Better Algorithms for the Pathwidth and Treewidth of Graphs*

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

Abstract

In this paper we give, for all constants k, explicit $O(n \log^2 n)$ algorithms, that given a graph $G = (V, E)$, decide whether the treewidth (or pathwidth) of G is at most k, and if so, find a tree-decomposition or (path-decomposition) of G with treewidth (or pathwidth) at most k. In contrast with previous solutions, our algorithms do not rely on non-constructive reasoning, and are single exponential in k. This result implies a similar result for several graph notions that are equivalent with treewidth or pathwidth.

1 Introduction

The notions of pathwidth and treewidth play an important role in many different fields of computer science, often with different terminologies, e.g.

- Choleski factorization and Gauss elimination. (See e.g. [16].)
- VLSI-layout theory. (See e.g. [27].)
- theory of expert systems. (See e.g. [25].)
- algorithmic graph theory.
- theory of graph grammars. (See e.g. [21].)

In many cases, notations and notions are different from those used in this paper (and from each other). For instance, a graph has treewidth at most k, iff it is a partial k-tree; iff it is the subgraph of a chordal (= triangulated) graph with no clique larger than $k + 1$ vertices; iff it has dimension at most k. A graph has pathwidth at most k, iff its vertex separation number is at most k; iff its interval thickness is at most $k + 1$; iff its node search number is at most $k + 1$; iff it models an instance of the Gate Matrix Layout problem with a solution with at most $k + 1$ tracks. There are also equivalent characterizations with help of graph grammars, or k-terminal recursive families of graphs. (See e.g., [1, 8, 20, 31].)

Formally, the treewidth (pathwidth) of a graph is the minimum treewidth (pathwidth) over all tree-decompositions (path-decompositions) of the graph. (See Section 2 for definitions.) When a tree- or path-decompostion is found of a graph G with optimal treewidth, then usually one can easily

*This work was supported by the Foundation for Computer Science (S.I.O.N.) of the Netherlands Organization for Scientific Research (N.W.O.) and by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (project ALCOM).
†email: hansb@cs.ruu.nl
‡email: ton@cs.ruu.nl

construct representations of the graph corresponding to the equivalent notions (e.g., chordal graphs with minimum clique size that contain G, optimal node search strategies, optimal solutions to the Gate Matrix Layout problem, etc.) Thus, given a graph G, finding a tree- or path-decompostion of G with minimum treewidth is an important problem.

The notion of treewidth is also interesting because of its vital role in the theory of Graph Minors of Robertson and Seymour [28]. Also, a very large number of intractable graph problems become solvable in polynomial, and even linear time (and belong to the class NC), when restricted to graphs with bounded treewidth, given together with a suitable tree-decomposition. This set of problems includes many well-known NP-complete problems like Hamiltonian Circuit, Independent Set, etc., and even some PSPACE-complete problems (see e.g. [4, 6, 9, 11, 13, 31]). Typically, these algorithms use time polynomial in the number of vertices, but at least exponential in the treewidth of the input graph. Also, researchers in expert system theory have found out that several otherwise time consuming statistical computations can be done quickly when a tree-decomposition (known as: junction tree, or clique tree) with small treewidth is known (see e.g., [25, 30].)

Much reseach has been done on the problem of determining the treewidth and pathwidth of a graph, and finding tree- or path-decompositions with optimal treewidth. These problems are NP-complete [2]. However, if k is a fixed constant, then the problems become solvable in polynomial time. The first algorithms solving the problems for fixed k are based on dynamic programming, and use $O(n^{k+2})$ and $O(n^{2k^2+4k+8})$ time [2, 17]. Then, with help of graph minor theory, Robertson and Seymour showed the existence of $O(n^2)$ algorithms (for fixed k). However, these results are non-constructive in two ways: first, only existence of the algorithm is proven, but the algorithm itself is not known, and secondly the algorithm only outputs *yes* or *no*, but no tree- or path-decomposition. Also, the constant factors of these algorithms make them infeasible. With help of the notion of *self-reduction*, introduced by Fellows and Langston, it is possible to avoid the non-constructive aspects, but at the cost of a further increase of the constant factors [12]. Combining the graph minor results of Robertson and Seymour with a result of Lagergren [22], one gets $O(nlog^2n)$ decision algorithms (or $O(log^3n)$ algorithms on a CRCW PRAM with $O(n)$ processors), and combining these graph minor results with results of Arnborg et. al. [3], one gets decision algorithms, which are based on graph rewriting techniques, that either use $O(n)$ time but $O(n^2)$ memory, or $O(n \log n)$ time and linear memory. Recently, Fellows and Abrahamson [18] showed that more efficient $O(n^2)$ algorithms can be obtained for deciding whether a given graph has pathwidth $\leq k$, k constant, using notions from finite state automata theory. However, they do not solve the construction variant, and do not deal with the problem for treewidth. It should also be remarked, that for $k = 1, 2, 3$, linear time algorithms exist [5, 26].

The main result of this paper is the following: we give, for each constant k, an $O(nlog^2n)$ algorithm that, given a graph $G = (V, E)$, decides whether the treewidth of G is at most k (or whether the pathwidth of G is at most k), and if so, constructs a tree-decomposition (or path-decomposition) with treewidth (pathwidth) at most k. In case a tree-decomposition of G with constant, but possibly not optimal, treewidth is known, our algorithms take linear time. We also discuss parallel versions of the algorithms. Our algorithms do not use non-constructive arguments, or graph minors, but instead are given directly, and need only constructive combinatorial arguments for their correctness. The constant factor, although still fast growing with k, is *only* singly exponential in k.

Strongly related results were obtained independently by Lagergren and Arnborg [24, 23].

2 Definitions and Preliminary Results

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 collection of subsets of V, and $T = (I, F)$ a tree, such that

- $\bigcup_{i \in I} X_i = V$

- for all edges $(v,w) \in E$ there is an $i \in I$ with $v, w \in X_i$
- 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 $G = (V,E)$ is the minimum treewidth over all tree-decompositions of G.

Definition. A path-decomposition of a graph $G = (V,E)$ is a sequence $(X_1, X_2, \cdots, X_r)$ of subsets of V, such that

- $\bigcup_{1 \leq i \leq r} X_i = V$
- for all edges $(v,w) \in E$ there is an i, $1 \leq i \leq r$ with $v, w \in X_i$
- for all i, j, k with $1 \leq i < j < k \leq r$: $X_i \cap X_k \subseteq X_j$.

The pathwidth of a path-decomposition $(X_1, \cdots, X_r)$ is $\max_{1 \leq i \leq r} |X_i| - 1$. The pathwidth of a graph $G = (V,E)$ is the minimum pathwidth over all path-decompositions of G.

Recently, Lagergren obtained the following result:

Theorem 2.1 (Lagergren [22]) *For each k, there exists an algorithm, that uses $O(n \log^2 n)$ (sequential) time, or $O(log^3 n)$ time on a CRCW PRAM with $O(n)$ processors, that given a graph $G = (V,E)$, either finds a tree-decomposition of G with treewidth $\leq 6k+5$, or determines that the treewidth of G is more than k.*

With a simple trick, we can improve the bound of $6k+5$ to $3k+3$. (We omit the proof in this abstract.)

In order to make it easier to present our algorithms, we introduce a special type of tree-decompositions, called *nice* tree-decompositions; a tree-decomposition is nice, if T can be seen as a rooted tree, with every node having at most two children, such that for all $i \in I$, if i has two children j and k in T, then $X_i = X_j = X_k$, and if i has one child j in T, then X_i and X_j differ in exactly one element.

Lemma 2.2 *Every graph $G = (V,E)$ with treewidth k has a nice tree-decomposition with treewidth k. Moreover, when a tree-decomposition of G with treewidth k is given, one can construct in linear time a nice tree-decomposition of G with treewidth k.*

The first step of our algorithms is to find a nice tree-decomposition of the input graphs G with constant bounded treewidth. We denote the treewidth of this tree-decomposition by l. Clearly, if the algorithm of theorem 2.1 outputs that the treewidth of G is larger than k, we are done.

Now consider a nice tree-decomposition $(\{X_i \mid i \in I\}), T = (I,F))$. For each node $i \in I$, let $V_i \subseteq V$ be the set of all vertices $v \in V$, that belong to at least one X_j with j a descendant of i in T, or belong to X_i; and let G_i be the subgraph of G, induced by V_i.

Now, if i has one child j, then either G_i is obtained by adding a vertex v to G_j (in this case, the neighbors of v in V_i must belong to X_i), or G_i and G_j are equal (in this case, $X_i \subset X_j$). If i has two children j_1 and j_2, then G_i can be obtained by first taking the disjoint union of G_{j_1}, and G_{j_2}, and then identifying vertices in X_{j_1} and X_{j_2} pairwise. (Here we use the definition of tree-decomposition.)

Let a $(\leq K)$-boundary graph be a pair $(H = (W,F), X)$, with H a graph, and $X \subseteq W$, $|X| \leq K$. Thus, G can be obtained by a sequence of the following operations, on $(\leq l+1)$-boundary graphs. (This method is only slightly different from that of Wimer [31], and can also be put into the framework of Courcelle (see e.g. [15]. See also [18].)

- **Start.** Take a $(\leq l+1)$-boundary graph $(H = (X,F), X)$, with $|X| \leq l+1$.

- **Forget.** Given a $(\leq l+1)$-boundary graph (H, X), take a vertex $v \in X$, and take the $(\leq l+1)$-boundary graph $(H, X-\{v\})$.
- **Introduce.** Given a $(\leq l+1)$-boundary graph $(H=(W,F),X)$ with $|X| \leq l$, take a vertex $v \notin W$, a subset $X' \subseteq X$, and then take the $(\leq l+1)$-boundary graph $(H' = (W \cup \{v\}, F \cup \{(v,w) \mid w \in X'\}), X \cup \{v\})$.
- **Join.** Given $(\leq l+1)$-boundary graphs $(H_1 = (V_1, F_1), X)$, $(H_2 = (V_2, F_2), X)$, with $V_1 \cap V_2 = X$, take the $(\leq l+1)$-boundary graph $H = (V_1 \cup V_2, F_1 \cup F_2, X)$.

Many algorithms for graphs with bounded treewidth can be described in the following way. To graphs G_i, one can associate 'partial solutions' (in our case: tree-decompositions or path-decompositions of G_i with treewidth or pathwidth at most k). A possible, but infeasible algorithm to solve the problem is to compute for each $i \in I$ all partial solutions. If the root of T has at least one partial solution, then this is a 'total solution', i.e. the desired solution to the problem. As the number of partial solutions usually is much to large, an equivalence relationship is defined on partial solutions. This equivalence relationship must satisfies the requirement, that partial solutions are equivalent, if and only if either both or neither can be extended to a total solution. Further, the number of equivalence classes per node $i \in I$ must be bounded by some constant. Then, a linear time algorithm to solve the problem can consist of computing for all $i \in I$, all equivalence classes that contain at least one partial solution. Tables of these equivalence classes belonging to a node i, are computed from the tables associated with the children of i, in constant time per table.

In some cases, not all equivalence classes containing partial solutions must be calculated; e.g., if equivalence classes c_1 and c_2 both contain partial solutions, and if partial solutions in c_1 can be extended to total solutions, then partial solutions in c_2 can be extended to total solutions, then it is sufficient to only have c_2 and not c_1 in the table of partial solutions. (We say: c_2 *dominates* c_1.)

Many algorithms on graphs with bounded treewidth (e.g., those in [1, 6, 9, 29, 31]) are based upon the above described principle (although usually with different details and terminology). We use this principle also in this paper.

3 A decision algorithm for pathwidth

Suppose we have a nice tree-decomposition $\{X_i \mid i \in I\}, T = (I, F))$ of input graph $G = (V, E)$ with treewidth $\leq l$. We give an algorithm to decide whether the pathwidth of G is at most k. k and l are considered to be constants.

A *partial path-decomposition, rooted at i* is a path-decomposition of G_i with pathwidth $\leq k$.

We say that path-decomposition $(Y_1, Y_2, \cdots, Y_r)$ is an *extension* of path-decomposition $(Z_1, Z_2, \cdots, Z_s)$, if the latter can be obtained by skipping sets Y_j, and removing some vertices from all remaining sets $Y_{j'}$, i.e., if there is a set S, and an increasing function $f : \{1, 2, \cdots, s\} \to \{1, 2, \cdots, r\}$ with for all j: $Z_j = Y_{f(j)} \cap S$.

We need a few more definitions and results before we describe our table of partial solutions. We omit the proofs in this abstract.

Definition. For any integer sequence $a(1 \ldots n)$ let $l(a) = n$ be the length, and let $max(a)$ be the maximum value (i.e. $max(a) = \max_{1 \leq i \leq n} a_i$). For two sequences a and b *of the same length* we define the sum $c = a + b$ as the sequence c with $\forall_{1 \leq i \leq l(a)}[c_i = a_i + b_i]$. For a *constant* A we write $a + A$ for the sequence with $(a + A)_i = a_i + A$ for all i.

Definition. For an integer sequence $a(1 \ldots n)$ we define the *typical sequence*, $C(a)$, as the limit sequence of the following recursively applied operation:

> If the current sequence contains two elements a_i and a_j such that $j - i > 2$ and $\forall_{i<k<j}[a_k \geq \max(a_i, a_j)]$, then replace the subsequence $a(i+1 \ldots j-1)$ by its maximum.

Lemma 3.1 *Let $a(1\ldots n)$ be an integer sequence with $max(a) = K$. Then $max(C(a)) = K$ and $l(C(a)) \leq 4K + 3$.*

Definition. Let $a(1\ldots n)$ be a sequence. We define $E(a)$ as the set of *extensions* of a: $E(a) = \{a^* \mid \exists_{1=t_1<t_2<\ldots<t_{n+1}} \forall_{1\leq i\leq n} \forall_{t_i \leq k < t_{i+1}} [a^*_k = a_i]\}$. (In other words, each element of $E(a)$ is of the form $a_1, \ldots, a_1, a_2, \ldots, a_2, \cdots, a_n, \ldots, a_n$, each a_i appearing at least once in the sequence.)

Definition. Let $a(1\ldots n)$ and $b(1\ldots m)$ be two integer sequences. The *ringsum* $a \oplus b$ is defined as: $a \oplus b = \{a^* + b^* | a^* \in E(a) \wedge b^* \in E(b) \wedge l(a^*) = l(b^*)\}$.

Definition. For two integer sequences a and b, write $a \prec b$ if and only if there are $a^* \in E(a)$ and $b^* \in E(b)$ of the same length such that $a^*_i \leq b^*_i$ for all i. If both $a \prec b$ and $b \prec a$ hold we write $a \equiv b$.

Lemma 3.2 *For any integer sequence a: $C(a) \equiv a$.*

Definition. For any sequence $Y = (Y_1, Y_2, \ldots, Y_r)$ define a sequence $1 = t_1 < t_2 < \ldots < t_{q+1} = r+1$ by: $\forall_{1\leq i\leq q} \forall_{t_i \leq s < t_{i+1}} [Y_s = Y_{t_i}] \wedge \forall_{1 \leq i < q} [Y_{t_{i+1}} \neq Y_{t_i}]$. The *interval model* of Y is the sequence $(Y_{t_i})_{1\leq i\leq q}$. In other words: remove all sets Y_i with $Y_i = Y_{i-1}$.

Definition. Let $Y = (Y_1, Y_2, \ldots, Y_r)$ be a partial path-decomposition rooted at i. Define $Y^* = (Y_1 \cap X_i, Y_2 \cap X_i, \ldots, Y_r \cap X_i)$. The *characteristic* of Y is a pair $((Y^*_{t_i})_{1\leq i\leq q^*}, (\delta_i)_{1\leq i \leq q^*})$, where $(Y^*_{t_i})_{1\leq i\leq q^*}$ is the interval model for Y^* and δ_i is the typical sequence of $(|Y_{t_i}|, \ldots, |Y_{t_{i+1}-1}|)$.

Notice that the interval model for Y^* is a path-decomposition for the subgraph induced by X_i. The characteristics define the equivalence classes as described in section 2: one can show that two partial path-decompositions with the same characteristic can either both be extended to a path-decomposition of G with pathwidth $\leq k$, or neither of these can be extended to such a path-decomposition.

Definition. For two partial path-decompositions A and B, rooted at i with characteristics $c_A = ((A^*_{t_i})_{1\leq i\leq q}, (\alpha_i)_{1\leq i\leq q})$ and $c_B = ((B^*_t)_{1\leq t\leq q}, (\beta_t)_{1\leq t\leq q})$ we write $A \prec B$ ('A dominates B'), if and only if the interval models are the same and for all typical sequences we have $\alpha_t \prec \beta_t$. If both $A \prec B$ and $B \prec A$ hold we write $A \equiv B$.

Lemma 3.3 *For every partial path-decomposition Y with characteristic $((Y^*_{t_i})_{1\leq i\leq q}, (\delta_i)_{1\leq i\leq q})$, there exists a partial path-decomposition Y'* with the same characteristic *such that for all i, $(|Y'_{t_i}|, \ldots, |Y'_{t_{i+1}-1}|) \in E(\delta_i)$.*

Definition. A set of characteristics S of partial path-decompositions rooted at some node i is called a *full set of characteristics* of node i, if for each partial path-decomposition rooted at i, either its characteristic is in S, or it is dominated by a path-decomposition with its characteristic in S.

Theorem 3.4 *Given a full set of characteristics of each of the children of node $p \in I$, a full set of characteristics of p can be computed in $O(1)$ time.*

Proof.
We must consider four cases, one for each of the rules **Start, Forget, Introduce, Join.**

The first and easiest case is that (G_p, X_p) is obtained by a **Start** operation. In this case, enumerate all partial path-decompositions of $G[X_p]$ with for all $i : Y_i \neq Y_{i+1}$, (the number of these is bounded by some constant) and make a list of all different characteristics.

The second case is that (G_p, X_p) is obtained by a **Forget** operation. It is also easy to see that the claim holds in this case.

Next consider the case in which (G_p, X_p) is obtained from (G_q, X_q) and (G_r, X_r) by a **Join** operation. Recall that $X_p = X_q = X_r$ in this case. Consider a partial path-decomposition $Y =$

$(Y_1,\ldots,Y_f)$ rooted at p. If we define $A_s = Y_s \cap V_q$ and $B_s = Y_s \cap V_r$ for all $1 \le s \le f$ then we obtain partial path-decompositions $A = (A_1,\ldots,A_f)$ rooted at q and $B = (B_1,\ldots,B_f)$ rooted at r. We write $a_s = |A_s|$, $b_s = |B_s|$ and $y_s = |Y_s|$ for all s. By assumption, we know that there exist A_0 and B_0 with $A_0 \prec A$ and $B_0 \prec B$ and such that the characteristics of A_0 and B_0 are in the full set of characteristics of q and r respectively. We show that there exists a partial path-decomposition $Y_0 \prec Y$ of which the characteristic can be computed from the characteristics of A_0 and B_0. Consider the characteristic of Y, say $((Y^*_{t_s})_{1\le s\le w}, (\gamma_s)_{1\le s\le w})$. Let $((A^*_{t'_s})_{1\le s\le w}, (\alpha_{t'_s})_{1\le s\le w})$ and $((B^*_{t''_s})_{1\le s\le w}, (\beta_{t''_s})_{1\le s\le w})$ be the characteristics for A and B respectively. The interval models of these three characteristics are the same.

Consider an interval $[t_s, t_{s+1})$. We know that $y_i = a_i + b_i - |Y^*_{t_s}|$ holds in this interval. We write $a' = a(t_s \ldots t_{s+1} - 1)$ and define b' and y' similary. Since $\alpha_s \equiv a'$ and $\alpha_{0,s} \prec \alpha_s$ (where $\alpha_{0,s}$ is the corresponding typical sequence of A_0), we know that also $\alpha_{0,s} \prec a'$ holds. So there are extensions $a^* \in E(a')$ and $\alpha^*_{0,s} \in E(\alpha_{0,s})$ such that $\alpha^*_{0,s,i} \le a^*_i$ holds for all i. The same applies to the subsequence b'. But then we may conclude that there also is a sequence $y_{0,s}$ with $y_{0,s} \prec y'$ and $y_{0,s} + |Y^*_{t_s}| \in \alpha_{0,s} \oplus \beta_{0,s}$. In this way we can find a partial path-decomposition $Y_0 \prec Y$ for node p. Notice that the different typical sequences of $\alpha_{0,s} \oplus \beta_{0,s}$ can be computed in constant time.

This gives the following procedure to compute the full set of characteristics for p: for each pair of characteristics $A^1 = ((A_s)_{1\le s\le w}, (\alpha_s)_{1\le s\le w})$ in the full set of q, and $A^2 = ((A_s)_{1\le s\le w}, (\beta_s)_{1\le s\le w})$ in the full set of r (i.e. with the same interval model), add to the full set of p all possible characteristics of the form $((A_s)_{1\le s\le w}, (\gamma s)_{1\le s\le w})$, with each γ_s the typical sequence of a sequence in $\{\delta - |A_s| \mid \delta \in \alpha_s \oplus \beta_s\}$. When we have a partial path-decomposition with characteristic A_1, and one with characteristic A_2, one can form for each of the above defined characteristics, the corresponding path-decomposition quite easy.

Finally, assume that (G_p, X_p) is obtained from (G_q, X_q) by an **Introduce** operation. Let $\{x\} = X_i \setminus X_j$. Consider a partial path-decomposition $Y = (Y_1,\ldots,Y_f)$ rooted at p and let $A = (A_1,\ldots,A_f)$ be the partial path-decomposition rooted at q obtained from Y by deleting the vertex x. Let the characteristic of A be $((A^*_{t_s})_{1\le s\le w}, (\alpha_s)_{1\le s\le w})$. By assumption we know that there exists a partial path-decomposition A_0 rooted at q with $A_0 \prec A$ and of which the characteristic is in the full set of characteristics of q. We may assume that $(|A_{0,t_i}|,\ldots,|A_{0,t_{i+1}-1}|) \in E(\alpha_{0,i})$, where the $\alpha_{0,i}$'s are the typical sequences in the characteristic of A_0. Let $A^*_{t_s}$ be the first interval in which x is included and assume for simplicity that the last interval in which x is included is a different interval. So we know that the interval $[t_s, t_{s+1})$ is split into two intervals in Y, say $[t_s, z)$ and $[z, t_{s+1})$. Write $a' = a(t_s \ldots t_{s+1} - 1)$. Let $a_{0,s} = (|A_{0,t_s}|,\ldots,|A_{0,t_{s+1}-1}|)$. We know that there exist extensions $a^*_{0,s} \in E(a_{0,s})$ and $a^* \in E(a')$ such that $a^*_{0,s,i} \le a^*_i$ holds for all i (because $a_{0,s} \equiv \alpha_{0,s} \prec \alpha_s \equiv a'$). Let z^* be a position in a^* corresponding to position z in a'. Now split the sequence $a^*_{0,s}$ at z^* and add one to every element of the last part obtaining two sequences δ_1 and δ_2. The typical sequences for δ_1 and δ_2 can be computed from $\alpha_{0,s}$. For the last interval where x is added, a similar split is made; for all intervals in between one must be added to each number (because x is added to the corresponding sets). In the corresponding extension A^*_0 of A_0, vertex x is added to the prescribed sets. In this way we obtain a partial path-decomposition $Y_0 \prec Y$ for node p, of which the characteristic can be computed from the characteristic of A_0 in constant time.

Basically, we obtain the full set for p as follows: for each characteristic in the full set of q, make in two intervals a 'split' as described above, add 1 to all typical sequences for the intervals in between, add x to all sets where 1 was added to the characteristic sequences, but check in the interval model whether adjacencies of x to vertices in X_q are satisfied. In the full paper we give the precise construction and a full proof. □

Now note that a partial path-decomposition rooted at the node i_r, that is the root of T, is in fact, a path-decomposition of G with pathwidth at most k. Hence, we have the following decision algorithm for the 'pathwidth $\le k$' problem: for all $i \in I$, compute a full set of characteristics — starting at leaf nodes of T, etc. Finally, check whether the full set of the root node of T is non-empty.

Theorem 3.5 *For each $k, l \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$ with a tree-decomposition of G with treewidth at most l, determines whether the pathwidth of G is at most k, and that uses $O(|V|)$ time.*

Combining this result with theorem 2.1, we get:

Theorem 3.6 *For each $k \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$ determines whether the pathwidth of G is at most k, and that uses $O(|V| \log^2 |V|)$ time.*

4 A decision algorithm for treewidth

In this section we sketch how the algorithm of the previous section can be modified to an algorithm for the 'treewidth $\leq k$' problem for constant k. The main techniques are similar, but we need a different, and slightly more complex notion of characteristic. We start with some definitions.

A tree-decomposition $(\{Z_i \mid i \in \mathcal{I}'\}, \mathcal{T}' = (\mathcal{I}', \mathcal{F}'))$ is an *extension* of tree-decomposition $(\{Y_i \mid i \in \mathcal{I}\}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$, if the latter can be obtained from the former by a sequence of the following operations: remove a leaf node from $\mathcal{T}'$; take a node $j \in \mathcal{I}'$ that has degree 2 in (the current version of) $\mathcal{T}'$ and remove j from $\mathcal{T}'$ and connect its neighbors; take a vertex $v \in \cup_{j \in \mathcal{I}'} Z_j$, and remove v from all sets Z_j.

Consider some fixed node $i \in I$. Consider a tree-decomposition $(\{Y_j \mid j \in \mathcal{I}\}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ of G_i with treewidth at most k. For each $j \in \mathcal{I}$, we look at $X_i \cap Y_j$. A set $X \subseteq X_i$ is called *maximal in the tree-decomposition*, if there exists an $j \in \mathcal{I}$ with $X = X_i \cap Y_j$, but for no Y, $X \subset Y \subseteq X_i$, there exists an $j \in \mathcal{I}$ with $Y = X_i \cap Y_j$.

For a set X, that is maximal in the tree-decomposition, let $\mathcal{T}_X$ be the subtree of $\mathcal{T}$, formed by the nodes $\{j \in \mathcal{I} \mid X = X_i \cap Y_j\}$. ($\mathcal{T}_X$ is a tree by definition of tree-decomposition.)

A node $j \in \mathcal{I}$ is called a *reference node*, if one of the two following cases holds

- j is adjacent to at least three different edges in $\mathcal{T}$, that are the first edge on a path from j to a node j' in some tree $\mathcal{T}_X$, with j not in $\mathcal{T}_X$.
- j belongs to a subtree $\mathcal{T}_X$ for some X maximal in the tree-decomposition, and there is a path in $\mathcal{T}$ from j to a node j' in another subtree $\mathcal{T}_{X'}$ for $X' \neq X$ that is maximal in the tree-decomposition, and this path does not use other nodes in $\mathcal{T}_X$.

It follows from the definition of tree-decomposition that there are at most $k+1$ sets maximal in the tree-decomposition, and that there are at most $O(k)$ reference nodes.

The *trunk* of $\mathcal{T}$ is the set of nodes $j \in \mathcal{I}$ that are either a reference node or are on the path between two reference nodes.

Lemma 4.1 *If there exists a tree-decomposition $(\{Z_j \mid j \in \mathcal{I}'\}, \mathcal{T}' = (\mathcal{I}', \mathcal{F}'))$ of G with treewidth at most k, that is an extension of the tree-decomposition $(\{Y_j \mid j \in \mathcal{I}\}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ of G_i, then there exists such a tree-decomposition of H such that for all nodes $j \in \mathcal{I} (\subseteq \mathcal{I}')$ that are not on the trunk of $\mathcal{T}$: $Z_j = Y_j$.*

In other words, when we try to extend a tree-decomposition, we may assume that nodes not on the trunk are not affected. The proof of this lemma relies on the observation that vertices that are in G_i but not in X_i are not adjacent to vertices in $H_i \setminus G_i$. The lemma shows, that it is sufficient to have a characteristic of the 'trunk of the tree-decomposition'. The characteristic of a tree-decomposition $(\{Y_j \mid j \in \mathcal{I}\}, \mathcal{T} = (\mathcal{I}, \mathcal{F}))$ of (G_i, X_i) is obtained in the following way:

Take the trunk.
Replace each path between two reference nodes that does not contain another reference node by an edge.
Label this edge by the characteristic (as defined for path-decompositions in Section 3) of the sequence of sets Y_α, obtained by going from j_1 to j_2.

With this notion of characteristic, we can proceed in the same way as in Section 3 for the case of pathwidth: the notion of *full set of characteristics* is defined in the same way, one can show that the size of a full set is bounded by a constant, exponential in k, and one can show the equivalent of theorem 3.4 for the case of treewidth. Again, if the full set of the root of T is non-empty, then there exists a tree-decomposition of G with treewidth $\leq k$.

Theorem 4.2 *(i) For each $k, l \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$ with a tree-decomposition of G with treewidth at most l, determines whether the treewidth of G is at most k, and that uses $O(|V|)$ time.*
(ii) For each $k \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$ determines whether the pathwidth of G is at most k, and that uses $O(|V| \log^2 |V|)$ time.

5 Turning decision algorithms into construction algorithms

In the previous sections, we showed how to obtain decision algorithms for the 'pathwidth $\leq k$' and 'treewidth $\leq k$' problems. It is not hard to turn these decision algorithms into construction algorithms, using the constructiveness of our arguments in Sections 3 and 4, and stardard techniques to make dynamic programming algorithms that construct solutions. Observe the following.

Each characteristic in a full set of characteristics for a leaf node $i \in I$ corresponds directly to a path- or tree-decomposition on X_i with that characteristic.

For internal nodes $i \in I$ where a **Forget** or **Introduce** operation takes place, each characteristic in a full set of characteristics of i is made from a characteristic in the full set of the child j of i. In the former case, the corresponding path- or tree-decomposition does not change. In case of an **Introduce** operation, we must add the unique vertex $v \in X_i \setminus X_j$ to a number of sets Y_α, and possibly we must add one or more new sets $Y_\alpha = \{v\}$ to the decomposition. However, in which way this must be done can be described precisely — cf. Section 3 (and 4). From the construction it follows that a tree- or path-decomposition with the required characteristic can be computed in time linear in the increase of $\sum_{i \in I} |Y_i|$, i.e., in the increase of the size of the description of the tree- or path-decomposition.

A similar argument can be made for nodes i where a **Join** operation takes place. Each characteristic in the full set of characteristics of i corresponds to one characteristic in each of the full sets of the two children j and k of i. The corresponding tree- or path-decomposition can be made by 'composing' the two decompositions corresponding to these two chararcteristics in the full sets of j and k. This composition can be done in time, linear in the size of the smallest of the two graphs that are joined.

We now describe our construction algorithms. First do the decision algorithms, as outlined in Section 3 and 4, but for each node i, for each characteristic s in the computed full set, store pointers to characteristics in the full sets of the children of i, such that a tree- or path-decomposition corresponding to s can be made from the decompositions corresponding to the characteristics with a pointer from s to it. (See discussion above.) In case the treewidth or pathwidth of G is at most k, continue by taking a characteristic s_0 in the full set of the root of T. Compute recursively a decomposition with treewidth or pathwidth at most k for each characteristic s_0 has a pointer to. Then, as described above, a tree- or path-decomposition, corresponding to s_0, i.e. a tree- or path-decomposition of G, can be computed.

Note that we must compute only one (tree- or path-)decomposition per node $i \in I$. The extra time, needed for the construction, is linear in the size of the description of the tree-decomposition and (hence) in the size of G.

Theorem 5.1 *For each $k, l \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$ with a tree-decomposition of G with treewidth at most l, determines whether the pathwidth (treewidth) of G is at most k, and if so, finds a path-decomposition (tree-decomposition) of G with pathwidth (treewidth) at most k, and that uses $O(|V|)$ time.*

Theorem 5.2 *For each $k \in N^+$, there exists an algorithm, that given a graph $G = (V, E)$, determines whether the pathwidth (treewidth) of G is at most k, and if so, finds a path-decomposition (tree-decomposition) of G with pathwidth (treewidth) at most k, and that uses $O(|V| \cdot \log^2 |V|)$ time.*

6 Final remarks

6.1 Parallel algorithms

In [10] it was shown that for fixed k, the problem of deciding whether the treewidth of a given graph is at most k is in NC. However, this algorithm uses a large number of processors, namely $O(n^{3k+4})$. Chandrasekharan and Hedetniemi [14] improved on this result, but the number of processors of their algorithm is still too large for practical purposes ($O(n^{2k+6})$). Combining Lagergrens result (theorem 2.1) with graph minor theory of Robertson and Seymour, one gets a proof of existence of decision algorithms for the 'treewidth $\leq k$' and 'pathwidth $\leq k$' problems that use $O(n)$ processors on a CRCW PRAM, and $O(\log^3 n)$ time. However, this approach suffers from drawbacks associated with the graph minor theory: the very large hidden constant factors, and the non-constructiveness.

Combining Lagergrens result with the techniques of Sections 3 and 4, we can give direct algorithms.

First we remark, that the tree T yielded by the algorithm of theorem 2.1 has depth $O(\log n)$. When we transform it to a nice tree-decomposition, we can do this such that the depth of T still is $O(\log n)$.

Now the algorithm, lined out in earlier sections in this paper can be used — but now we process different branches of the tree T in parallel, e.g., we compute all full sets for nodes with maximum depth in T simultaneously, then we compute all full sets for nodes with maximum depth -1 in T, etc. In this way, after $O(depth(T)) = O(\log n)$ steps, a full set of characteristics of the root node of T is computed. This gives am $O(\log^3 n)$ algorithm using $O(n)$ processors on a CRCW PRAM for the decision problems.

A close observation of the construction algorithms (cf. Section 5), shows that in the same time/processor bounds the corresponding decompositions can be made.

Theorem 6.1 *For each k, there exists an algorithm, that uses $O(log^3 n)$ time on a CRCW PRAM with $O(n)$ processors, that given a graph $G = (V, E)$, determines whether the pathwidth (treewidth) of G is at most k, and if so, finds a path-decomposition (tree-decomposition) of G with treewidth at most k.*

6.2 Attacking the constants

The constant factors of the algorithms, described in this paper, are still quite large (although 'only singly exponential in k'.) Note that, due to the NP-completeness of the problems for non-bounded k, we should not expect to do better than worst case running time exponential in k. However, we think it most likely that with optimizations, the algorithms can be made practical for small values of k. There are several approaches to do this:

- Use modified characterizations, and/or modified graph building rules.
- Use domination or stronger forms of domination, and try to remove characteristics in a full set that are dominated by another characteristic in the full set.
- Use (Myhill-Nerode) state reduction techniques. Bern et. al. [7] discuss how to do this in a slightly similar setting. Fellows et. al. have studied this idea further [18, 19]. With these techniques, further equivalences are defined on the characteristics, such that a minimum number of equivalence classes results. It must be studied, whether the construction of corresponding tree- or path-decompositions can be carried over with the new equivalences.
- Use 'memoization'; i.e., do not compute full sets at once, but always try to find characteristics that can be included in a (not yet full) set of a node i that is as close to the root as possible. As soon as we find a characteristic in the set of the root of T we are done. This approach may help to significantly decrease the average case time for inputs with a positive answer to the decision problem, but it does not significantly increase the worst case time. However, it also will not improve the time for 'negative' inputs.
- Try to find algorithms, similar as the one in theorem 2.1, that give better approximations on the treewidth. It is feasible to have an algorithm that uses several steps, each step improving a little on the treewidth, only the last one yielding tree- or path-decompositions with optimal tree- or pathwidth.
- Combine the results of this paper with graph rewriting techniques, as in [5, 3].

Probably it will be useful for most research in this direction, when algorithms are actually implemented and tested.

6.3 Other problems

Clearly, the results in this paper can be transformed to similar results for notions, equivalent to treewidth or pathwidth (see section 1.) We believe that the techniques in this paper can also yield similar results for related notions, like: search number, topological bandwidth, minimum cut linear arrangement, etc. Results for the decision variants of some of these problems are announced in [18].

References

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

[2] 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.

[3] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. Technical Report 90-02, Laboratoire Bordelais de Recherche en Informatique, Bordeaux, 1990. To appear in Proceedings 4th Workshop on Graph Grammars and Their Applications to Computer Science.

[4] S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs (extended abstract). In *Proceedings of the 15'th International Colloquium on Automata, Languages and Programming*, pages 38–51. Springer Verlag, Lect. Notes in Comp. Sc. 317, 1988. To appear in J. of Algorithms.

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

[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] M. W. Bern, E. L. Lawler, and A. L. Wong. Linear time computation of optimal subgraphs of decomposable graphs. *J. Algorithms*, 8:216–235, 1987.

[8] H. L. Bodlaender. Classes of graphs with bounded treewidth. Technical Report RUU-CS-86-22, Dept. of Computer Science, Utrecht University, Utrecht, 1986.

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

[10] H. L. Bodlaender. NC-algorithms for graphs with small treewidth. In J. van Leeuwen, editor, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science WG'88*, pages 1–10. Springer Verlag, LNCS 344, 1988.

[11] H. L. Bodlaender. Complexity of path forming games. Technical Report RUU-CS-89-29, Utrecht University, Utrecht, 1989.

[12] 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.

[13] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear algorithms from predicate calculus descriptions of problems on recursive constructed graph families. Manuscript, 1988.

[14] N. Chandrasekharan and S. T. Hedetniemi. Fast parallel algorithms for tree decomposing and parsing partial k-trees. In *Proc. 26th Annual Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, Illinois, 1988.

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

[16] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[17] J. A. Ellis, I. H. Sudborough, and J. Turner. Graph separation and search number. Report DCS-66-IR, University of Victoria, 1987.

[18] M. R. Fellows and K. R. Abrahamson. Cutset regularity beats well-quasi-ordering for bounded treewidth. Manuscript, 1990.

[19] 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.

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

[21] A. Habel. *Hyperedge Replacement: Grammars and Languages.* PhD thesis, Univ. Bremen, 1988.

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

[23] J. Lagergren. *Algorithms and Minimal Forbidden Minors for Tree-decomposable Graphs.* PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 1991.

[24] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. To appear in: proceedings ICALP'91.

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

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

[27] R. H. Möhring. Graph problems related to gate matrix layout and PLA folding. Technical Report 223/1989, Technical University of Berlin, 1989.

[28] N. Robertson and P. D. Seymour. Graph minors — a survey. In I. Anderson, editor, *Surveys in Combinatorics*, pages 153–171. Cambridge Univ. Press, 1985.

[29] P. Scheffler. Linear-time algorithms for NP-complete problems restricted to partial k-trees. Report R-MATH-03/87, Karl-Weierstrass-Institut Für Mathematik, Berlin, GDR, 1987.

[30] L. C. van der Gaag. *Probability-Based Models for Plausible Reasoning.* PhD thesis, University of Amsterdam, 1990.

[31] T. V. Wimer. *Linear algorithms on k-terminal graphs.* PhD thesis, Dept. of Computer Science, Clemson University, 1987.