if there is a path from v to w that uses only vertices in $V - W$ as internal vertices.

The next lemma formalizes the intuition behind $TWR(V - W, W)$: when computing $TWR(V - W, W)$, we look for the best ordering of the vertices in W, after all vertices in $V - W$ are eliminated — i.e., in the graph $G^+[W]$). We also give a formal proof.

Lemma 16 *Let $G = (V, E)$ be a graph, and $W \subseteq V$ a set of vertices. Then $\mathbf{tw}(G^+[W]) = TWR(V - W, W)$.*

Proof: Consider a linear ordering $\pi \in \Pi(V)$ of the vertices in V. Let π' be the linear ordering of the vertices in W, obtained by restricting π to W, i.e., for $v, w \in W$: $\pi(v) < \pi(w) \Leftrightarrow \pi'(v) < \pi'(w)$. Now, for all $v \in V$

$$Q_{G^+[W]}(\pi'_{<,v}, v) = Q_G(V - W \cup \pi_{<,v}, v)$$

This is because for each edge $\{v, w\}$ in the graph $G^+[W]$, there is a path from v to w using only vertices in $V - W \cup \{v, w\}$.

Suppose that for linear ordering π' of the vertices in W, we have

$$\mathbf{tw}(G^+[W]) = \max_{v \in W} Q_{G^+[W]}(\pi'_{<,v}, v)$$

Take an arbitrary ordering $\pi \in \Pi(V, W)$ such that π' is the ordering obtained by restricting π to W, i.e., we extend π' by placing the vertices of $V - W$ in some arbitrary ordering before all vertices in W. Now

$$\begin{aligned} \mathbf{tw}(G^+[W]) &= \max_{v \in W} Q_{G^+[W]}(\pi'_{<,v}, v) \\ &= \max_{v \in W} Q_G(V - W \cup \pi_{<,v}, v) \\ &\geq TWR(V - W, W) \end{aligned}$$

Suppose that for linear ordering π of the vertices in V, we have

$$TWR(V - W, W) = \max_{v \in W} Q_G(V - W \cup \pi_{<,v}, v)$$

Let π' be the restriction of π to W. Now

$$\begin{aligned} TWR(V - W, W) &= \max_{v \in W} Q_G(V - W \cup \pi_{<,v}, v) \\ &= \max_{v \in W} Q_{G^+[W]}(\pi'_{<,v}, v) \\ &\geq \mathbf{tw}(G^+[W]) \end{aligned}$$

□

Lemma 17 *Let $G = (V, E)$ be a graph, and let $S = S_1 \cup S_2 \subseteq V$. Suppose $S_1 \cap S_2 = \emptyset$, and that there is no edge between a vertex in S_1 and a vertex in S_2. Then $TW(S) = \max\{TW(S_1), TW(S_2)\}$.*

Proof: Suppose for $i \in \{1, 2\}$, $\pi^i \in \Pi(V)$ is a linear ordering of G with

$$TW(S_i) = \max_{v \in S_i} |Q(\pi^i_{<,v}, v)|$$

Let $\pi \in \Pi(V)$ be the linear ordering of G, constructed as follows: first, we take the vertices in S_1 in the order as they appear in π_1, i.e., for all $v, w \in S_1$, $\pi(v) < \pi(w) \Leftrightarrow \pi^1(v) < \pi^1(w)$. Then, we take the vertices in S_2 in the order as they appear in π^2, and then the vertices in $V - S$ in some arbitrary order.

Now, for $i \in \{1, 2\}$, and all vertices $v \in S_i$, we have that

$$Q(\pi^i_{<,v}, v) = Q(\pi_{<,v}, v)$$

and hence

$$\begin{aligned} TW(S) &\leq \max_{v\in S} |Q(\pi_{<,v}, v)| \\ &= \max\left\{\max_{v\in S_1} |Q(\pi_{<,v}, v)|, \max_{v\in S_2} |Q(\pi_{<,v}, v)|\right\} \\ &= \max\left\{\max_{v\in S_1} |Q(\pi^1_{<,v}, v)|, \max_{v\in S_2} |Q(\pi^2_{<,v}, v)|\right\} \\ &= \max\{TW(S_1), TW(S_2)\} \end{aligned}$$

The result follows as clearly, $TW(S) \geq \max\{TW(S_1), TW(S_2)\}$. □

The lemmas above are summarized in the following result, which gives a main idea of the improved recursive algorithm.

Corollary 18 *Let $G = (V, E)$ be a graph, and let k, r be integers, $0 \leq k < r \leq |V|$. The treewidth of G is at most k, if and only if there is a set of vertices $S \subseteq V$, with*

- $|S| = r$.
- *Each connected component of $G[V - S]$ contains at most $(|V| - r + 1)/2$ vertices.*
- *For each connected component W of $G[V - S]$, $TWR(\emptyset, W) \leq k$.*
- *The treewidth of $G^+[S]$ is at most k.*

Proof: Let $G = (V, E)$ be a graph, $0 \leq k < r \leq |V|$.

First, suppose the treewidth of G is at most k. By Lemma 15, there is a set S, with $|S| = r$, each connected component of $G[V-S]$ contains at most $(|V|-r+1)/2$ vertices, and $\mathbf{tw}(G) = \max\{TWR(\emptyset, V - S), TWR(V - S, S)$. By Lemma 16, $\mathbf{tw}(G^+[S]) = TWR(V - S, S) \leq \mathbf{tw}(G) \leq k$. For each connected component W of $G[V - S]$, $TWR(\emptyset, W) \leq TWR(\emptyset, V - S) \leq \mathbf{tw}(G) \leq k$. So, each of the conditions holds for S.

Suppose we have a set $S \subseteq V$, that fulfills each of the four conditions above. For each connected component W of $G[V - S]$, we have $TW(W) = TWR(\emptyset, W) \leq k$. By Lemma 17, $V - S$, which is the disjoint union of these connected components, fulfills $TWR(\emptyset, V - S) = TW(V - S) \leq k$. By Lemma 16, $\mathbf{tw}(G^+[S]) = TWR(V - S, S)$. Now, (cf. Lemma 7), $\mathbf{tw}(G) \leq \max\{TWR(\emptyset, V - S), TWR(V - S, S)\} \leq k$. □

We now present the main result of this section.

Theorem 19 *The treewidth of a graph G on n vertices can be computed in polynomial space and time $O^*(2.9512^n)$.*

Proof: We describe a decision algorithm for treewidth: given a graph G, and an integer k, it decides whether the treewidth of G is at most k. Of course, an algorithm that, given a graph G, computes $\mathbf{tw}(G)$ can be constructed at the cost of an additional multiplicative factor $O(\log n)$, suppressed by the O^*-notation. Let $\gamma = 0.4203$.

Algorithm 4 gives the pseudo-code of the algorithm. It works as follows. If $|V| \leq k+1$, then the treewidth of G is at most $|V|-1 \leq k$, so the algorithm returns true.

Otherwise, the algorithm checks if $k \leq 0.25 \cdot |V|$ or $k \geq \gamma \cdot |V|$. If this is the case, then we search for a set S, as implied by Corollary 18 when we take $r = k+1$. I.e., we enumerate all sets S of size $k+1$. For each such S, we check if all connected components of $G = (V, E)$ have size at most $(|V|-|S|+1)/2$. If so, we use the algorithm of Theorem 8 (Algorithm 2 Recursive-Treewidth) to compute for each connected component W the value $TWR(W, \emptyset)$. If for each such component W, $TWR(\emptyset, W)$, then the algorithm returns true: as $G^+(W)$ has $k+1$ vertices, its treewidth is trivially at most k, and hence all conditions of Corollary 18 are fulfilled, so G has treewidth at most k. If there is no set S of size $k+1$ that yields true, then the algorithm returns false; correctness is implied by Corollary 18.

The remaining case is that $0.25 \cdot |V| < k < \gamma \cdot |V|$. Now, we search for a set S as implied by Corollary 18 when taking $r = \lceil \gamma \cdot |V| \rceil$. I.e., we enumerate all sets S of size $r = \lceil \gamma \cdot |V| \rceil$. For each we check if all connected components W of $G[V-S]$ have size at most $(|V|-r+1)/2$. If so, we use Algorithm 2 Recursive-Treewidth for deciding if all connected components W of $G[V-S]$ fulfill $TWR(\emptyset, W) \leq k$. We also recursively call the algorithm on $G^+(W)$ to decide if this graph has treewidth at most k. If all these checks succeed, the algorithm returns true. If no S of size r made the algorithm return true, the algorithm returns false. Correctness again follows from Corollary 18.

We now analyze the running time of the algorithm. Write $\alpha = k/|V|$.

We start with analyzing the case where $k \leq 0.25 \cdot |V|$ or $\gamma \cdot |V| \leq k$. We have $\alpha \leq 0.25$ or $\alpha \geq \gamma$. The number of subsets of size $\alpha \cdot n$ of a set of size n is known to be of size

$$O^*((\alpha^{-\alpha} \cdot (1-\alpha)^{\alpha-1})^n)$$

Each connected component W of $G[V-S]$ for which the algorithm calls Recursive-Treewidth has size at most $(|V|-\alpha \cdot |V|+1)/2$, thus we use at most $O^*(4^{(|V|-\alpha\cdot|V|+1)/2}) = O^*(2^{(1-\alpha)|V|}$ time for one such component. Thus, the total time in this case is bounded by

$$O^*\left((\alpha^{-\alpha} \cdot (1-\alpha)^{\alpha-1} \cdot 2^{1-\alpha}\right)^n$$

Write $f(\alpha) = \alpha^{-\alpha} \cdot (1-\alpha)^{\alpha-1} \cdot 2^{1-\alpha}$. The function f monotonically increases in the interval $(0, \frac{1}{3})$, and monotonically decreases in the interval $(\frac{1}{3}, 1)$. As $f(0.25) < 2.9512$, and $f(\gamma) < 2.9512$, we have for all α with $0 < \alpha \leq 0.25$ or $\gamma \leq \alpha < 1$, that $f(\alpha) < 2.9512$, and hence that the algorithm uses $O^*(2.9512^n)$ time.

We now look at the case where $0.25 \cdot |V| < k < \gamma \cdot |V|$, i.e., where $0.25 < \alpha < \gamma$. As in the previous case, the time for all computations of $TWR(\emptyset, W)$ for all connected components of $G[V-S]$ over all sets $S \subseteq V$ of size r is bounded by

$$O^*((\gamma^{-\gamma} \cdot (1-\gamma)^{\gamma-1} \cdot 2^{1-\gamma})^n = O^*(f(\gamma)^n)$$

Algorithm 4 Improved-Recursive-Treewidth(Graph $G = (V, E)$, Integer k)

```
if $|V| \leq k + 1$ then
  return true
else if $k \leq 0.25 \cdot |V|$ or $k \geq 0.4203 \cdot |V|$ then
  for all sets $S \subseteq V$ of size $k + 1$ do
    if each connected component of $G[V - S]$ contains at most $(|V| - |S| + 1)/2$ vertices
    then
      tbool = true;
      for all connected components $W$ of $G[V - S]$ do
        tbool = tbool or (Recursive-Treewidth($G$, $\emptyset$, $W$) $\leq k$;
      end for
      if tbool then
        return true
      end if
    end if
  end for
else
  for all sets $S \subseteq V$ of size $\lceil 0.4203 \cdot |V| \rceil$ do
    if Each connected component of $G[V - S]$ contains at most $(|V| - |S| + 1)/2$ vertices
    then
      Compute the graph $G^+[S]$.
      tbool = Improved-Recursive-Treewidth($G^+[S]$, $k$);
      for all connected components $W$ of $G[V - S]$ do
        tbool = tbool or Recursive-Treewidth($G$, $\emptyset$, $W$) $\leq k$;
      end for
      if tbool then
        return true
      end if
    end if
  end for
end if
return false
```

As $f(\gamma) = f(0.4203) < 2.9512$, the total time for these steps is bounded by $O^*(2.9512^n)$.

We have to add to this time the total time over all recursive calls to the algorithm with graphs of the form $G^+(S)$. Note that the recursion depth is at most 1: in the recursion, the value of k is unchanged, while we now have a graph with $|S| = \lceil \gamma \cdot |V| \rceil$ vertices. So, in the recursive call, $k > 0.25 \cdot |V| > \gamma \cdot |S|$, and the algorithm executes the first case. From the analysis above, it follows that each recursive call of Improved-Recursive-Treewidth on a graph $G^+[S]$ costs

$$O^*(\left(\beta^{-\beta} \cdot (1-\beta)^{\beta-1} \cdot 2^{1-\beta}\right)^{\gamma \cdot n})$$

time, with $\beta = \alpha/\gamma$. (Note that $\frac{k}{|S|} \sim \frac{\alpha|V|}{\gamma|V|} = \beta$.) Write

$$g(\alpha) = \gamma^\gamma \cdot (1-\gamma)^{\gamma-1} \cdot \left(\left(\frac{\alpha}{\gamma}\right)^{-\frac{\alpha}{\gamma}} \cdot \left(1 - \frac{\alpha}{\gamma}\right)^{\frac{\alpha}{\gamma}-1} \cdot 2^{1-\frac{\alpha}{\gamma}} \right)^\gamma$$

As there are $O^*((\gamma^\gamma \cdot (1-\gamma)^{\gamma-1})^n)$ vertex sets $S \subseteq V$ of size γn, the total time of all calls of Improved-Recursive-Treewidth with graphs of the form $G^+[S]$ is bounded by $O^*(g(\alpha)^n)$. On the interval $[0.25, \gamma]$, the function g is monotonically decreasing, with $g(0.25) < 2.9511$. Thus, it follows that the total time over all calls of Improved-Recursive-Treewidth for graphs $G^+[S]$ is bounded by $O^*(2.9511^n)$, and the total time of the algorithm is bounded by $O^*(2.9512^n)$. □

We conjecture that with a more detailed analysis and more levels of recursion, small improvements to the running time are possible.

7 Other problems

In this section, we discuss a number of other problems that are related to treewidth, and will show that in several cases, algorithms with running time $O^*(2^n)$ and space $O^*(2^n)$, and with running time $O^*(4^n)$ and polynomial space are also possible, using techniques very similar to the proofs of Theorem 6 and Theorem 8.

Theorem 20 *Let f be a function, mapping 3-tuples, consisting of a graph $G = (V, E)$, a vertex set $S \subseteq V$, and a vertex $v \in V$ to an integer. Then we can compute in $O^*(2^n)$ time and $O^*(2^n)$ space, or in $O^*(4^n)$ time and polynomial space the following values for a given graph $G = (V, E)$:*

$$\min_{\pi \in \Pi(V)} \max_{v \in V} f(G, \pi_{<,v}, v)$$

or

$$\min_{\pi \in \Pi(V)} \sum_{v \in V} f(G, \pi_{<,v}, v)$$

We omit the proof of this theorem here: it is a more or less straightforward generalization of the proofs of Theorem 6 and Theorem 8. For many problems, we can see that they can be written in the form of Theorem 20. We discuss several of these below.

A good overview paper, discussion several linear ordering problems is [10]. Relations between treewidth, pathwidth, and other parameters can be found in [3].

Minimum fill-in A problem, related to treewidth, is the MINIMUM FILL-IN problem. Exact algorithms for MINIMUM FILL-IN were obtained by Fomin et al. [12] and Villanger [25]: the same time bounds ($O(1.9601^n)$ time and $O(1.8899^n)$ time) are given as for treewidth; these algorithms use the same techniques as for treewidth; they use exponential space. The MINIMUM FILL-IN problem has important applications in Gaussian elimination.

The *minimum fill-in* of a graph $G = (V, E)$, $\mathbf{mfi}(G)$, is the minimum over all triangulations $H = (V, E_H)$ of G of $|E_H - E|$, i.e., the minimum number of edges that, when added to G, make G chordal.

The following proposition is a variant of Proposition 3, but now targeted at the minimum fill-in problem.

Proposition 21 *Let $G = (V, E)$ be a graph, and k a non-negative integer. The minimum fill-in of G is at most k iff there is a linear ordering π of G, such that*

$$\sum_{v \in V} R_\pi(v) \leq k + |V|$$

This shows that Theorem 20 can be applied. It is probably harder to obtain a practical version from the $O^*(2^n)$-algorithm: as we sum here costs, working with an upper bound probably causes much fewer dropped entries from tables. This seems an interesting topic for an experimental study.

The techniques of Section 6 cannot directly be used to improve upon the polynomial space bound. We leave it as an open problem to improve upon the $O^(4^n)$ bound for polynomial space algorithms for MINIMUM FILL-IN; possibly a variant of the techniques of Section 6 or the notion of potential maximal clique can be of use here.

Pathwidth The pathwidth of a graph is usually defined in terms of path decompositions, but it has several equivalent characterizations, see e.g., [3] for an overview. Kinnersley [19] showed that pathwidth can be defined as a vertex ordering problem. We use this characterization to obtain the exact algorithms.

Definition 22 *The* vertex separation number *of a linear ordering π of $G = (V, E)$ is*

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

The vertex separation number *of a graph G is the minimum vertex separation number over all linear orderings of G.*

Theorem 23 (Kinnersley [19]) *The vertex separation number of a graph equals its pathwidth.*

It is not hard to see that the VERTEX SEPARATION NUMBER problem has the shape of Theorem 20.

Sum Cut Replacing taking a maximum by taking a summation in the definition of vertex separation gives us the Sum Cut problem. See for instance [10]. In the Sum Cut problem, we look for a linear ordering π which minimizes

$$\sum_{v \in V} |\{w \in V \mid \exists x \in V : \{w, x\} \in E \wedge \pi(w) < \pi(v) \leq \pi(x)\}|$$

Minimum Interval Graph Completion Another problem, related to Pathwidth, which can be solved with Theorem 20 is the Minimum Interval Graph Completion problem. The Minimum Interval Graph Completion problem is the following: given a graph $G = (V, E)$, what is the minimum size of a set of edges, that, when added to G yield an interval graph. In a similar way as pathwidth can be reformulated as vertex separation number, this problem can be formulated as linear ordering problem too.

Lemma 24 *Let $G = (V, E)$ be a graph. There is an interval graph $H = (V, F)$ that contains G as a subgraph ($E \subseteq F$) with $|F| \leq k$, if and only if there is a linear ordering π of G with*

$$|\{v, w\} \mid v \neq w \wedge x \in V : \{w, x\} \in E \wedge \pi(w) < \pi(v) \leq \pi(x)\}| \leq k$$

Cutwidth and variants The *cutwidth* of a linear ordering π of a graph $G = (V, E)$ is

$$\max_{v \in V} |\{w, x\} \in E \wedge \pi(w) \leq \pi(v) < \pi(x)\}|$$

The *modified cutwidth* of a linear ordering π of a graph $G = (V, E)$ is

$$\max_{v \in V} |\{w, x\} \in E \wedge \pi(w) < \pi(v) < \pi(x)\}|$$

The cutwidth (modified cutwidth) of a graph is the minimum cutwidth (modified cutwidth) of a linear ordering of it. The parameters have variants for directed acyclic graphs. The cutwidth (modified cutwidth) of a directed acyclic graph $G = (V, A)$ is the minimum cutwidth (modified cutwidth) of a topological ordering of G; the latter are defined similar to the undirected counterparts. By setting $f(G, S, v)$ to a very high value when there is an arc $(v, w) \in A$ with $w \in S$, we can force that the minimum is taken at a topological sort, and thus fit the problem into the form of Theorem 20.

Optimal Linear Arrangement The Optimal Linear Arrangement problem, of which the decision variant was proved NP-complete in [14], asks for a given graph $G = (V, E)$ for the minimum over all linear orderings π of $\sum_{\{v,w\} \in E} |\pi(v) - \pi(w)|$. The following simple lemma shows that we can write the problem again in a form such that Theorem 20 applies.

Lemma 25 *For each graph $G = (V, E)$, and for each linear ordering π of G,*

$$\sum_{\{v,w\} \in E} |\pi(v) - \pi(w)| = \sum_{v \in V} |\{\{x, y\} \in E \mid \pi(x) \leq \pi(v) < \pi(w)\}|$$

The directed variant, where we look for topological orderings π of $G = (V, A)$ with $\sum_{(v,w)\in A}(f(w) - f(v))$ can be handled in a similar way.

Directed Feedback Arc Set The Directed Feedback Arc Set is the following: given a directed graph $G = (V, A)$, find a set of arcs $F \subseteq A$ with $|F|$ as small as possible, such that $(V, A - F)$ is acyclic, i.e., each cycle in G contains at least one arc in F. It is a variant of the well known Feedback Vertex Set and Directed Feedback Vertex Set problems (which look for a set of vertices that break all cycles). (The problem to find in an undirected graph a minimum size set of edges that breaks all cycles is trivial; its weighted variant is a reformulation of the polynomial time solvable Minimum Spanning Tree problem. The (Directed) Feedback Vertex Set problems are trivially solvable in $O^*(2^n)$ time with linear space, and thus we have to focus only to Directed Feedback Arc Set.) One can also look to a weighted variant: each arc has a weight, and we look for a set of arcs that break all cycles of minimum total weight.

The following lemma shows that we can formulate (Weighted) Directed Feedback Arc Set in a shape such that Theorem 20 can be applied. Recall that a graph is acyclic, if and only if it has a topological ordering.

Lemma 26 *Let $G = (V, A)$ be a directed graph, and let $w : A \to \mathbf{N}$ be a function that assigns each arc a non-negative integer weight. Let $K \in \mathbf{N}$ be an integer. There exists a set of arcs $F \subseteq A$ with $(V, A - F)$ acyclic and $\sum_{a\in F} w(a) \leq K$, if and only if there is a linear ordering π of G, such that $\sum_{(x,y)\in A,\ \pi(x)>\pi(y)} w((x, y)) \leq K$.*

Summary The following result summarizes the discussion in the paragraphs above.

Theorem 27 *Each of the following problems:* Minimum Fill-In, Pathwidth, Sum Cut, Minimum Interval Graph Completion, Cutwidth, Directed Cutwidth, Modified Cutwidth, Directed Modified Cutwidth, Optimal Linear Arrangement, Directed Optimal Linear Arrangement *and* Directed Feedback Arc Set

1. *can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space.*
2. *can be solved in $O^*(4^n)$ time and polynomial space.*

In each case, the $O^*(2^n)$ algorithm resembles the classic Held-Karp algorithm for TSP [18], and the $O^*(4^n)$ its variant by Gurevich and Shelah [17].

8 Conclusions

In this paper, we have given dynamic programming and recursive algorithms to compute the treewidth of a given graph. The dynamic programming algorithm for the treewidth problem has been implemented; for small instances (slightly below 50 vertices), the algorithm appears to be practical. Also, it can be used to obtain better lower bounds (by running the algorithm on a minor of the input graph), or upper bounds (by dropping table entries when space or time does not permit us to compute the full tables). On a more theoretical side, we gave the first exponential time algorithms for TREEWIDTH with a running time of the type $O^*(c^n)$ for some constant c that use polynomial space and we reduced the running time of the algorithm with polynomial space to $O^*(2.9512^n)$.

A comparison of the dynamic programming algorithm for TREEWIDTH with other algorithms, (e.g., a branch and bound algorithm as in [15] or the algorithm of Shoikhet and Geiger [22]) would be very interesting.

We also have seen that several problems that can be formulated as a 'linear ordering problem' can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space, or $O^*(4^n)$ time and polynomial space. There are here several angles for interesting further work. It would be interesting to carry out experiments with (variants of) the dynamic programming algorithm for several of these problems. On the more theoretical side, it is interesting to try to improve the time bounds. Some problems appear to be hard, e.g., to improve upon the $O^*(2^n)$ time for PATHWIDTH.

References

[1] 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.

[2] E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 217–227. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.

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

[4] H. L. Bodlaender. Discovering treewidth. In P. Vojtáš, M. Bieliková, and B. Charron-Bost, editors, *SOFSEM 2005: Theory and Practive of Computer Science: 31st Conference on Current Trends in Theory and Practive of Computer Science*, pages 1–16. Springer-Verlag, Lecture Notes in Computer Science 3381, 2005.

[5] 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.

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

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

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

[9] N. D. Dendris, L. M. Kirousis, and D. M. Thilikos. Fugitive-search games on graphs and related parameters. *Theor. Comp. Sc.*, 172:233–254, 1997.

[10] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

[11] F. V. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.

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

[13] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

[14] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comp. Sc.*, 1:237–267, 1976.

[15] V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence UAI-04*, pages 201–208, Arlington, Virginia, USA, 2004. AUAI Press.

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

[17] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16:486–502, 1987.

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

[19] N. G. Kinnersley. The vertex separation number of a graph equals its path width. *Information Processing Letters*, 42:345–350, 1992.

[20] A. M. C. A. Koster, T. Wolle, and H. L. Bodlaender. Degree-based treewidth lower bounds. In S. E. Nikoletseas, editor, *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms WEA 2005*, pages 101–112. Springer Verlag, Lecture Notes in Computer Science, vol. 3503, 2005.

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

[22] 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.

[23] 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.

[24] Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib, 2004.

[25] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In *Proceedings of the 7th Latin American Theoretical Informatics Symposium (LATIN 2006)*, volume 3887 of *LNCS*, pages 800–811. Springer-Verlag, Berlin, 2006.

[26] 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.