A linear time algorithm for finding tree-decompositions of small treewidth*

Hans L. Bodlaender†
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

In this paper, we give, for constant k, a linear time algorithm, that given a graph $G = (V, E)$, determines whether the treewidth of G is at most k, and if so, finds a tree-decomposition of G with treewidth at most k. A consequence is that every minor-closed class of graphs that does not contain all planar graphs has a linear time recognition algorithm. Another consequence is that a similar result holds when we look instead for *path*-decompositions with pathwidth at most some constant k.

Keywords: graph algorithms, treewidth, pathwidth, partial k-trees, graph minors.
AMS-classification: 68R10, 05C85, 05C05

1 Introduction

1.1 Background

The notions of 'tree-decomposition' and 'treewidth' have received much attention recently, not in the least due to the important role they play in the deep results on graph minors by Robertson and Seymour (see e.g. [26, 27, 29, 30, 28], and many other papers in this series). (See also [20].) Also, many graph problems, including a very large number of well known NP-hard problems, have been shown to be linear time solvable on graphs that are given together with a tree-decomposition of treewidth at most k, for constant k. (See, amongst others [2, 5, 6, 7, 11, 9, 13, 14, 15, 31, 33].)

The first step of algorithms that exploit small treewidth of input graphs is to find a tree-decomposition with treewidth bounded by a constant, although possible

*This work was partially supported by the ESPRIT Basic Research Actions of the EC under contract 7141 (project ALCOM II).

†Email: hansb@cs.ruu.nl

not optimal. So far, this step dominated the running time of most algorithms, as the second step (some kind of 'dynamic programming' algorithm using the tree-decomposition) usually costs only linear time. The best algorithm known so far for this 'first step' was an algorithm by Reed [25], which costs $O(n \log n)$. In this paper, we improve on this result, and give a linear time algorithm.

The problem 'Given a graph $G = (V, E)$ and an integer k, is the treewidth of G at most k' is NP-complete [3]. Much work has been done on this problem for constant k. For $k = 1, 2, 3$, linear time algorithms exist [24]. Arnborg et al. [3] showed that the problem is solvable in $O(n^{k+2})$ time for constant k. Then, Robertson and Seymour gave a non-constructive proof of the existence of $O(n^2)$ decision algorithms [28]. Actually, this algorithm is of a 'two steps' form, as described above. The first step is to apply an $O(n^2)$ algorithm, that either outputs that the treewidth of G is larger than k, or outputs a tree-decomposition with treewidth at most $4k$. (Actually, the result is stated in [28] in terms of 'branchwidth', but this is an unimportant technical difference.) The second step checks in linear time a finite characterization of the graphs with treewidth at most k in terms of forbidden minors. In [8] (using results from [19]) it was shown that the non constructive elements can be avoided using self-reduction without increasing the running time by more than a (huge) constant factor.

Both Lagergren [22] and Reed [25] improve on the 'first step'. Lagergren gives a sequential algorithm that uses $O(n \log^2 n)$ time, and a parallel algorithm that uses $O(n)$ processors and $O(log^3 n)$ time. Reed gives a sequential $O(n \log n)$ algorithm, which can be seen to have a parallel implementation with $O(n/ \log n)$ processors and $O(\log^2 n)$ time. A related probabilistic result (with running time $O(n \log^2 n + n|\log p|)$, p the error of probability) was found by Matousěk and Thomas [24]. Each of these algorithms either determines that the input graph G has treewidth more than k, or finds a tree-decomposition of G with treewidth bounded by some constant (linear in k). They all are based upon finding 'balanced separators' in some clever ways. Our algorithm uses a different approach: we reduce the problem in linear time to a problem on a smaller graph by edge contraction or removing 'simplicial vertices'.

Independently, Lagergren and Arnborg [23] and Bodlaender and Kloks [11, 21] showed that the 'second step' can be done without use of graph minors, and give explicit algorithms to test in linear time whether G has treewidth at most k, once a tree-decomposition of G with bounded treewidth is available. Moreover, from these results it follows that a technique of Fellows and Langston [18] can be used to compute the obstruction set of the class of graphs with treewidth $\leq k$. Bodlaender and Kloks also show how if existing, a tree-decomposition with treewidth at most k can be computed in the same time bounds. Results of a similar flavor were obtained independently by Abrahamson and Fellows [1].

Recognition algorithms for graphs with treewidth $\leq k$ (k constant) have been designed by Arnborg et al. [4]. These algorithms use linear time, but polynomial, not linear memory (it is allowed that the algorithm consults the contents of memory that is never written to). A disadvantage of this approach is that it is not known how to *construct* tree-decompositions with small treewidth by the method.

1.2 Main idea of algorithm

The main result in this paper is the following.

Theorem 1.1 *For all $k \in N$, there exists a linear time algorithm, that tests whether a given graph $G = (V, E)$ has treewidth at most k, and if so, outputs a tree-decomposition of G with treewidth at most k.*

The main idea of the algorithm is as follows: vertices are partitioned into two sets, one with vertices of 'low degree', and one with vertices of 'high degree'. It can be shown for graphs with treewidth at most fixed constant k, that there are only 'few' high degree vertices. Two cases are distinguished:

1. 'Sufficiently many' vertices of low degree are adjacent to one or more other vertices of low degree. In this case, it can be shown that any maximal matching in G contains sufficiently many ($\Omega(n)$) edges. We compute the graph G' obtained by contracting all edges in a maximal matching. Recursively, we compute a tree-decomposition of treewidth at most k of G', or conclude that the treewidth of G', and hence the treewidth of G is larger than k. From this tree-decomposition, one easily can build a tree-decomposition of G with treewidth at most $2k + 1$. This latter tree-decomposition is used to solve the problem, using the algorithm of Bodlaender and Kloks [11, 21], mentioned above.

2. 'Only few' vertices of low degree are adjacent to one or more other vertices of low degree. It is shown that a certain collection of edges can be added to G without increasing the treewidth from a number at most k to a number larger than k. The 'improved graph of G' (G with this collection of edges added) is shown to have sufficiently many ($\Omega(n)$) vertices which are I-simplicial: their neighbors form a clique in the improved graph (and they fulfill some other, less important conditions). Recursively, a tree-decomposition with treewidth at most k is computed of G', obtained by removing all I-simplicial vertices from the improved graph of G, or one concludes that the treewidth of G', and hence of G is larger than k. Given such a tree-decomposition of G', one easily computes a tree-decomposition of G with treewidth at most k.

In each case, the amount of work of the non-recursive steps is linear, and each G' has size at most a constant fraction of the size of G. It follows that the algorithm uses linear time.

2 Definitions and preliminary results

The notion of treewidth was introduced by Robertson and Seymour [26].

Definition. 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 family of subsets of V, one for each node of T, and T a tree such that

- $\bigcup_{i \in I} X_i = V$.
- for all edges $(v, w) \in E$, there exists an $i \in I$ with $v \in X_i$ and $w \in X_i$.
- for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.

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

We use (X, T) as a shorthand notation for $(\{X_i \mid i \in I\}, T = (I, F))$. There are several equivalent notions, e.g. a graph is a partial k-tree, if and only if its treewidth is at most k [32]. We give some well known or easily derivable results.

Lemma 2.1 *If the treewidth of $G = (V, E)$ is at most k, then $|E| \leq k|V| - \frac{1}{2}k(k+1)$.*

Lemma 2.2 (See e.g. [12].) *Suppose $(\{X_i \mid i \in I\}, T = (I, F))$ is a tree-decomposition of $G = (V, E)$.*
(i) If $W \subseteq V$ forms a clique in G, then $\exists i \in I : W \subseteq X_i$.
(ii) If W_1, W_2 induce a complete bipartite subgraph in G, i.e., each vertex in W_1 is adjacent to each vertex in W_2, then $\exists i \in I : W_1 \subseteq X_i$ or $W_2 \subseteq X_i$.

Lemma 2.3 *If (X, T) is a tree-decomposition of $G = (V, E)$, and $\exists i \in I : v, w \in X_i$, $v \neq w$, then (X, T) is also a tree-decomposition of $G + (v, w) = (V, E \cup \{(v, w)\})$.*

Lemma 2.4 *Suppose v and w have at least $k+1$ common neighbors in $G = (V, E)$. If the treewidth of G is at most k, then the treewidth of $G + (v, w)$ is also at most k. Moreover, any tree-decomposition of G with treewidth at most k is also a tree-decomposition of $G + (v, w)$ with treewidth at most k and vice versa.*

Proof: Suppose (X, T) is a tree-decomposition of G with treewidth at most k. By lemma 2.2(b) either there exists an $i \in I$ with $v, w \in X_i$, in which case the lemma directly follows, or there exists an $i \in I$ with X_i contains the set W of all common neighbors of v and w. Now, if we add edges between all non-adjacent vertices in W, we obtain a graph G' that contains a clique with size at least $k+2$ (namely $W \cup \{v\}$) and has (X, T) as a tree-decomposition. But a graph with treewidth $\leq k$ cannot contain a clique with more than $k+1$ vertices, contradiction. □

A *contraction* is the operation that removes two adjacent vertices v, w, and replaces them by one new vertex that is made adjacent to all vertices that were adjacent to v and w.

We say a tree-decomposition (X,T) of treewidth k is *smooth*, if for all $i \in I$: $|X_i| = k+1$, and for all $(i,j) \in F$: $|X_i \cap X_j| = k$. Any tree-decomposition of a graph G can be transformed to a smooth tree-decomposition of G with the same treewidth: apply the following operations until none is possible:

- If for $(i,j) \in F$, $X_i \subseteq X_j$, then contract the edge (i,j) in T and take for the new node $X_{j'} = X_j$.
- If for $(i,j) \in F$, $X_i \not\subseteq X_j$ and $|X_j| < k+1$, then choose a vertex $v \in X_i - X_j$, and add v to X_j.
- If for $(i,j) \in F$, $|X_i| = |X_j| = k+1$, and $|X_i - X_j| > 1$, then subdivide the edge (i,j) in T; let i' be the new node; choose a vertex $v \in X_i - X_j$ and a vertex $w \in X_j - X_i$, and let $X_{i'} = X_i - \{v\} \cup \{w\}$.

Lemma 2.5 *If (X,T) is a smooth tree-decomposition of $G=(V,E)$ with treewidth k, then $|I| = |V| - k$.*

Proof: With induction to $|I|$. If $|I| = 1$, then clearly $|V| = k+1$. Suppose the lemma holds for $|I| = r-1$. Consider a smooth tree-decomposition (X,T) of a graph $G=(V,E)$ with treewidth k, with $|I| = r$. Let i be a leaf of T. There is a unique vertex v that belongs to X_i, but not to any set X_j, $j \in I - \{i\}$. If we remove node i from T, we get a smooth tree-decomposition of $G[V - \{v\}]$ with treewidth k and with $|I| - 1$ nodes. The result now follows by induction. □

In the remainder, we assume that k is a given fixed constant. We choose constants c_1, $c_2 \in R^+$, such that

$$c_2 = \frac{1}{4k^2 + 12k + 16} - \frac{c_1 \cdot k^2(k+1)}{2} > 0$$

(Note that c_1 and c_2 are both constants between 0 and 1. c_1 denotes an upper bound on the fraction of vertices that is of 'high degree' (defined below), and c_2 denotes a lower bound on the fraction of vertices that is remove in one of the two cases of the main part of the algorithm.)

Let $d = \max(k^2 + 4k + 4, \lceil 2k/c_1 \rceil)$. We say a vertex with degree at most d is a *low degree vertex*, and a vertex with degree larger than d is a *high degree vertex*. A vertex is said to be *friendly*, if it is a low degree vertex and it is adjacent to at least one other low degree vertex.

Lemma 2.6 *There are less than $c_1 \cdot |V|$ high degree vertices in a graph with treewidth k.*

Proof: If there are n_l high degree vertices, then G must contain at least $n_l \cdot d/2$ edges. By lemma 2.1, $n_l \cdot d/2 < k|V|$. □

A *maximal matching* of a graph $G = (V, E)$ is a set of edges $M \subseteq E$ such that no two edges in M share an endpoint, and every $e \in E - M$ shares an endpoint with an edge in M. One easily finds a maximal matching in $O(|V| + |E|)$ time with a greedy algorithm.

Lemma 2.7 *If there are n_f friendly vertices in $G = (V, E)$, then any maximal matching of G contains at least $n_f/(2d)$ edges.*

Proof: Consider a maximal matching M. Any friendly vertex must be endpoint of an edge in M, or adjacent to a friendly vertex that is endpoint of an edge in M. To each edge e of M, we can associate the at most $2d$ friendly vertices that are endpoint of e or adjacent to a friendly (and hence low degree) endpoint of e. If a friendly vertex has not been associated to at least one edge in M, then M is not maximal. Hence $|M| \geq n_f/(2d)$. □

Let M be a maximal matching in $G = (V, E)$, and let $G' = (V', E')$ be the graph obtained by contracting all edges in M. Define $f_M : V \to V'$ by $f_M(v) = v$ if v is not an endpoint of an edge in M, and let $f_M(v) = f_M(w)$ be the vertex resulting of contracting edge $(v, w) \in M$.

Lemma 2.8 *Let M, G, G', f_M be as above. If (X, T) is a tree-decomposition of G' with treewidth k, then (Y, T), defined by $Y_i = \{v \in V \mid f_M(v) \in X_i\}$ is a tree-decomposition of G with treewidth at most $2k + 1$.*

Proof: This follows easily from the definitions. □

Lemma 2.9 *Let G, G' be as above. The treewidth of G' is at most the treewidth of G.*

Proof: G' is a minor of G. The treewidth of a graph cannot increase by taking minors. □

The set of neighbors of a vertex v in $G = (V, E)$ is denoted by $N_G(v) = \{w \in V \mid (v, w) \in E\}$. Finally, we mention some algorithmic results.

Theorem 2.10 *(Bodlaender, Kloks [11, 21]) For all k, l, there exists a linear time algorithm, that when given a graph $G = (V, E)$ together with a tree-decomposition (X, T) of G with treewidth at most l, determines whether the treewidth of G is at most k, and if so, finds a tree-decomposition of G with treewidth at most k.*

When we are given a graph $G = (V, E)$ as a collection of r edges where some edges of G may appear more than once in the collection, we can remove all multiple copies of edges in $O(|V| + r)$ time, by applying bucket sort twice.

3 Simplicial vertices

For a graph $G = (V, E)$, let the *improved* graph $G' = (V, E')$ of G be the graph, obtained by adding an edge (v, w) to E for all pairs v, $w \in V$ such that v and w have at least $k + 1$ common neighbors of low degree in G.

Lemma 3.1 *If the treewidth of G is at most k, then the treewidth of the improved graph of G is at most k. Moreover, any tree-decomposition of G with treewidth at most k is also a tree-decomposition of the improved graph with treewidth at most k, and vice versa.*

Proof: This follows directly from lemma 2.4. □

We say a vertex v is *simplicial in* G, if its neighbors form a clique in G. We say v is I-simplicial, if it is simplicial in the improved graph of G, it is of low degree in G, and it is not a friendly vertex in G. (These latter two requirements are needed for obtaining a linear running time of the algorithm.) We now derive, by a series of lemmas the following result that states that if we have 'few' high degree vertices and 'few' friendly vertices, then if the treewidth of G is at most k, then we have 'many' I-simplicial vertices.

Theorem 3.2 *For every graph $G = (V, E)$, at least one of the following properties holds:*

- *G contains at least $|V|/(4k^2 + 12k + 16)$ friendly vertices.*
- *The improved graph of G contains at least I-simplicial vertices.*
- *The treewidth of G is larger than k.*

Proof: The proof of theorem 3.2 will be given with help of several lemmas.

A vertex $v \in V$ is said to be *pretty* with respect to some tree-decomposition (X, T), if it is of low degree, not friendly, and there exists a node $i \in I$, such that all neighbors of v belong to X_i.

Lemma 3.3 *Suppose (X, T) is a smooth tree-decomposition of $G = (V, E)$ with treewidth k.*
(i) To every leaf i of T, one can associate a low degree vertex $v \in X_i$, that is friendly or pretty with respect to (X, T), and there does not exist a $j \in I$, $j \neq i$, with $v \in X_j$.
(ii) To every path $i_0, i_1, \ldots, i_{k^2+3k+3}$ in T with $i_1, \ldots, i_{k^2+3k+2}$ nodes of degree 2 in T, one can associate at least one vertex $v \in X_{i_1} \cup \cdots \cup X_{i_{k^2+3k+2}}$ that is friendly or pretty with respect to (X, T), such that v does not belong to a set X_j with $j \in I$ a node, not on this path.

Proof: (i) Let j be the neighbor of leaf i in T. Let v be the unique vertex in $X_i - X_j$. v is only adjacent to vertices in X_i. Either all neighbors of v are of high degree, in which case v is pretty w.r.t. (X, T), or a neighbor of v is of low degree, in which case v is friendly.
(ii) Note that $|X_{i_0} \cup \cdots \cup X_{i_{k^2+3k+3}}| = k^2 + 4k + 4 \leq d$. Hence all vertices in $X_{i_1} \cup \cdots \cup X_{i_{k^2+3k+2}} - (X_{i_0} \cup X_{i_{k^2+3k+3}})$ are of low degree. Suppose neither of them is friendly, i.e. they are only adjacent to high degree vertices in $X_{i_0} \cup X_{i_{k^2+3k+3}}$. Suppose X_{i_0} contains r high degree vertices, say $w_1, \ldots, w_r$. Clearly $r \leq |X_{i_0}| = k+1$. Each of these, say w_s, belongs to successive sets $X_{i_0}, X_{i_1}, \cdots, X_{i_{w_s}}$. Suppose w.l.o.g. $i_{w_1} \leq i_{w_2} \leq \cdots \leq i_{w_r}$. If some low degree vertex v belongs to exactly one set X_{i_j}, $1 \leq j \leq k^2+3k+2$, then it must be pretty w.r.t. (X, T). If some low degree vertex v belongs only to (a subset of) sets $X_{i_{w_j}+1}, \ldots, X_{i_{w_{j+1}}}$, then all neighbors of v belong to $X_{i_{w_{j+1}}}$, hence v is pretty w.r.t. (X, T). All vertices in $X_{i_1} \cup \cdots \cup X_{i_{k^2+3k+2}}$ not of one of these two types must belong to at least one of the sets $X_{i_0}, X_{i_{w_1}}, \ldots, X_{i_{w_r}}, X_{i_{k^2+3k+3}}$. This are in total at most $(k+1)(k+3) = k^2 + 4k + 3$ vertices. So, at least one vertex in $X_{i_1} \cup \cdots \cup X_{i_{k^2+3k+2}} - X_{i_0} - X_{i_{k^2+3k+3}}$ must be pretty w.r.t. (X, T). $\square$

A leaf-path collection of a tree T is a collection of leaves in T, plus a collection of paths of length $k^2 + 3k + 4$ in T, where all nodes on a path which are not an endpoint of a path have degree 2 in T and do not belong to any other path in the collection. The size of the collection is the total number of leaves plus the total number of paths in the collection.

Lemma 3.4 *Each tree with r nodes contains a leaf-path collection of size at least $r/(2k^2 + 6k + 8)$.*

Proof: Let r_b be the number of nodes of degree at least 3, r_l the number of leaves, and r_2 the number of nodes of degree 2. Clearly, $r_b < r_l$. All nodes of degree 2 belong to $< r_l + r_b$ connected components of the forest, obtained by removing all leaves and all nodes with degree 3 or larger from the tree. Each such component contains at most $k^2 + 3k + 3$ nodes, not part of a leaf-path collection of maximum size. So, there are less than $(r_b + r_l)(k^2 + 3k + 3)$ nodes of degree 2, not on a path in the collection. Hence, there are at least

$$\frac{r_2 - (r_b + r_l)(k^2 + 3k + 1)}{k^2 + 3k + 4}$$

paths in a leaf-path collection of maximum size. It follows that the maximum size of a leaf-path collection is at least

$$\max(r_l, \frac{r_2 - (r_b + r_l)(k^2 + 3k + 3)}{k^2 + 3k + 4} + r_l) \geq \frac{1}{2} \cdot \frac{r}{k^2 + 3k + 4}.$$

$\square$

Corollary 3.5 *If (X, T) is a smooth tree-decomposition of $G = (V, T)$ with treewidth k, then G contains at least $|V|/(2k^2+6k+8)-1$ vertices that are friendly, or pretty with respect to (X, T).*

Proof: T contains $|V| - k$ nodes (lemma 2.5). Now apply lemma 3.3 and 3.4. □

A set $Y \subseteq V$ of high degree vertices is said to be *semi-important* with respect to tree-decomposition (X, T) of $G = (V, E)$, if there exists an $i \in I$ with $Y \subseteq X_i$. A set Y is said to be *important*, if it is semi-important w.r.t. (X, T) and not contained in any larger semi-important set w.r.t. (X, T).

Lemma 3.6 *Let (X, T) be a tree-decomposition of $G = (V, E)$ with treewidth k. The number of different important sets w.r.t. (X, T) is at most the number of high degree vertices in G.*

Proof: Let L be the set of high degree vertices in G. $(\{X_i \cap L \mid i \in I\}, T)$ is a tree-decomposition of $G[L]$. Each important set Y is a set $X_i \cap L$ which is not contained in another set $X_{i'} \cap L$. Repeatedly, contract edges (i, i') in T with $X_i \cap L \supseteq X_{i'} \cap L$, with the new formed node containing all vertices in X_i. The resulting tree-decomposition of $G[L]$ contains the same maximal sets X_i and will have at most $|L|$ nodes. □

A function f that maps each pretty (with respect to some tree-decomposition (X, T)) vertex v to an important (with respect to (X, T)) set Y with $N_G(v) \subseteq Y$, is called a *pretty-to-important-function* for (X, T). By definition, a pretty-to-important-function always exists.

Lemma 3.7 *Let f be a pretty-to-important-function for a smooth tree-decomposition (X, T) of $G = (V, E)$ with treewidth k. Let Y be an important set w.r.t. (X, T). Then at most $\frac{1}{2}k^2(k+1)$ pretty vertices w.r.t (X, T) (and G) in $f^{-1}(Y)$ are not I-simplicial.*

Proof: Assign each non-I-simplicial pretty vertex v to a pair of neighbors of v, that are non-adjacent in the improved graph. To each pair of vertices, there cannot be assigned more than k vertices, as otherwise they would have at least $k+1$ common low degree neighbors, and there would be an edge between them in the improved graph.

It follows that the number of non-I-simplicial pretty vertices v with $f(v) = Y$ is at most $\frac{1}{2}|Y| \cdot (|Y| - 1) \leq \frac{1}{2}k^2(k+1)$. □

Now we can prove theorem 3.2. Suppose that G contains less than $|V|/(4k^2 + 12k + 16)$ friendly vertices and that the treewidth of G is at most k. By lemma 2.6, there are at most $c_1 \cdot |V|$ high degree vertices in G, and hence, by lemma 3.6, the number of important sets w.r.t. an arbitrary smooth tree-decomposition (X, T) of

G with treewidth $\leq k$ is at most $c_1 \cdot |V|$. Using the fact that a pretty-to-important function always exists, and lemma 3.7, it follows that at most $\frac{1}{2}k^2(k+1)\cdot(c_1\cdot|V|-1)$ helpful vertices w.r.t. (X,T) are not I-simplicial. Using corollary 3.5, it follows that $|V|/(2k^2+6k+8)-1-|V|/(4k^2+12k+16)-\frac{1}{2}k^2(k+1)\cdot(c_1\cdot|V|-1) \geq c_2\cdot|V|$ vertices are I-simplicial. This ends the proof of theorem 3.2 □

Lemma 3.8 *Let (X,T) be a tree-decomposition of treewidth at most k of the graph G', obtained by removing all I-simplicial vertices (and adjacent edges) from the improved graph of graph $G=(V,E)$. Then, for all I-simplicial vertices v, there exists an $i \in I$ with $N_G(v) \subseteq X_i$.*

Proof: Note that by definition, I-simplicial vertices are non-adjacent in G, and their neighborhood forms a clique in the improved graph of G. The result now follows directly from lemma 2.2(i). □

4 Main algorithm

We now give a recursive description of the main algorithm. Some details will be discussed in section 5. Our algorithm, when given a graph $G=(V,E)$, either

- outputs that the treewidth of G is larger than k
- or outputs a tree-decomposition of G with treewidth at most k.

For 'very small graphs' (i.e. with at most some constant number of vertices) any other finite algorithm is used to solve the problem. Otherwise, the following algorithm is used:

First, check whether $|E| \leq k\cdot|V| - \frac{1}{2}k(k+1)$. If this is not the case, we know by lemma 2.1 that the treewidth of G is larger than k: **stop**.

Now, count the number of friendly vertices. If there are at least $|V|/(4k^2+12k+16)$ friendly vertices, do the following:

- Find a maximal matching $M \subseteq E$ in G.
- Compute the graph $G'=(V',E')$, obtained by contracting every edge in M.
- Recursively, apply the algorithm to G'.
- If G' has treewidth larger than k: **stop**. The treewidth of G is also larger than k. (See lemma 2.9.)
- Suppose the recursive call yielded a tree-decomposition (X,T) of G' with treewidth k. Construct a tree-decomposition (Y,T) of G with treewidth at most $2k+1$, as in lemma 2.8.

- Use the algorithm of theorem 2.10 and solve the problem.

If there are less than $|V|/(4k^2 + 12k + 16)$ friendly vertices, do the following:

- Compute the improved graph of G. (See section 5.)
- If there exists a I-simplicial vertex with degree at least $k+1$ then **stop**: the improved graph of G contains a clique with $k+2$ vertices, hence the treewidth of G is more than k.
- Put all I-simplicial vertices in some set SL. Compute the graph G', obtained by removing all I-simplicial vertices and adjacent edges from G.
- If $|SL| < c_2|V|$, then **stop**: the treewidth of G is larger than k. (See theorem 3.2.)
- (Now $|SL| \geq c_2|V|$.) Recursively apply the algorithm on G'.
- If the treewidth of G' is larger than k, then **stop**: as G' is a subgraph of G, also the treewidth of G is larger than k.
- Suppose the recursive call yielded a tree-decomposition (X, T) of G' with treewidth k. For all $v \in SL$, find an $i_v \in V$ with $N_G(v) \subseteq X_{i_v}$, and add a new node j_v to T with $X_{j_v} = \{v\} \cup N_G(v)$, and make j_v adjacent to i_v in T. (Such a node i_v exists by lemma 3.8.) The result is a tree-decomposition of G with treewidth at most k.

Correctness of the algorithm follows from results given in sections 2 and 3.

The running time of the algorithm can be estimated as follows. Either we recursively apply the algorithm on a graph with $(1 - 1/(2d(4k^2+12k+16))) \cdot |V|$ vertices (lemma 2.7) or on a graph with $(1 - c_2)|V|$ vertices. Write

$$c_3 = \max(1 - c_2,\ 1 - \frac{1}{2d \cdot (4k^2 + 12k + 16)}).$$

Note that $0 \leq c_3 < 1$. As all non recursive steps have a linear time implementation (see also section 5), we have that, if the algorithm takes $T(n)$ time on a graph with n vertices in the worst case, then $T(n) \leq T(c_3 \cdot n) + O(n)$, hence $T(n) = O(n)$. It also follows that the algorithm uses linear memory.

5 Some details of the algorithm

In this section we show that the steps of the algorithm given in section 4 can be implemented in linear time and linear memory. Most steps are rather straightforward, and left to the reader, or follow from earlier results. Note that we always may assume that the number of edges we are working with is linear in the number of vertices. We represent all graphs we work with by their adjacency lists.

Computing the improved graph and the I-simplicial vertices Number the vertices v_1, v_2, …, v_n. We use a queue Q, containing triples of the form $((v,w),x)$ with $v,w,x \in V$, or of the form $((v,w),\text{—})$, $v,w \in V$. Also, we use an array S, with for each $v_i \in V$ a stack $S[v_i]$ containing pairs of vertices. For all $(v_i,v_j) \in E$ with $i<j$, put $((v_i,v_j),\text{—})$ on Q. For all low degree vertices $v \in V$, for all pairs of neighbors $v_i, v_j \in N_G(v)$ with $i<j$, put $((v_i,v_j),v)$ on Q. Now 'bucket sort' Q twice, once to the first vertex entries, and once to the second vertex entries. After this double bucket sort, all pairs of the form $((v_i,v_j),\ldots)$ for fixed v_j, v_j will be in consecutive positions in Q. By inspecting Q, one can directly see what pairs of vertices have at least $k+1$ low degree common neighbors. (If at least $k+1$ entries $((v_i,v_j),v)$ are adjacent in Q for some pair v_i, v_j, (v_i,v_j) must be present in the improved graph.) For each such pair (v_i,v_j), and if a triple $((v_i,v_j),\text{—})$ is in Q, add the pair (v_i,v_j) to all stacks $S[v]$ for vertices v with $((v_i,v_j),v)$ in Q. This all can be done in linear time, using the consecutiveness.

Checking whether a low degree vertex v is I-simplicial can be done by inspecting $S[v]$: $S[v]$ will consist precisely of all edges between neighbors of v. As v is of low degree, $S[v]$ is of constant bounded size.

Adding I-simplicial vertices back in the tree-decomposition Suppose we have a tree-decomposition (X,T) of $G[V-SL]$ and we want to add all I-simplicial vertices in SL. For all $l \leq k$, we take a queue Q_l, in which we place all pairs $((v_{i_1},\ldots,v_{i_l}),i)$ for v_{i_1}, …, $v_{i_l} \in X_i$, $i \in I$, $i_1 < i_2 < \cdots i_l$, and all pairs $((v_{i_1},\ldots,v_{i_l}),v)$ with $N_G(v) = \{v_{i_1},\ldots,v_{i_l}$, $i_1 < i_2 < \cdots < i_l$.

For each l, $1 \leq l \leq k$, bucket sort Q_l l times, once for each of the l positions in the l-tuple. All entries of the form $((v_{i_1},\ldots,v_{i_l}),\ldots)$ will be at successive positions in Q_l after this operation. By a simple scan of Q_l, one can find for each entry $((v_{i_1},\ldots,v_{i_l}),v)$ an entry of the form $((v_{i_1},\ldots,v_{i_l}),i)$ for some $i \in I$. This node i is precisely the node where the new node j_v with $X_{j_v} = \{v\} \cup N_G(v)$ can be made adjacent to.

6 Final remarks

A consequence of the result of this paper is that all results that state that certain problems are solvable in linear time for graphs that are given together with a tree-decomposition of constant bounded treewidth, are turned into results that state that these problems can be solved in linear time on graphs with constant bounded treewidth. One of the most notable of such results is the following.

Theorem 6.1 *Every class of graphs that is closed under taking of minors and does not contain all planar graphs has a linear time recognition algorithm.*

Proof: See e.g. [28]. Use the algorithm, described in this paper, to find a tree-decomposition with constant bounded treewidth of the input graph, and use this tree-decomposition to test for minor inclusion for all graphs in the obstruction set of the class. □

In [16, 17] several such classes of graphs can be found. For several of these notions, we expect that constructive linear time algorithms can be designed. Recent research [10] shows that linear time algorithms can be constructed that solve Minimum Cut Linear Arrangement, Search Number, and some related problems for constant k, and output for 'yes'-instances the required linear arrangement.

Note that the result shown in this paper is equivalent to stating that (for fixed k) partial k-trees can be recognized and embedded in a k-tree (or a chordal graph with maximum clique size $k+1$) in linear time. Also, a direct consequence is the following:

Theorem 6.2 *For all $k \in N$, there exists a linear time algorithm, that tests whether a given graph $G = (V, E)$ has pathwidth at most k, and if so, outputs a path-decomposition of G with pathwidth at most k.*

Proof: First, use the algorithm described in this paper. When the treewidth of G is larger than k, then clearly also the pathwidth of G is larger than k. Otherwise, use the result from [11, 21], that states for all constants k, l that gives a linear time algorithm, that given a tree-decomposition of treewidth at most l of G, tests whether G has pathwidth at most k and if so, finds a path-decomposition of G with pathwidth at most k. □

The constant factor of the algorithm in this paper is very large, probably even for $k = 4$ it might be too large for practical purposes. Still, it is 'only' exponential in a polynomial in k. Still, ideas and techniques in this paper may help to develop really practical algorithms for the 'treewidth $\leq k$' problem. As a small increase in the treewidth often gives a large increase in the constant factors, an advantage of our algorithm over others is that we never have to work with tree-decompositions with treewidth more than $2k+1$. Also, finding I-simplicial vertices is done quite fast, and may be a good heuristic. Variants of our algorithm, perhaps with a better estimate of the constant c_2 needed for the test $|SL| \leq c_2 \cdot |V|$ may yield important savings in the constant factor of the running time.

It is also possible to modify the algorithm, such that it uses the algorithm in [11, 21] only on tree-decompositions with treewidth at most $k+1$, at the cost of increasing the running time to $O(n \log n)$. Provided that the algorithm in [11, 21] can be implemented quick enough, this modification may well be quite practical for small values of k (like $k = 4$ or $k = 5$). The idea is as follows: instead of using the construction of lemma 2.8, first find a set M' of at least $|M|/(k+1)$ edges in M such that no two vertices which are the result of contracting an edge in M' belong to a common set X_i. (Such a set can be quickly found in $O(n)$ time: the graph H, obtained by adding an edge between every pair of vertices in G' that share a common set X_i is a graph with treewidth k, hence is $(k+1)$-colorable. Hence, the set of vertices that are a result of an edge contraction contains an independent set in H of size at least $|M|/(k+1)$. Take M' the set of edges corresponding to the vertices in this independent set.) Define $f_{M'}$ as in section 2. Now (Y,T), defined by $Y_i = \{v \in v \mid f_{M'} \in X_i\}$ is a tree-decomposition of the graph, obtained from G by contracting all edges in $M - M'$ with treewidth at most $k+1$. Use the algorithm from [11, 21] to find a tree-decomposition of treewidth at most k of this graph. Repeat the process with this last tree-decomposition, and edge set $M - M'$ until the edge set is empty. This are at most $O(\log n)$ iterations. (This observation was also made by Jens Lagergren.)

It is possible to implement the algorithm, such that it runs on a pointer machine (a correct use of pointers is necessary, such that the addressing in the bucket sort algorithms can be done), and still uses linear time. We omit the (easy) details.

An interesting open problem is to find a parallel algorithm with a linear (or close to linear) time-processor product for the 'treewidth $\leq k$' problem.

Acknowledgements

I thank Jens Gustedt, Ton Kloks, Dieter Kratsch, and Bruce Reed for very useful comments on earlier draft versions of this paper.

References

[1] K. R. Abrahamson and M. R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics vol. 147*, pages 539–564. American Mathematical Society, 1993.

[2] S. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT*, 25:2–23, 1985.

[3] 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.

[4] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. In H. Ehrig, H. Kreowski, and G. Rozenberg, editors, *Proceedings of the Fourth Workshop on Graph Grammars and Their Applications to Computer Science*, pages 70–83. Springer Verlag, Lecture Notes in Computer Science, vol. 532, 1991. To appear in J. ACM.

[5] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12:308–340, 1991.

[6] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Disc. Appl. Math.*, 23:11–24, 1989.

[7] H. L. Bodlaender. Dynamic programming algorithms on graphs with bounded tree-width. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, pages 105–119. Springer Verlag, Lecture Notes in Computer Science, vol. 317, 1988.

[8] H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. In *Proc. 15th Int. Workshop on Graph-theoretic Concepts in Computer Science WG'89*, pages 232–244. Springer Verlag, Lect. Notes in Computer Science, vol. 411, 1990. To appear in: Annals of Discrete Mathematics.

[9] H. L. Bodlaender. A tourist guide through treewidth. Technical Report RUU-CS-92-12, Department of Computer Science, Utrecht University, Utrecht, 1992. To appear in Acta Cybernetica.

[10] H. L. Bodlaender, M. R. Fellows, and M. Halett, 1993. Unpublished results.

[11] H. L. Bodlaender and T. Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 544–555. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

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

[13] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7:555–582, 1992.

[14] B. Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[15] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sc.*, 109:49–82, 1993.

[16] M. R. Fellows and M. A. Langston. Nonconstructive advances in polynomial-time complexity. *Inform. Proc. Letters*, 26:157–162, 1987.

[17] M. R. Fellows and M. A. Langston. Nonconstructive tools for proving polynomial-time decidability. *J. ACM*, 35:727–739, 1988.

[18] M. R. Fellows and M. A. Langston. An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 520–525, 1989.

[19] M. R. Fellows and M. A. Langston. On search, decision and the efficiency of polynomial-time algorithms. In *Proceedings of the 21rd Annual Symposium on Theory of Computing*, pages 501–512, 1989.

[20] D. S. Johnson. The NP-completeness column: An ongoing guide. *J. Algorithms*, 8:285–303, 1987.

[21] T. Kloks. *Treewidth*. PhD thesis, Utrecht University, Utrecht, the Netherlands, 1993.

[22] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the 31rd Annual Symposium on Foundations of Computer Science*, pages 173–182, 1990.

[23] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pages 533–543. Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

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

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

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

[27] N. Robertson and P. D. Seymour. Graph minors. V. Excluding a planar graph. *J. Comb. Theory Series B*, 41:92–114, 1986.

[28] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. Manuscript, 1986.

[29] N. Robertson and P. D. Seymour. Graph minors. IV. Tree-width and well-quasi-ordering. *J. Comb. Theory Series B*, 48:227–254, 1990.

[30] N. Robertson and P. D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory Series B*, 52:153–190, 1991.

[31] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*. PhD thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.

[32] J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, A: Algorithms and Complexity Theory*, pages 527–631, Amsterdam, 1990. North Holland Publ. Comp.

[33] T. V. Wimer. *Linear Algorithms on k-Terminal Graphs*. PhD thesis, Dept. of Computer Science, Clemson University, 1987.