On Exact Algorithms for Treewidth

HANS L. BODLAENDER, Utrecht University, the Netherlands
FEDOR V. FOMIN, University of Bergen, Norway
ARIE M. C. A. KOSTER, Aachen University, Germany
DIETER KRATSCH, Université de Metz, France
DIMITRIOS M. THILIKOS, University of Athens, Greece

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. We also consider the problem of computing TREEWIDTH under the restriction that the space used is only polynomial and 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.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*; G.2.2 [**Discrete Mathematics**]: Graph Theory

General Terms: Algorithms, Design, Theory, Experimentation

Additional Key Words and Phrases: Graph algorithms, exact algorithms, treewidth, dynamic programming, separators

ACM Reference Format:
Bodlaender, H. L., Fomin, F. V., Kratsch, D., Koster, A. M. C. A., and Thilikos, D. M. 2012. On exact algorithms for treewidth. ACM Trans. Algor. 9, 1, Article 12 (December 2012), 23 pages.
DOI = 10.1145/2390176.2390188 http://doi.acm.org/10.1145/2390176.2390188

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 D. M. Thilikos was supported by the Spanish CICYT project TIN-2004-07925 (GRAMMARS) and the project "Kapodistrias" (AII 02839/28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/4/8757).
Authors' addresses: H. L. Bodlaender, Department of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands; email: h.l.bodlaender@uu.nl; F. V. Fomin, Department of Informatics, University of Bergen, N-5020 Bergen, Norway; email: fomin@ii.uib.no; A. M. C. A. Koster, Lehrstuhl II für Mathematik, RWTH Aachen University, Wüllnerstr. zwischen 5 und 7, D-52062 Aachen, Germany; email: koster@math2.rwth-aachen.de; D. Kratsch LITA, Université de Metz, F-507045 Metz Cedex 01, France; email: kratsch@sciences.univ-metz.fr; D. M. Thilikos, Department of Mathematics, National and Kapodistrian University of Athens, Panepistimioupolis, GR-15784, Athens, Greece; email: sedthilk@math.uoa.gr.

DOI 10.1145/2390176.2390188 http://doi.acm.org/10.1145/2390176.2390188

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, a large number of papers appeared, studying the problem to determine the treewidth of a graph, including both theoretical and experimental results. See, for example, Bodlaender [2005] and Bodlaender and Koster [2010, 2011] for overviews. Since the problem is NP complete [Arnborg et al. 1987], there is a little hope in finding an algorithm that 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 [Fomin et al. 2005; Woeginger 2003] for an introduction to the area of exponential algorithms.) Arnborg et al. [1987] 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 and also uses $O^*(2^n)$ memory.[1] See also Shoikhet and Geiger [1997]. Fomin et al. [2004] 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 [2002, 2001]. In a number of steps (see [Villanger 2006; Fomin et al. 2008; Fomin and Villanger 2012, 2010]) the running time of the algorithm was improved. The currently best-known bound for the running time is $O(1.7347^n)$, due to Fomin and Villanger [2010]. The space required for the algorithms from Fomin et al. [2004, 2008], Villanger [2006], Fomin and Villanger [2012, 2010] is in each case bounded by a polynomial in n times the number of potential maximal cliques, that is, $O(1.7347^n)$ by the analysis by Fomin and Villanger [2010]. While these algorithms provide the best-known running times, they are based on computations of potential maximal cliques and may be difficult to implement.

Our article contains results of two different types. Our first result is an exact algorithm for treewidth whose running time is larger than the best-known asymptotic bounds but which seems to be much more suitable for implementations. Our second result is of theoretical nature, giving an exact algorithm with polynomial memory.

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, for example, Bachoore and Bodlaender [2005], Clautiaux et al. [2004], and Gogate and Dechter [2004]. In this article, we exploit this characterization again, and a lesser known property of it. Thus, we can show that an old dynamic programming method, introduced by Held and Karp [1962] for the TRAVELING SALESMAN problem 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 [1962] 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 article: 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 article, we answer this question in the affirmative. We show that there is an algorithm to compute the treewidth that uses

[1] We sometimes use O^*-notation which is a modified O-notation suppressing all polynomially bounded factors, introduced by Woeginger [2003].

$O^*(4^n)$ 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 [1987] for HAMILTONIAN PATH.

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. Using similar methods, but a better method of enumerating potential maximal cliques, our result was recently improved to $O(2.6151^n)$ time by Fomin and Villanger [2012].

It should be noted that, while the research on the polynomial-space algorithms was initiated by observations from our experiments, the results are only of theoretical interest. Both polynomial-space algorithms have to consider a very large number of subsets of a specific size of the set of vertices, and it cannot be expected that these algorithms are practical when our exponential space algorithm uses too much space.

2. PRELIMINARIES

2.1. Definitions

We assume the reader to be familiar with standard notions from graph theory. Throughout this article, $n = |V|$ 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, that is, there is an edge connecting two nonconsecutive 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 2.1. 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, for example, Bodlaender [1998].

PROPOSITION 2.2. *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, for example, Bachoore and Bodlaender [2005], Clautiaux et al. [2004], Dendris et al. [1997], and Gogate and Dechter [2004].

A *linear ordering* of a graph $G = (V, E)$ is a bijection $\pi : V \to \{1, 2, \ldots, |V|\}$. 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, that is, 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 Golumbic [1980, 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 nonadjacent 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, that is, $R_\pi(v) = |\{w \in V \mid \pi(w) > \pi(v) \wedge P_\pi(v, w)\}|$.

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. For disjoint sets S and R, we write $\Pi(S, R)$ for the collection of permutations of $S \cup R$ that end with vertices in R, that is, each such permutation starts with the vertices in S in some ordered, and then ends with the vertices in R in some order.

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 $\pi \in \Pi(V)$.

The proof of the following proposition is an immediate consequence of a lemma of Rose et al. [1976]. (See also Bodlaender [1998], Clautiaux et al. [2004], and Dendris et al. [1997].)

PROPOSITION 2.3. *Let $G = (V, E)$ be a graph. The treewidth of G equals* $\min_{\pi \in \Pi(V)} \max_{v \in V} R_\pi(v)$.

PROOF. We use the following result from Rose et al. [1976]. 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 [1965] 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 $|\{v, w\} \in E_H \mid \pi(w) > \pi(v)\}|$. Suppose the treewidth of G equals k, for some nonnegative integer k.

G has a triangulation $H = (V, E_H)$ with maximum clique size $k+1$ (Proposition 2.2). For a perfect elimination scheme π of H, we have:

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

Let $\pi' \in \Pi(V)$. There is a chordal graph $H' = (V, E_{H'})$ for which π' is a perfect elimination ordering. Let C be a maximum clique in H. By Proposition 2.2, $|C| \geq k+1$. The vertex in C with minimum $\pi(v)$ has $C \subseteq \{v\} \cup \{\{v, w\} \in E_{H'} \mid \pi(w) > \pi(v)\}$, so $R_\pi(v) \geq k$. The result now follows. □

3. EXACT ALGORITHMS FOR TREEWIDTH

3.1. An $O^*(2^n)$ Time and $O^*(2^n)$ Space Dynamic Programming Algorithm

In this section, we develop a theoretical exact algorithm for TREEWIDTH that uses $O^*(2^n)$ time and $O^*(2^n)$ space. A more practical implementation is discussed in Section 4.

We define for $\emptyset \neq S \subseteq V$

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

Let $TW_G(\emptyset) = -\infty$. 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 3.1. *For each graph $G = (V, E)$, the treewidth of G equals $TW(V)$.*

PROOF. Using Proposition 2.3, we have

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

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

LEMMA 3.2. *For any graph $G = (V, E)$, and any set of vertices $S \subseteq V$, $S \neq \emptyset$,*

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

PROOF. First, suppose $|S| = 1$. As $TW(S - \{v\}) = -\infty$, and $\pi_{<,v} = S - \{v\} = \emptyset$, the result directly follows.

Suppose $|S| > 1$. Let $\pi \in \Pi(S)$ 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, that is, the vertex with $S \subseteq \pi_{\leq,v}$. Let $\pi' \in \Pi(S - \{v\})$ be obtained by removing v from π. Note that, for all $w \in S - \{v\}$, $\pi_{<,w} = \pi'_{<,w}$).

As $S \subseteq \pi_{\leq,v}$, we have $|Q(S - \{v\}, v)| \leq |Q(\pi_{<,v}, v)|$. Now

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

Thus,

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

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

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

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

Theorem 3.3. *The treewidth of a graph on n vertices can be determined in $O^*(2^n)$ time and $O^*(2^n)$ space.*

Proof. By Lemma 3.2, 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 3.2. Here, we give pseudocode for a simple form of the algorithm Dynamic-Programming-Treewidth.

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

```
Set TW(∅) = −∞.
for i = 1 to n do
  for all sets S ⊂ V with |S| = i do
    Set TW(S) = min_{v∈S} max {TW(S − {v}), |Q(S − {v}, v)|}
  end for
end for
return TW(V)
```

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. $\square$

In Section 4, we report on an implementation of the algorithm for Treewidth (with additional improvements to decrease the time for actual instances). Since the $O^*(2^n)$ space requirement turns out to be a real limitation, we next consider polynomial-space

(but exponential-time) algorithms. In the pseudocode of Algorithm 1, we stored a value for all sets S. In a practical computation, this is not necessary: when computing values $TW(S)$ for sets S of size i, we only need these values for sets of size $i-1$, and thus can delete the information for sets S with $|S| < i-1$.

3.2. An $O^*(4^n)$ Time and Polynomial Space Recursive Algorithm for Treewidth

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

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

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 3.4. *Let $G = (V, E)$ be a graph. Let $S \subseteq V$, $L \subseteq V$, and $L \cap S = \emptyset$.*

(1) *Suppose $|S| = 1$. Then $TWR(L, S) = |Q(L, v)|$.*
(2) *Suppose $|S| \geq 2$, and $1 \leq k < |S|$. Then*

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

PROOF. First, suppose $S = \{v\}$. For each $\pi \in \Pi(L, S)$, $L \cup \pi_{<,v} = L$, and thus part (1) of the lemma follows.

Suppose $|S| \geq 2$ and $1 \leq k < |S|$. Suppose $\pi \in \Pi(L, S)$ 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, that is, all vertices in $S - S'$ have a higher index in π than any element in S' and $|S'| = k$. Now,

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

For the other direction, suppose that $S' \subseteq S$ with $|S'| = k$ fulfills

$$\begin{aligned}
&\max \left\{ TWR(L, S'), TWR(L \cup S', S - S') \right\} \\
&\quad = \min_{S'' \subseteq S, |S'|=k} \max \left\{ TWR(L, S''), TWR(L \cup S', S - S'') \right\}.
\end{aligned}$$

Let $\pi' \in \Pi(L \cup S')$ be a permutation with $TWR(L, S') = \max_{v \in S'} |Q(L \cup \pi'_{<,v}, v)|$. Let $\pi'' \in \Pi(L \cup S)$ 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 $\pi \in \Pi(L \cup S)$ 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'. That is, 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''. That is, 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. For this order π, we have

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

This proves the result. □

By making use of Lemma 3.4 with $k = \lfloor |S|/2 \rfloor$, we obtain the following result.

THEOREM 3.5. *The treewidth of a graph on n vertices can be determined in* $O^*(4^n)$ *time and polynomial space.*

PROOF. Lemma 3.4 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.

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 = ∞.
for all sets S' ⊆ S, |S'| = ⌊|S|/2⌋ do
  Compute v1 = Recursive-Treewidth(G, L, S');
  Compute v2 = Recursive-Treewidth(G, L ∪ S', S − S');
  Set Opt = min {Opt, max {v1, v2}};
end for
return Opt
```

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

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

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

$$T(n, r) \leq 4^r \cdot p'(n). \tag{2}$$

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

Compared to the exponential-space algorithm, the running time of this algorithm is clearly worse. In the next section, we improve upon this without losing the polynomial-space usage.

3.3. An $O^*(2.9512^n)$ Time and Polynomial Space Algorithm

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 3.6. *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

$$\mathbf{tw}(G) = \max\left\{TW(V - S), TWR(V - S, S)\right\}.$$

PROOF. Recall that $TWR(\emptyset, S) = TW(S)$, for all $S \subseteq V$. By Lemma 3.4,

$$\mathbf{tw}(G) = TWR(\emptyset, V) \leq \max\left\{TWR(\emptyset, V - S), TWR(V - S, S)\right\}.$$

We have that $TW(V - S) \leq \max_{v \in V - S} R_\pi(v) \leq \mathbf{tw}(G)$. Observing that for $v \in S$: $V - S \subseteq \pi_{<,v}$, we have

$$\begin{aligned} TWR(V - S, S) &\leq \max_{v \in S} |Q(V - S \cup \pi_{<,v}, v)| \\ &= \max_{v \in S} R_\pi(v) \leq \mathbf{tw}(G). \quad \square \end{aligned}$$

LEMMA 3.7. *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 $\pi \in \Pi(V - S, S)$, 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 4.2, there is a perfect elimination scheme π of H that ends with the vertices in S, that is, with $\pi \in \Pi(V - S, S)$. 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 $R_\pi(v) \leq \mathbf{tw}(G)$. $\square$

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

LEMMA 3.8. *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 $G[W]$ of $G[V - X_i]$ has all its vertices in one

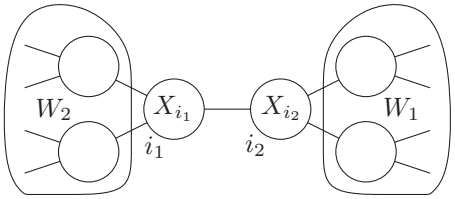

Fig. 1. Illustration to the proof of Lemma 3.8.

such set, that is, 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 $G[W_1]$ be the connected component of $G[V - X_{i_1}]$ that contains more than $(|V| - k)/2$ vertices. Let $G[W_2]$ be the connected component of $G[V - X_{i_2}]$ that contains more than $(|V| - k)/2$ vertices.

Now, W_1 and W_2 are disjoint sets. (See Figure 1. 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 $G[W]$ 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 3.9. *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 3.8. Let $\pi \in \Pi(V - S, S)$ be the linear ordering with $\mathbf{tw}(G) = \max_{v \in V} R_\pi(v)$, see Lemma 3.7. If $k + 1 = r$, then we can take $W = S$, and we are done by Lemma 3.6.

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, that is, 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 $\pi \in \Pi(V - W, W)$, that is, 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 3.6, we know 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 $V - W$, that is, $G^+[W] = (W, F)$, 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. That is, the fill-in graph has all edges that appear in any triangulation obtained by an elimination scheme that starts with W.

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—that is, in the graph $G^+[W]$). We also give a formal proof.

LEMMA 3.10. *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, that is, 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 $\pi \in \Pi(V - W, W)$ such that π' is the ordering obtained by restricting π to W, that is, 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

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

LEMMA 3.11. *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 $\pi \in \Pi(S)$ be the permutation of S, constructed as follows: we take the vertices in S_1 in the order as they appear in π_1, that is, 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.

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

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

Consider a permutation $\pi \in \Pi(S)$ with $TW(S) = \max_{v \in S} |Q(\pi_{<,v}, v)|$. Let $\pi^1 \in \Pi(S_1)$ be obtained by restricting π to S_1, that is, for all $v, w \in S_1$, v is before w in π if and only if v is before w in π^1. Again, $Q(\pi^1_{<,v}, v) = Q(\pi_{<,v}, v)$, and it follows that $TW(S) \geq TW(S_1)$. Similarly, $TW(S) \geq TW(S_2)$. □

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

COROLLARY 3.12. *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 $G[W]$ 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 3.9, 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 3.10, $\mathbf{tw}(G^+[S]) = TWR(V - S, S) \leq \mathbf{tw}(G) \leq k$. For each connected component $G[W]$ 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 these four conditions. For each connected component $G[W]$ of $G[V - S]$, we have $TW(W) = TWR(\emptyset, W) \leq k$. By Lemma 3.11, $V - S$, which is the disjoint union of the vertex sets of these connected components, fulfills $TWR(\emptyset, V - S) = TW(V - S) \leq k$. By Lemma 3.10, $\mathbf{tw}(G^+[S]) = TWR(V - S, S)$. Now, (cf., Lemma 3.4), $\mathbf{tw}(G) \leq \max\{TWR(\emptyset, V - S), TWR(V - S, S)\} \leq k$. □

We now present the main result of this section.

THEOREM 3.13. *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 3 gives the pseudocode 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.

ALGORITHM 3: Improved-Recursive-Treewidth(Graph $G = (V, E)$, Integer k)

```
if |V| ≤ k + 1 then
  return true
else if k ≤ 0.25 · |V| or k ≥ 0.4203 · |V| then
  for all sets S ⊆ 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 G[W] of G[V − S] do
        tbool = tbool or (Recursive-Treewidth(G, ∅, W) ≤ k);
      end for
      if tbool then
        return true
      end if
    end if
  end for
else
  for all sets S ⊆ V of size ⌈0.4203 · |V|⌉ 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 G[W] of G[V − S] do
        tbool = tbool or Recursive-Treewidth(G, ∅, W) ≤ k;
      end for
      if tbool then
        return true
      end if
    end if
  end for
end if
return false
```

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 3.12 when we take $r = k + 1$. That is, 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 3.5 (Algorithm 2 Recursive-Treewidth) to compute for each connected component $G[W]$ 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 3.12 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 3.12.

The remaining case is that $0.25 \cdot |V| < k < \gamma \cdot |V|$. Now, we search for a set S as implied by Corollary 3.12 when taking $r = \lceil \gamma \cdot |V| \rceil$. That is, we enumerate all sets S of size $r = \lceil \gamma \cdot |V| \rceil$. For each, we check if all connected components $G[W]$ 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 $G[W]$ 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 3.12.

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 $G[W]$ 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

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

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 $0.25 \cdot |V| < k < \gamma \cdot |V|$, that is, 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

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

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 this analysis, 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 $O^*(2.9512^n)$. □

Recently, the bound has been improved to $O(2.6151^n)$ time by Fomin and Villanger [2012]. This improvement is mainly obtained by a better method to enumerate and bound the number of potential maximal cliques.

4. EXPERIMENTAL RESULTS AND ALGORITHM ENGINEERING

In this section, we discuss experiments with the algorithms proposed in the previous section. We first discuss algorithm engineering aspects of our exponential-space

algorithms: small modifications and improvements turn the algorithm into an algorithm that is practical for small sized graphs. We then report on results of experiments with implementations of the modified algorithm with optimizations. Finally, we briefly discuss the polynomial-space algorithms.

4.1. Exponential Space and Time Implementation

4.1.1. Algorithm Engineering. For practical considerations, we use a scheme that is slightly different than that of Theorem 3.3. 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 $r' = \max\{r, |Q(S, x)|\}$. Recall that $Q(S, x)$ is the set of vertices in $V - S - \{x\}$ that are reachable from x with paths using vertices in $S \cup \{x\}$. A simple variant of depth first search can be used for this. 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.

Starting with a good upper bound up is beneficial for the running time, as fewer table entries have to be handled. See also Section 4.2.

The scheme is shown in Algorithm 4. In our implementation, we use two additional optimizations that appeared to give significant savings in time and space consumption. Both of these are shown in the code of Algorithm 4. The first of these is based upon Lemma 4.1.

ALGORITHM 4: Algorithm TWDP (Graph $G = (V, E)$, clique $C \subseteq V$)

```
n = |V|.
Compute some initial upper bound up on the treewidth of G. (For example, set up = n − 1.)
Let TW_0 be the set, containing the pair (∅, −∞).
for i = 1 to n − |C| do
  Set TW_i to be an empty set.
  for each pair (S, r) in TW_{i−1} do
    for each vertex x ∈ V − S do
      Compute q = |Q(S, v)|.
      Set r' = min{r, q}.
      if r' < up then
        up = min{up, n − |S| − 1}
        if There is a pair (S ∪ {x}, t) in TW_i for some t then
          Replace the pair (S ∪ {x}, t) in TW_i by (S ∪ {x}, min(t, r')).
        else
          Insert the pair (S ∪ {x}, r') in TW_i.
        end if
      end if
    end for
  end for
end for
if TW_{n−|C|} contains a pair (V − C, r) for some r then
  return r
else
  return up
end if
```

LEMMA 4.1. *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 we claim that

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

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

Lemma 4.1 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 4.3. We use the following basic fact on chordal graphs and cliques.

LEMMA 4.2. *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, that is, such that for each $v \in C$: $\pi(v) \geq |V| - |C| + 1$.*

PROOF. Consider (for instance) the Maximum Cardinality Search (MCS) algorithm from Tarjan and Yannakakis [1984]. 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 4.3. *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 2.3 and Lemma 4.2, we obtain that the treewidth of G is at most some nonnegative integer k, if and only if there is a linear ordering $\pi \in \Pi(V - C, C)$ of G, 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 3.1, we have

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

By Lemma 4.3, 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 algorithm to compute all maximum cliques. It recursively extends a clique by all candidate vertices once.

4.1.2. When and How to Use the TWDP Algorithm. Several other algorithmic engineering studies have been made for the computation of treewidth. These include exact algorithms, upper-bound heuristics, lower-bound heuristics, and preprocessing methods. Many of these results are also reported in TreewidthLib [2004]. A series of three overview papers is being made [Bodlaender and Koster 2010, 2011, **?**], reporting on respectively upper-bound heuristics, lower-bound heuristics, and exact algorithms and preprocessing for treewidth. A few of the experiences that were obtained are useful to see when and where the TWDP algorithm is of use.

An approach that is sometimes successful to obtain the exact treewidth is the following: we try out a few different upper-bound heuristics, and a good lower-bound heuristic, for example, the LBP+(MMD+) heuristic from Bodlaender et al. [2006]. In several instances reported in TreewidthLib [2004], the best upper-bound matches the lower bound, and we have obtained in a relatively fast way an exact bound on the treewidth of the instance graph. When these bounds do not match, and the graph is not too large, the TWDP algorithm can be of good use. We give the TWDP algorithm as input the best available upper bound for the treewidth.

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 TWDP algorithm with 15 as input for an upper bound, we obtained the exact treewidth of 14 for this graph, and hence also for celar03.

The TWDP 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, for example, 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, this 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 Bodlaender et al. [2006], Gogate and Dechter [2004]; Koster et al. [2005], a lower bound on $\mathbf{tw}(G')$ is computed to obtain a lower bound for G. With Algorithm 4, we can compute $\mathbf{tw}(G')$ exactly to obtain a lower bound for $\mathbf{tw}(G)$. That is, given a graph whose size is too large to compute its treewidth exactly, we contract edges until we obtain a graph that is small enough for our program. The exact treewidth of this minor gives a lower bound for the treewidth of G. In our experiments, and other results report on [TreewidthLib 2004], this has shown to be a viable lower bound strategy.

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 Bodlaender et al. [2006]). Figure 2 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 2 shows the impact of the upper bound on the memory consumption (and time consumption) of the algorithm.

4.1.3. Implementation Details. The algorithm was implemented in C++, using the Boost graph library [Boost 1999–2009], 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 (some algorithms can be tested

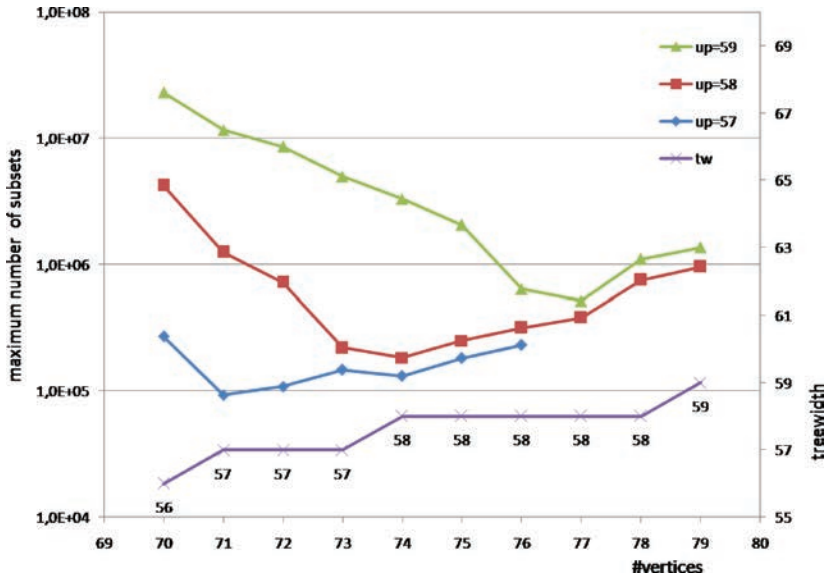

Fig. 2. Maximum number of subsets S during algorithm for different upper bounds.

online [Koster 2009]). Experiments were carried out on a number of graphs taken from applications; several were used in other experiments. See TreewidthLib [2004] for the used graphs, information on the graphs, and other results of experiments to compute the treewidth. The experiments were carried out on a Linux-operated PC with Intel Core 2 Quad 2.83-GHz processor and 8 GB of internal memory, with one exception: the results for queen7-7 (marked *) have been obtained on a Sun computer with 4 AMD Dualcore Opteron 875, 2.2-GHz processor, and 20-GB internal memory. The program did not use parallelism.

4.1.4. Experiments. In Table I, 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 (i.e., we start with $up = n - 1$. 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. (The case no-C corresponds to setting $C = \emptyset$ in Algorithm 4.) In several instances reported in TreewidthLib [2004], 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 TWDP algorithm can be of good use. The LBP+(MMD+) lower bound heuristic is described in Bodlaender et al. [2006], combining contractions with a technique by Clautiaux et al. [2003].

In the last two columns of Table I, we compare the TWDP algorithm with the *quickBB* algorithm of Gogate and Dechter [2004] (a branch-and-bound algorithm based on the linear ordering characterization). Here, *#nodes* denotes the number of nodes in the search tree. The algorithm was run on the same machine as the TWDP algorithm. The results show that our algorithm outperforms the quickBB algorithm on 9 out of

Table I. Experimental Results for Some DIMACS Vertex Coloring Graphs, Some Probabilistic Networks and Frequency Assignment

				no *up*, no *C*			with *up*, no *C*			with *up*, w *C*		quickBB	
instance	$\|V\|$	$\|E\|$	**tw**	CPU	max $\|TW\|$	*up*	CPU	max $\|TW\|$	ω	CPU	max $\|TW\|$	CPU	#nodes
myciel3	11	20	5	0.00	240	5	0.00	35	2	0.00	21	0.00	12
myciel4	23	71	10	4.88	296835	10	0.10	4422	2	0.08	4064	0.08	1193
queen5-5	25	160	18	0.09	18220	18	0.01	944	5	0.00	392	0.83	7735
queen6-6	36	290	25	22.95	2031716	26	0.69	18872	6	0.22	6994	31.69	232678
queen7-7*	49	476	35	-	-	37	1012.12	96517095	7	248.03	24410915	12381.70	63291238
pathfinder-pp	12	43	6	0.00	107	6	0.00	1	6	0.00	1	0.00	26
oesoca+-pp	14	75	11	0.00	48	11	0.00	5	9	0.00	5	0.00	32
fungiuk	15	36	4	0.06	4713	4	0.00	4	5	0.00	4	0.00	1
weeduk	15	49	7	0.02	2906	7	0.00	35	8	0.00	35	0.00	1
munin-kgo-pp	16	41	5	0.08	6892	5	0.00	2	4	0.00	2	0.00	24
wilson	21	27	3	9.59	350573	3	0.06	2412	3	0.06	2342	0.00	1
water-pp	22	96	9	0.95	77286	10	0.03	816	6	0.00	475	0.03	370
oow-trad-pp	23	54	6	28.03	1065120	6	0.06	2953	4	0.04	1895	2.45	55345
barley-pp	26	78	7	226.05	6110572	7	0.43	13597	5	0.20	7971	0.50	9114
oow-bas	27	54	4	939.65	19937301	4	0.01	303	4	0.00	111	0.00	8
oow-solo-pp	27	63	6	655.36	17048070	6	0.60	22484	4	0.21	9426	33.93	615350
ship-ship-pp	30	77	8	-	-	9	176.04	3062863	4	33.16	820910	4824.59	834334410
water	32	123	9	-	-	10	7.32	127545	6	0.96	25874	0.03	370
oow-trad	33	72	6	-	-	6	77.02	1162650	4	8.48	178846	2.82	56036
mildew	35	80	4	-	-	4	1.90	33045	4	0.25	5431	0.00	37
mainuk	48	198	7	-	-	8	-	-	8	1162.24	11748147	0.00	54
celar03-pp-001	38	238	14	-	-	15	76.08	911918	8	2.41	55504	122.25	1106812

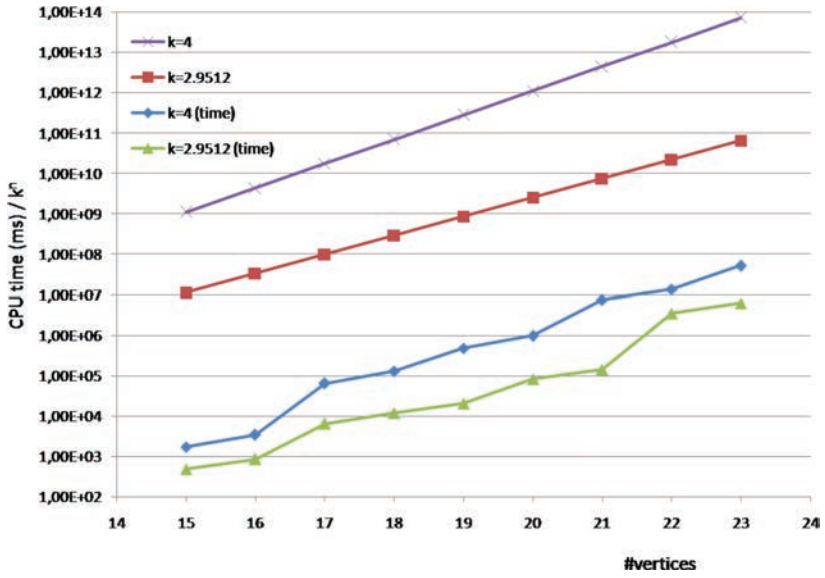

Fig. 3. CPU time to generate all required subproblems for graphs of different sizes.

22 instances, whereas it is slower in only 5 cases. We were unable to compare our results with the algorithm of Shoikhet and Geiger [1997]; the only other implemented exact algorithm for treewidth that we know of [Bachoore and Bodlaender 2006] is outperformed by *quickBB*.

4.2. Polynomial-Space Algorithms

The computational results in Table I are limited to relatively small graphs due to the exponential space requirement. We nevertheless suggest the use of the TWDP algorithm instead of the polynomial space algorithms in practice. Algorithms 2 and 3 require the computations of large numbers of vertex-subsets. Already the generation of these subsets requires significant computation time. Figure 3 illustrates this for $n = 15, \ldots, 23$. The lower lines show the CPU time (in milliseconds) for only generating recursively all required subsets of $\{1, \ldots, n\}$ without any further computations. The upper lines show the number k^n for $k = 2.9512$ and $k = 4$. From this figure, it will be clear that any real implementation will be significantly slower than the TWDP algorithm for the graphs of Table I and beyond.

5. CONCLUDING REMARKS

In this article, 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 $O^*(2.9512^n)$. Using similar methods, but a better method of enumerating potential maximal cliques, our result was recently improved to $O(2.6151^n)$ time by Fomin and Villanger [2012].

For several problems that can be formulated as linear ordering problems, there are algorithms, similar to those we gave here, that is, an algorithm that with running time and space $O^*(2^n)$, resembling the classic Held-Karp algorithm for TSP [Held and Karp 1962], and an algorithm with running time $O^*(4^n)$ and polynomial space that resembles the algorithm by Gurevich and Shelah [1987]. These include 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. The proofs are more or less straightforward but tedious modifications of the proofs given for treewidth in this paper, and are given in Bodlaender et al. [2012]. In a few cases, better algorithms are known, for example, for MINIMUM FILL-IN [Fomin et al. 2008; Fomin and Villanger 2012, 2010] and very recently for PATHWIDTH [Suchan and Villanger 2009; Kitsunai et al. 2012] and CUTWIDTH [Cygan et al. 2011] of bipartite graphs. Recently, Björklund [2010] introduced a randomized algorithm for HAMILTONIAN CYCLE of time $O^*(1.66^n)$. It is interesting if faster randomized algorithms can be found for other vertex ordering problems.

Koivisto and Parviainen [2010] have very recently obtained algorithms that give a tradeoff between the Held-Karp and Gurevich-Shelah approaches, that is, algorithms with less than $O^*(T^n)$ time with $T < 4$ and $O^*(C^n)$ memory with $C < 2$. In several cases, $TC < 4$. The approach of Koivisto and Parviainen [2010] can be applied to all of these problems, TREEWIDTH and several other problems, including TRAVELLING SALESMAN PROBLEM.

Several approaches can be tried to improve the algorithm reported in Section 4.1.1. We mention two such approaches, that possibly decrease the number of table entries that are created during the TWDP algorithm, but give a significant increase in the time per entry. For an entry (S, r) in a table TW_{i-1}, we can first apply some *lower-bound heuristic* to the graph $G^+[S]$, and then set r to the maximum of r and this lower bound, which can in some cases allow us to delete the table entry. While this can decrease the number of table entries that must be processed, it gives a large increase in the time per entry. Several different lower-bound methods that can be tried here exist, see, for example, the overviews in Bodlaender [2006] and Bodlaender and Koster [2011]. Running one or more lower-bound heuristics on G and checking if the largest known lower-bound matches the best-known upper bound is a simple and sometimes very effective approach. One can also try to see if $G^+[S]$ contains simplicial or almost simplicial vertices. If $G^+[S]$ has a *simplicial* vertex v, then there is an optimal elimination scheme for $G^+[S]$ that starts with v, so, instead of trying each set $W \cup \{x\}$ for all $x \in V - S$, we can only take the set $W \cup \{v\}$. Under additional conditions, *almost simplicial* vertices can play the same role. Some details can be found in Bodlaender et al. [2005]. Another approach may be to use some of our techniques in combination with a branch and bound approach, for example , add memorization and a clique at the end of the elimination sequence to the algorithm of Gogate and Dechter [2004].

Other recent theoretical results for which an experimental evaluation would be very interesting are the new approach to list potential maximal cliques by Fomin and Villanger [2010], the space-time tradeoff algorithms by Koivisto and Parviainen [2010], and a comparison between the simple $O^*(2^n)$ algorithm and algorithms for PATHWIDTH from Suchan and Villanger [2009] and Kitsunai et al. [2012].

ACKNOWLEDGMENT

We thank the referees for useful comments.

REFERENCES

ARNBORG, S., CORNEIL, D. G., AND PROSKUROWSKI, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algeb. Disc. Meth. 8*, 277–284.

BACHOORE, E. H. AND BODLAENDER, H. L. 2005. New upper bound heuristics for treewidth. In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*. S. E. Nikoletseas, Ed., Lecture Notes in Computer Science, vol. 3503, Springer Verlag, 217–227.

BACHOORE, E. H. AND BODLAENDER, H. L. 2006. A branch and bound algorithm for exact, upper, and lower bounds on treewidth. In *Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM 2006)*, S.-W. Cheng and C. K. Poon, Eds., Lecture Notes in Computer Science, vol. 4041, Springer Verlag, 255–266.

BJÖRKLUND, A. 2010. Determinant sums for undirected Hamiltonicity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*. IEEE, 173–182.

BODLAENDER, H. L. 1998. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci. 209*, 1–45.

BODLAENDER, H. L. 2005. Discovering treewidth. In *Proceedings of the 31st Conference on Current Trends in Theory and Practive of Computer Science (SOFSEM 2005)*. P. Vojtáš, M. Bieliková, and B. Charron-Bost, Eds., Lecture Notes in Computer Science, vol. 3381, Springer Verlag, 1–16.

BODLAENDER, H. L. 2006. Treewidth: Characterizations, applications, and computations. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2006)*. F. V. Fomin, Ed., Lecture Notes in Computer Science, vol. 4271, Springer Verlag, 1–14.

BODLAENDER, H. L., FOMIN, F. V., KOSTER, A. M. C. A., KRATSCH, D., AND THILIKOS, D. M. 2012. A note on exact algorithms for vertex ordering problems on graphs. *Theory Comput. Syst. 50,* 3, 420–432.

BODLAENDER, H. L., AND KOSTER, A. M. C. A. 2010. Treewidth computations I. Upper bounds. *Inf. Computat. 208*, 259–275.

BODLAENDER, H. L., AND KOSTER, A. M. C. A. 2011. Treewidth computations II. Lower bounds. *Inf. Computat. 209*, 1103–1119.

BODLAENDER, H. L., KOSTER, A. M. C. A., AND VAN DEN EIJKHOF, F. 2005. Pre-processing rules for triangulation of probabilistic networks. *Computat. Intell. 21,* 3, 286–305.

BODLAENDER, H. L., KOSTER, A. M. C. A., AND WOLLE, T. 2006. Contraction and treewidth lower bounds. *J. Graph Algor. Appl. 10*, 5–49.

BOOST 1999–2009. Boost C++ libraries. http://www.boost.org/.

BOUCHITTÉ, V., AND TODINCA, I. 2001. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput. 31*, 212–232.

BOUCHITTÉ, V., AND TODINCA, I. 2002. Listing all potential maximal cliques of a graph. *Theoret. Comput. Sci. 276*, 17–32.

CLAUTIAUX, F., CARLIER, J., MOUKRIM, A., AND NÉGRE, S. 2003. New lower and upper bounds for graph treewidth. In *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA 2003)*, J. D. P. Rolim, Ed., Lecture Notes in Computer Science, vol. 2647, Springer Verlag, 70–80.

CLAUTIAUX, F., MOUKRIM, A., NÉGRE, S., AND CARLIER, J. 2004. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Oper. Res. 38*, 13–26.

CYGAN, M., LOKSHTANOV, D., PILIPCZUK, M., PILIPCZUK, M., AND SAURABH, S. 2011. On cutwidth parameterized by vertex cover. In *Proceedings of the 6th International on Parameterized and Exact Computation (IPEC 2011)*. Lecture Notes in Computer Science Series, vol. 7112, Springer, 246–258.

DENDRIS, N. D., KIROUSIS, L. M., AND THILIKOS, D. M. 1997. Fugitive-search games on graphs and related parameters. *Theoret. Comput. Science 172*, 233–254.

FOMIN, F. V., GRANDONI, F., AND KRATSCH, D. 2005. Some new techniques in design and analysis of exact (exponential) algorithms. *Bull. EATCS 87*, 47–77.

FOMIN, F. V., KRATSCH, D., AND TODINCA, I. 2004. Exact (exponential) algorithms for treewidth and minimum fill-in. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, (ICALP 2004)*, J. Díaz, J. Karhumäki, A. Lepistö, and D. Sanella, Eds., Lecture Notes in Computer Science, vol. 3142, Springer Verlag, 568–580.

FOMIN, F. V., KRATSCH, D., TODINCA, I., AND VILLANGER, Y. 2008. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput. 38*, 1058–1079.

FOMIN, F. V. AND VILLANGER, Y. 2010. Finding induced subgraphs via minimal triangulations. In *Proceedings 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, J.-Y. Marion and T. Schwentick, Eds., Dagstuhl Seminar Proceedings Series, vol. 5, Leibniz-Zentrum für Informatik, Schloss Dagstuhl, Germany, 383–394.

FOMIN, F. V. AND VILLANGER, Y. 2012. Treewidth computation and extremal combinatorics. *Combinatorica 32,* 3, 289–308.

FULKERSON, D. R. AND GROSS, O. A. 1965. Incidence matrices and interval graphs. *Pac. J. Math. 15*, 835–855.

GOGATE, V. AND DECHTER, R. 2004. A complete anytime algorithm for treewidth. In *Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence (UAI 2004)*, D. M. Chickering and J. Y. Halpern, Eds., AUAI Press, 201–208.

GOLUMBIC, M. C. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

GUREVICH, Y. AND SHELAH, S. 1987. Expected computation time for Hamiltonian path problem. *SIAM J. Comput. 16*, 486–502.

HELD, M. AND KARP, R. 1962. A dynamic programming approach to sequencing problems. *J. SIAM 10*, 196–210.

KITSUNAI, K., KOBAYASHI, Y., KOMURO, K., TAMAKI, H., AND TANO, T. 2012. Computing directed pathwidth in $O(1.89^n)$ time. In *Proceedings of the 7th International on Parameterized and Exact Computation (IPEC 2012)*. Lecture Notes in Computer Science Series, vol. 7535, Springer, 182–193.

KOIVISTO, M. AND PARVIAINEN, P. 2010. A space-time tradeoff for permutation problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*. ACM, 484–492.

KOSTER, A. M. C. A. 2009. ComputeTW – An interactive platform for computing treewidth of graphs. http://www.math2.rwth-aachen.de/∼koster/ComputeTW.

KOSTER, A. M. C. A., WOLLE, T., AND BODLAENDER, H. L. 2005. Degree-based treewidth lower bounds. In *Proceedings of the 4th International Workshop on Experimental and Efficient Algorithms (WEA 2005)*, S. E. Nikoletseas, Ed., Lecture Notes in Computer Science, vol. 3503, Springer Verlag, 101–112.

ROSE, D. J., TARJAN, R. E., AND LUEKER, G. S. 1976. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput. 5*, 266–283.

SHOIKHET, K. AND GEIGER, D. 1997. A practical algorithm for finding optimal triangulations. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'97)*. Morgan Kaufmann, 185–190.

SUCHAN, K. AND VILLANGER, Y. 2009. Computing pathwidth faster than 2^n. In *Proceedigs of the International Workshop on Parameterized and Exact Computaion (IWPEC 2009)*. Lecture Notes in Computer Science, vol. 5917, Springer, 324–335.

TARJAN, R. E. AND YANNAKAKIS, M. 1984. Simple linear time algorithms to test chordiality of graphs, test acyclicity of graphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput. 13*, 566–579.

TREEWIDTHLIB. 2004. Treewidthlib. http://www.cs.uu.nl/people/hansb/treewidthlib.

VILLANGER, Y. 2006. Improved exponential-time algorithms for treewidth and minimum fill-in. In *Proceedings of the 7th Latin American Symposium on Theoretical Informatics (LATIN 2006)*, J. R. Correa, A. Hevia, and M. A. Kiwi, Eds., Lecture Notes in Computer Science, vol. 3887, Springer Verlag, 800–811.

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

Received October 2006; revised November 2009; accepted May 2012