Treewidth and pathwidth of permutation graphs

H. Bodlaender, T. Kloks, D. Kratsch

RUU-CS-92-30
September 1992

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Treewidth and Pathwidth of Permutation Graphs

H. Bodlaender* T. Kloks †
Department of Computer Science, Utrecht University
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands

D. Kratsch ‡
Fakultät Mathematik, Friedrich-Schiller-Universität
Universitätshochhaus, O-6900 Jena, Germany

Abstract

In this paper we show that the treewidth and pathwidth of a permutation graph can be computed in polynomial time. In fact we show that, for permutation graphs, the treewidth and pathwidth are equal. These results make permutation graphs one of the few non-trivial graph classes for which at the moment, treewidth is known to be computable in polynomial time. Our algorithm to decide whether the treewidth (pathwidth) is at most some given integer k, can be implemented to run in $O(nk^2)$ time, when the matching diagram is given. We show that this algorithm can easily be adapted to compute the treewidth of a permutation graph in $O(nk^2)$ time, where k is the treewidth.

1 Introduction

In many recent investigations in computer science, the notions of treewidth and pathwidth play an increasingly important role. One reason for this is that many problems, including many well studied NP-complete graph problems, become solvable in polynomial and usually even linear time, when restricted to the class of graphs with bounded tree- or pathwidth [1, 4, 3, 10, 7, 23]. Of crucial importance for these algorithms is, that a tree-decomposition or path-decomposition of the graph is given

*Email: hansb@cs.ruu.nl

†This author is supported by the foundation for Computer Science (S.I.O.N.) of the Netherlands Organisation for Scientific Research (N.W.O.). Email: ton@cs.ruu.nl

‡Email: DIETER.KRATSCH@mathematik.uni-jena.dbp.de

ISSN: 0924-3275

in advance. Much research has been done in finding efficient algorithms computing a tree-decomposition with a reasonable small treewidth. Recent results [25, 9, 22] show that an $O(n \log n)$ algorithm exists to find a suitable tree-decomposition for a graph with bounded treewidth. However, the constant hidden in the 'big oh', is exponential in the treewidth, limiting the practicality of this algorithm.

For some *special classes* of graphs, it has been shown that the treewidth can be computed efficiently. In this paper we discuss the problem of finding tree- and path-decompositions for permutation graphs. We also show that for these graphs, the treewidth and the pathwidth are the same.

Permutation graphs are a well-studied and well-characterized class of perfect graphs. They are exactly the comparability graphs of posets of dimension at most two. Moreover, permutation graphs are exactly those graphs for which both the graph and its complement is a comparability graph. They have many applications in scheduling problems. See for example [14] where permutation graphs are used to describe the memory requirements of a number of programs at a certain time (see also [17]). Permutation graphs also arise in a very natural way in the problem of sorting a permutation, using queues in parallel. In [17] it is shown that this problem is closely related to the coloring problem of permutation graphs. Other applications occur for example in VLSI layout (see e.g. [26]). There is a long list of papers, which mainly appeared in the last ten years, studying the algorithmic complexity of NP-complete graph problems when restricted to permutation graphs. Indeed, there exist fast algorithms for many NP-complete problems like CLIQUE, INDEPENDENT SET, FEEDBACK VERTEX SET and DOMINATING SET when restricted to permutation graphs [17, 15, 11, 12]. However some problems remain NP-complete, like COCHROMATIC NUMBER [29, 16], and ACHROMATIC NUMBER [6].

We give a $O(nk^2)$ time algorithm which determines whether the pathwidth is at most k, when the matching diagram is given. The algorithm can easily be adapted such that it computes the pathwidth of a permutation graph within the same time bound.

2 Preliminaries

In this section we start with some definitions and easy lemmas. For more information on perfect graphs the reader is referred to [17, 13, 5].

Definition 2.1 *A graph is* chordal *if it has no induced chordless cycle of length at least four.*

Chordal graphs are also called triangulated. There are basically two ways two define the treewidth of a graph. One way is to use the concept of a *tree-decomposition*. For more information on tree-decompositions the reader is referred to the survey paper [10]. In this paper we introduce the treewidth of a graph by means of k-trees.

Definition 2.2 *Let k be an integer. A k-tree is a graph which is defined recursively as follows. A clique with $k+1$ vertices is a k-tree. Given a k-tree T_n with n vertices, a k-tree with $n+1$ vertices can be constructed by making a new vertex adjacent to the vertices of a k-clique in T_n. A graph is a* partial k-tree *if either it has at most k vertices or it is a subgraph of a k-tree T with the same vertex set as T.*

k-Trees are chordal and have $\omega(G) = k+1$ (where $\omega(G)$ is the maximum clique size). Notice that any graph is a partial k-tree for some k; take k equal to the number of vertices minus one.

Definition 2.3 *The* treewidth *of a graph G is the minimum value k for which G is a partial k-tree.*

Definition 2.4 *A* triangulation *of a graph G is a graph H with the same vertex set as G, such that G is a subgraph of H and H is chordal.*

For a proof of the following lemma see for example [23].

Lemma 2.1 *A graph G has treewidth $\leq k$ if and only if there is a triangulation H of G with $\omega(H) \leq k+1$.*

It follows that the treewidth of a chordal graph is the maximum clique size minus one. The treewidth can be defined in terms of the minimum over all triangulations of the maximum clique size. We define the *pathwidth* of a graph using triangulations of a special kind.

Definition 2.5 *An* interval graph *is a graph of which the vertices can be put into one to one correspondence with intervals on the real line, such that two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection.*

There are many ways to characterize interval graphs. We state only one of the first characterizations [24].

Lemma 2.2 *An undirected graph is an interval graph if and only if the following two conditions are satisfied:*

1. *G is chordal and*
2. *any three vertices of G can be ordered in such a way that every path from the first to the third vertex passes through a neighbor of the second vertex.*

Three vertices which do not satisfy the second condition are called an *astroidal triple*. These are pairwise non adjacent and for any pair of them there is a path that avoids the neighborhood of the remaining vertex.

Definition 2.6 *Let k be an integer. A graph G has* pathwidth $\leq k$ *if and only if there is a triangulation H of G such that H is an interval graph with $\omega(H) \leq k+1$.*

Determining the treewidth or the pathwidth of a graph is NP-complete [2]. However, for *constant* k, graphs with treewidth $\leq k$ are recognizable in $O(n \log n)$ time [25, 9, 22]. The large constants involved in these algorithms make them usually not very practical. It is therefore of importance to find algorithms which are polynomial both in the number of vertices and in the treewidth, for TREEWIDTH and PATHWIDTH for special classes of graphs which are as large as possible.

One of the main reasons why there exist fast algorithms for many problems when restricted to graphs with bounded treewidth, is the existence of vertex separators of bounded size.

Definition 2.7 *A subset $S \subseteq V$ is a a,b-separator for nonadjacent vertices a and b, if the removal of S separates a and b in distinct connected components. If no proper subset of S is an a,b-separator then S is a minimal a,b-separator. A minimal separator S is a subset such that S is a minimal a,b-separator for some nonadjacent vertices a and b.*

The following lemma, which must have been rediscovered many times, appears for example as an exercise in [17].

Lemma 2.3 *Let S be a minimal a,b-separator, and let C_a and C_b be the connected components of $G[V-S]$, containing a and b respectively. Then every vertex of S has a neighbor in C_a and a neighbor in C_b.*

Theorem 2.1 *Let G be a partial k-tree. There exists a triangulation of G into a chordal graph H such that the following three statements hold:*

1. *$\omega(H) \leq k+1$.*
2. *If a and b are nonadjacent vertices in H then every minimal a,b-separator of H is also a minimal a,b-separator in G.*
3. *If S is a minimal separator in H and C is the vertex set of a connected component of $H[V-S]$, then C induces also a connected component in $G[V-S]$.*

Proof. Take a triangulation H, with treewidth at most k and with a minimum number of edges (this exists by lemma 2.1). Suppose H has a minimal vertex separator C for nonadjacent vertices a and b, such that either C induces no minimal a,b-separator in G, or the vertex sets of the connected components of $H[V-C]$ are different from those of $G[V-C]$. Let $S \subseteq C$ be a minimal a,b-separator in G. Let $C_1,\ldots,C_t$ be the connected components of $G[V-S]$. Make a chordal graph H' as follows. For each $C_i \cup S$ take the chordal subgraph of H induced by these vertices. Since S is a clique in H, this gives a chordal subgraph H' of H. Notice that the vertex sets of the connected components of $H'[V-S]$ are the same as those of $G[V-S]$. We claim that the number of edges of H' is less than the number of edges of H, which is a contradiction. Clearly, the number of edges of H' does not

exceed the number of edges of H. First assume that $S \neq C$, and let $x \in C \setminus S$. By lemma 2.3, in H, x has a neighbor in the component containing a and a neighbor in the component containing b. Not both these edges can be present in H'. Thus we may assume $S = C$. Then the vertex sets of the connected components of $H[V - C]$ are different from those of $H'[V - C]$. Since H' is a subgraph of H, every connected component $H'[V - C]$ is contained in some connected component of $H[V - C]$. It follows that there must be a connected component in $H[V - C]$ containing two different connected components of $H'[V - C]$. This can only be the case if there is some edge between these components in $H[V - C]$ (which is not there in $H'[V - C]$). This proves the theorem. □

Definition 2.8 *We call a triangulation of which the existence is guaranteed by theorem* 2.1, *a* minimal triangulation.

Let $G = (V, E)$ be a graph and let C be a minimal vertex separator. Let $C_1, \ldots, C_t$ be the connected components of $G[V - C]$. We denote by $\overline{C}_i$ $(i = 1, \ldots, t)$ the graph obtained as follows. Take the induced subgraph $G[C \cup C_i]$, and add edges such that the subgraph induced by C is complete. The following lemma easily follows from theorem 2.1 (a similar result appears in [2]).

Lemma 2.4 *A graph G with at least $k + 2$ vertices is a partial k-tree if and only if there exists a minimal vertex separator C such that the graphs $\overline{C}_i$ are partial k-trees $(i = 1, \ldots, t)$.*

In this paper we show that the treewidth and pathwidth of a permutation graph can be computed in polynomial time. We think of a permutation π of the numbers $1, \ldots, n$ as the sequence $\pi = [\pi_1, \ldots, \pi_n]$.

Definition 2.9 *If π is a permutation of the numbers $1, \ldots, n$, we can construct a graph $G[\pi] = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set E:*

$$(i, j) \in E \Leftrightarrow (i - j)(\pi_i^{-1} - \pi_j^{-1}) < 0$$

An undirected graph is a permutation graph *if there is a permutation π such that $G \cong G[\pi]$.*

The graph $G[\pi]$ is sometimes called the inversion graph of π. If the permutation is not given, it can be computed in $O(n^2)$ time [27, 17]. In this paper we assume that the permutation is given and we identify the permutation graph with the inversion graph. A permutation graph is an intersection graph, which is illustrated by the matching diagram.

Definition 2.10 *Let π be a permutation of $1, \ldots, n$. The matching diagram can be obtained as follows. Write the numbers $1, \ldots, n$ horizontally from left to right. Underneath, write the numbers $\pi_1, \ldots, \pi_n$, also horizontally from left to right. Draw straight line segments joining the two 1's, the two 2's, etc.*

Notice that two vertices i and j of $G[\pi]$ are adjacent if and only if the corresponding line segments intersect. In figure 1 (page 6) we give an example.

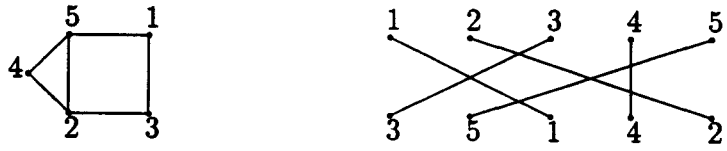

Figure 1: permutation graph and matching diagram

3 Scanlines

In this section we show that every minimal separator in a permutation graph can be obtained by using a *scanline*. Recall the definition of the matching diagram. It consists of two horizontal lines, one above the other, and a number of straight line segments, one for each vertex, such that each line segment has one end vertex on each horizontal line. Two vertices are adjacent, if the corresponding line segments intersect. We say that two line segments *cross* if they have a nonempty intersection.

Definition 3.1 *A* scanline *in the diagram is any line segment with one end vertex on each horizontal line. A scanline s is* between *two line segments x and y if the top point of s is in the open interval bordered by the top points of x and y and the bottom point of s is in the open interval bordered by the bottom points of x and y.*

If a scanline s is between line segments x and y then the intersection of each pair of the three line segments is empty. Consider two nonadjacent vertices x and y. The line segments in the diagram corresponding to x and y do not cross in the diagram. Hence we can find a scanline s between the lines x and y. Take out all the lines that cross the scanline s. Clearly this corresponds with an x,y-separator in the graph. The next lemma shows that we can find all minimal x,y-separators in this way.

Lemma 3.1 *Let G be a permutation graph, and let x and y be nonadjacent vertices in G. Every minimal x,y-separator consists of all line segments crossing a scanline which lies between the line segments of x and y.*

Proof. Let S be a minimal x,y-separator. Consider the connected components of $G[V-S]$. Let C_x be the component containing x and C_y be the component containing y. Clearly these must also be 'connected' parts in the diagram, and we may assume without loss of generality that the component containing x is completely to the left of the component containing y. Every vertex of S is adjacent to some vertex in C_x and to some vertex in C_y (lemma 2.3). Notice that we can choose a scanline s crossing no line segment of $G[V-S]$, and which is between x and y. Then all lines crossing the scanline must be elements of S. But for all elements of S the corresponding line segment must cross s, since it is intersecting with a line segment of C_x, which is to the left of s, and with a line segment of C_y, which is to the right of s. □

If s is a scanline, then we denote by S the set of vertices of which the corresponding line segments cross s. In the rest of this paper we consider only scanlines, of which the end points do not coincide with end points of other line segments.

Definition 3.2 *Two scanlines s_1 and s_2 are* equivalent, $s_1 \equiv s_2$, *if they have the same position in the diagram relative to every line segment; i.e. the set of line segments with the top (or bottom) end point to the left of the top (or bottom) end point of the scanline is the same for s_1 and s_2.*

Notice that $S_1 = S_2$ does not necessarily imply that $s_1 \equiv s_2$ (the converse implication however holds).

Corollary 3.1 *There are $O(n^2)$ minimal separators in a permutation graph with n vertices.*

We are only interested in scanlines which do not cross too many line segments, since these correspond with suitable separators.

Definition 3.3 *A scanline s is* k-small *if it crosses with at most $k+1$ line segments.*

Lemma 3.2 *There are $O(nk)$ pairwise non-equivalent k-small scanlines.*

Proof. Consider the matching diagram, with numbers $1, \ldots, n$ written from left to right and underneath written $\pi_1, \ldots, \pi_n$. Consider a scanline t and assume the top end point is between i and $i+1$ and the bottom end point is between π_j and π_{j+1}. Assume that α line segments are such that the top end point is to the left of the top of t and the bottom end point to the left of the bottom of t. Then the number of line segments crossing t is $i + j - 2\alpha$. Since $\alpha \leq i$ and $\alpha \leq j$, it follows that $i - k - 1 \leq j \leq i + k + 1$ must hold. This proves the lemma. □

4 Treewidth = pathwidth

In this section we show that a permutation graph can be triangulated optimally such that the result is an interval graph.

Theorem 4.1 *Let G be a permutation graph, and let H be a minimal triangulation of G. Then H is an interval graph.*

Proof. Assume H has an astroidal triple x, y, z. Since x, y and z are pairwise nonadjacent, the corresponding line segments in the matching diagram pairwise do not cross. We may assume without loss of generality that the line segment of y is between those of x and z. Take a path p between x and z which avoids the neighborhood of y. Then each line of the path lies totally to the left or totally to the right of y. It follows that there are x' to the left of y and z' to the right of y

such that x' and z' are adjacent in H, but neither x' or z' is a neighbor of y in H. Let S be a minimal x', y-separator in H. Since H is a minimal triangulation, S is also a minimal x', y-separator in G. By lemma 3.1 S consists of all lines crossing some scanline s between x' and y. Clearly, the connected component of $G[V - S]$ containing x' lies totally to the left of s and the connected component containing z' in $G[V - S]$ lies totally to the right of s (notice $z' \notin S$ since z' lies totally to the right of y). It follows that x' and z' must be in different components of $G[V - S]$. Since H is minimal, by theorem 2.1 they must also be in different components of $H[V - S]$. But then x' and z' can not be adjacent in H. It follows that there can not be an astroidal triple, and by the characterization of Lekkerkerker and Boland ([24], stated in lemma 2.2), H is an interval graph. □

Corollary 4.1 *For any permutation graph G, the pathwidth of G is equal to the treewidth of G.*

5 Candidate components

Consider the matching diagram of G.

Definition 5.1 *Let s_1 and s_2 be two scanlines of which the intersection is either empty or one of the end points of s_1 and s_2. A* candidate component $C = C(s_1, s_2)$ *is a subgraph of G induced by the following sets of lines:*

- *All lines that are* between *the scanlines (in case the scanlines have a common end point, this set is empty).*
- *All lines crossing at least one of the scanlines.*

We identify the candidate component $C = C(s_1, s_2)$ with the diagram containing s_1, s_2 and the set of lines corresponding with vertices of C.

Definition 5.2 *Let k be an integer. A candidate component $C = C(s_1, s_2)$ is* k-feasible *if there is a triangulation H of C such that $\omega(H) \leq k+1$ and such that for each scanline s_i $(i = 1, 2)$ the set of lines crossing this scanline forms a clique in H.*

Notice that, if a candidate component has at most $k+1$ vertices then it is k-feasible.

Definition 5.3 *Let $C = C(s_1, s_2)$ be a candidate component. We define the* realizer *$R(C)$ as the graph obtained from C, by adding all edges between vertices of S_1 and between vertices of S_2 (i.e. the two subgraphs of $R(C)$ induced by S_1 and by S_2 are cliques).*

Remark. A candidate component $C = C(s_1, s_2)$ is k-feasible if and only if the realizer $R(C)$ has treewidth at most k.

Lemma 5.1 *If $C = C(s_1, s_2)$ is a candidate component, then the realizer $R(C)$ is a permutation graph.*

Proof. Consider the matching diagram. Assume s_1 is to the left of s_2. First consider lines that cross only s_1 and with top end point to the right of the top end point of s_1. Let $(a_1, b_1), \ldots, (a_r, b_r)$ be these line segments with top end points $a_1, \ldots, a_r$. Assume $a_1 < a_2 < \ldots < a_r$. Change the order of $b_1, \ldots, b_r$ such that $b_1 > b_2 > \ldots > b_r$. This is illustrated in figure 2. Now consider the line segments crossing s_1 of which the top end point is to the left of the top end point of s_1. Reorder in the same way the *top end points* of these line segments. The lines crossing s_2 are handled similarly. The resulting diagram is a matching diagram for $R(C)$. □

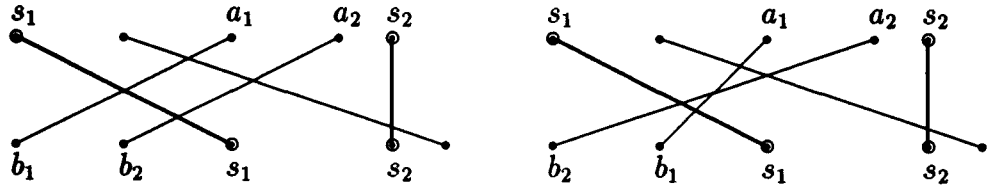

Figure 2: diagrams of candidate component and realizer

Let $C = C(s_1, s_2)$ be a candidate component such that C has at least $k+2$ vertices. The realizer $R(C)$ has treewidth at most k if and only if there is a minimal vertex separator S with at most k vertices such that all components, with S added as a clique, have treewidth at most k (see lemma 2.4). Consider the diagram of $R(C)$, obtained from the diagram of C by the method described in the proof of lemma 5.1. By lemma 3.1 a minimal separator can be found by a scanline. Let H be a minimal triangulation of $R(C)$ and let the scanline s represent a minimal vertex separator in H for non-adjacent vertices a and b. The separator consists exactly of the lines crossing this scanline s.

Definition 5.4 *Let $C = C(s_1, s_2)$ be a candidate component with realizer $R(C)$. A scanline t is* nice *if the top point of t is in the closed interval between the top points of s_1 and s_2, and the bottom point of t is in the closed interval between the bottom points of s_1 and s_2.*

Lemma 5.2 *There is a scanline $s^* \equiv s$ such that s^* is nice.*

Proof. Consider the diagram of $R(C)$ with the scanlines s_1, s_2 and s. Without loss of generality, we assume s_1 is to the left of s_2. s separates non adjacent vertices a and b. Let the line segment of a be to the left of the line segment of b. The scanline s lies between the line segments of a and b. Assume s is not nice. Without loss

of generality assume it crosses s_1. Then $a \in S_1$ and $b \notin S_1$ (since a and b are not adjacent). (see the left diagram in figure 3). Let s^* be the line segment with top point the top of s_1 and bottom point the bottom of s. We want to proof that $s \equiv s^*$. This clearly is the case if there is no line segment of which the top point is between the top points of s and s_1. Assume there is such a vertex p (see the right diagram in figure 3). Notice that, since p and a both cross s_1 they are adjacent. We claim

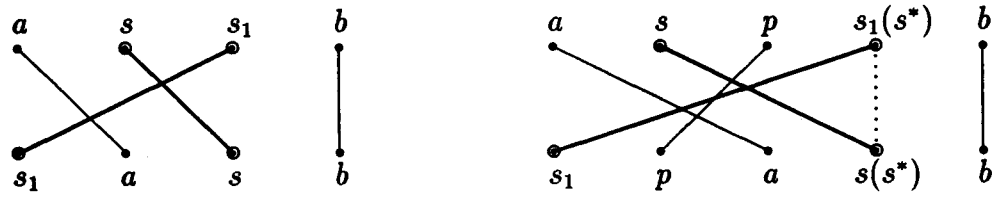

Figure 3: there is an equivalent nice scanline

that $S^* \subset S$. Let x be a line segment crossing s^*. If the bottom end of x is to the left of the bottom end of s^*, then the segment x clearly also crosses s. Assume the bottom vertex of x is to the right of the bottom vertex of s^*. Then the line segment also crosses s_1. But then x and a are adjacent, hence the top vertex of x must be to the left of the top vertex of a. This implies that x also crosses s. Clearly, S^* is an a, b-separator in $R(C)$. Since $p \in S \setminus S^*$, S can not be a minimal a, b-separator in $R(C)$, and since H is a minimal triangulation of $R(C)$, it can not be a minimal a, b-separator in H (theorem 2.1). This is a contradiction. □

This proves that all lines crossing the scanline s are in C, and the next lemma follows.

Lemma 5.3 *Let $C = C(s_1, s_2)$ be a candidate component with at least $k+2$ vertices. Then C is k-feasible if and only if there is a nice scanline s such that the two candidate components $C_1 = C(s_1, s)$ and $C_2 = C(s, s_2)$ are both smaller than C and are both k-feasible.*

We are now ready to proof our main theorem.

Theorem 5.1 *Let $C = C(s_1, s_2)$ be a candidate components with s_2 to the right of s_1. Then C is k-feasible if and only if there exists a sequence of scanlines $s_1 = t_1, t_2, \ldots, t_r = s_2$ such that the following conditions are satisfied:*

- *t_i and t_{i+1} have an endpoint in common for $i = 1, \ldots, r-1$, and the other end point of t_{i+1} lies to the right of the other end point of t_i.*
- *Each $C(t_i, t_{i+1})$ has at most $k+1$ vertices ($i = 1, \ldots, r-1$).*

Proof. First assume such a sequence exists. Let $t_1, \ldots t_r$ be the sequence of scanlines. If C has at most $k+1$ vertices, then C is k-feasible. Hence we may assume that C has at least $k+2$ vertices. Then $r \geq 3$. By induction we show that $\mathcal{C}(t_1, t_i)$ is k-feasible for $i = 2, \ldots r$. If $i = 2$ then $\mathcal{C}(t_1, t_2)$ has at most $k+1$ vertices and hence it is k-feasible. Assume $\mathcal{C}(t_1, t_{i-1})$ is k-feasible and $\mathcal{C}(t_1, t_i) \neq \mathcal{C}(t_1, t_{i-1})$. Then t_{i-1} is a nice scanline in $\mathcal{C}(t_1, t_i)$. $\mathcal{C}(t_1, t_{i-1})$ and $\mathcal{C}(t_{i-1}, t_i)$ are both k-feasible, hence, by lemma 5.3, also $\mathcal{C}(t_1, t_i)$ is k-feasible.

Now assume that C is k-feasible. Consider the case that C has at most $k+1$ vertices. If s_1 and s_2 have an end point in common we can take $r = 2$. If s_1 and s_2 do not have an end point in common, take a scanline t which has one end point in common with s_1 and the other end point in common with s_2. The sequence of scanlines s_1, t, s_2 satisfies the requirements. Assume C has at least $k+2$ vertices. Then by lemma 5.3 the is a nice scanline s^* such that $\mathcal{C}(s_1, s^*)$ and $\mathcal{C}(s^*, s_2)$ are both k-feasible. The theorem follows by induction on the number of vertices of C. □

6 Algorithm

In this section we show how to compute the treewidth (and pathwidth) of a permutation graph G. Let k be an integer. The algorithm we present checks whether the treewidth of the permutation graph does not exceed k.

We use a directed acyclic graph $W_k(G)$ as follows. The vertices of the graph are the pairwise non-equivalent k-small scanlines. Direct an arc from scanline s to t if the following two conditions hold:

- The intersection of s and t is one end point of s and t, and the other end point of t is to the right of the other end point of s, and
- the candidate component $\mathcal{C}(s, t)$ has at most $k+1$ vertices.

We call this graph the *scanline graph*.

Lemma 6.1 *Let s_L be the scanline which lies totally to the left of all line segments and let s_R be the scanline which lies totally to the right of all line segments. G has treewidth at most k if and only if there is a directed path in the scanline graph from s_L to s_R.*

Proof. Clearly s_L and s_R are k-small. The result follows immediately from theorem 5.1. □

Lemma 6.2 *The scanline graph has $O(nk)$ vertices and each vertex is incident with $O(k)$ arcs.*

Proof. The bound on the number of vertices is proved in lemma 3.2. The bound on the number of arcs incident with a vertex can be seen as follows. Let s be a k-small scanline. First we count the number of scanlines t which have an incoming arc from s in $W_k(G)$ and such that s and t have the same top end point. Consider the matching diagram and assume the top point of s and t is between i and $i+1$, the bottom vertex of s is between π_j and π_{j+1}, and the bottom point of t is between π_m and π_{m+1} (hence $m > j$). Clearly, the number of vertices in $\mathcal{C}(s,t)$ is at least $m-j$. Since there is an arc from s to t, $m-j \leq k+1$. The other cases are similar. This proves that s is incident with $O(k)$ arcs. □

We now describe the algorithm which determines if the treewidth of G is at most k.

Step 1 Make a maximal list $\mathcal{L}$ of pairwise non-equivalent k-small scanlines.

Step 2 Construct the acyclic digraph $W_k(G)$.

Step 3 If there exists a path in $W_k(G)$ from s_L to s_R, then the pathwidth of G is at most k. If such a path does not exist, then the pathwidth of G is larger than k.

We now discuss the running time of the algorithm in more detail.

Lemma 6.3 *The algorithm can be implemented to run in $O(nk^2)$ time.*

Proof. By lemma 3.2 each k-small scanline can be characterized by two indices i and θ with $0 \leq i \leq n$ and $-(k+1) \leq \theta \leq k+1$. The scanline with these indices has top end point between i and $i+1$ and bottom end point between $\pi_{i+\theta}$ and $\pi_{i+\theta+1}$ (with obvious boundary restrictions). Let $A(i,\theta)$ be the number of line segments of which the top end point is to the left of the top endpoint of this scanline and which cross the scanline. Notice that $A(0,\theta) = 0$ for $\theta = 0,\ldots,k+1$. The rest of the table follows from:

$$A(i,\theta) = \begin{cases} A(i,\theta-1) & \text{if } \pi_{i+\theta} > i \\ A(i,\theta-1)-1 & \text{if } \pi_{i+\theta} \leq i \end{cases} \quad \text{for } \theta > -\min(k+1,i), \text{ and}$$

$$A(i,\theta) = \begin{cases} A(i-1,\theta+1)+1 & \text{if } \pi_i^{-1} > i+\theta \\ A(i-1,\theta+1) & \text{if } \pi_i^{-1} \leq i+\theta \end{cases} \quad \text{for } i \geq 1 \wedge \theta = -\min(k+1,i).$$

The number of line segments crossing the scanline with indices i and θ is $2A(i,\theta)+\theta$. It follows that the list $\mathcal{L}$ of k-small scanlines can be made in $O(nk)$ time.

We now show that the scanline graph $W_k(G)$ can be constructed in $O(nk^2)$ time. Consider two k-small scanlines s and t, which have a top end point in common, say between i and $i+1$. Assume the bottom end point of s is between π_j and π_{j+1} and the bottom end point of t is between π_m and π_{m+1}. According to lemma 6.2 we must have $|m-j| \leq k+1$, otherwise there can not be an arc between s and t. Assume without loss of generality that $j < m$. Notice that the number of vertices

of $C(s,t)$ is $m - i + A(i, m-i) + A(i, j-i)$. This shows that the adjacency list for each k-small scanline can be computed in $O(k)$ time. Computing a path in W from s_L to s_R, if it exists, clearly also takes $O(nk^2)$ time. Hence the total algorithm can be implemented to run in $O(nk^2)$ time. □

Theorem 6.1 *Let G be a permutation graph. Then the pathwidth and treewidth of G are the same, and there is an $O(nk^2)$ algorithm that correctly determines whether the treewidth of G is at most k.*

We end this section by remarking that the algorithm can be adapted such that it computes, within the same time bound, the pathwidth of a permutation graph (when the matching diagram is given). This can be seen as follows. Let the treewidth of G be k. First compute a number L such that $L < k \leq 2L$. This can be done, using the algorithm described above $O(\log k)$ times, in time $O(nk^2)$ (execute the algorithm described above for $L = 1, 2, 4, \ldots$). In fact, using the algorithm described in [20], this step can be performed in $O(nk)$ time. Now construct the scanline graph $W_{2L}(G)$, and put weights on the arcs, saying how many vertices are in the corresponding candidate component. Then search for a path from s_L to s_R, such that the maximum over weights of arcs in the path is minimized. This maximum weight minus one gives the exact treewidth k of G.

7 Conclusions

In this paper we described a very simple and efficient algorithm to compute the treewidth and pathwidth of a permutation graph. In fact we have shown that, for any permutation graph, the pathwidth and treewidth are equal. It is possible to generalize these results to complements of comparability graphs of which the partial order dimension is bounded by some constant [18] (then the algorithm is exponential in this dimension). Also, these results show that, for *any* cocomparability graph, the pathwidth and treewidth are equal. There are some other classes of graphs for which the exact pathwidth and treewidth can be computed efficiently. For example cographs [8], splitgraphs and interval graphs. The treewidth can also be computed efficiently for chordal graphs, circular arc graphs [28] and chordal bipartite graphs [21]. In this respect we like to mention the result of [19] which shows that the pathwidth of chordal graphs is NP-complete. Permutation graphs are exactly the graphs that are comparability and cocomparability. It is easy to see that treewidth is NP-complete for bipartite graphs. Hence our result can not be generalized to the class of comparability graphs. However, an open problem is whether the results of this paper can be generalized to the class of cocomparability graphs (when the partial order dimension of the complement is not bounded). One of the other open problems we like to mention is whether the running time of our algorithm can be improved. Notice that there is a *linear* time (i.e. $O(nk)$) approximation algorithm for treewidth and pathwidth of permutation graphs with performance ratio 2 ([20]).

8 Acknowledgements

We like to thank A. Brandstädt, H. Müller, A. Jacobs, D. Seese and B. Reed for valuable discussions.

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

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

[5] C. Berge and C. Chvatal, *Topics on Perfect Graphs*, Annals of Discrete Math. **21**, 1984.

[6] H.L. Bodlaender, Achromatic number is NP-complete for cographs and interval graphs, *Information Processing Letter* **31**, 135 – 138, 1989.

[7] H.L. Bodlaender, Dynamic programming algorithms on graphs with bounded treewidth, *Proceedings of the* 15^{th} *International colloquium on Automata, Languages and Programming*, 105 – 119, Springer Verlag, Lecture Notes in Computer Science, vol. 317, 1988.

[8] H. Bodlaender and R.H. Möhring, The pathwidth and treewidth of cographs, In *Proceedings 2nd Scandinavian Workshop on Algorithm Theory*, 301 – 309, Springer Verlag, Lecture Notes in Computer Science vol. 447, 1990. To appear in: *SIAM J. Discr. Math.*

[9] H. Bodlaender and T. Kloks, Better algorithms for the pathwidth and treewidth of graphs, *Proceedings of the* 18^{th} *International colloquium on Automata, Languages and Programming*, 544 – 555, Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

[10] H.L. Bodlaender, A tourist guide through treewidth, Technical report RUU-CS-92-12, Department of computer science, Utrecht University, Utrecht, The Netherlands, 1992. To appear in: *Proceedings* 7^{th} *International Meeting of Young Computer Scientists*, Springer Verlag, Lecture Notes in Computer Science.

[11] A. Brandstädt and D. Kratsch, On the restriction of some NP-complete graph problems to permutation graphs, *Fundamentals of Computation Theory, proc. FCT* 1985, 53 – 62, Lecture Notes in Comp. Science vol. 199, Springer Verlag, New York, 1985.

[12] A. Brandstädt and D. Kratsch, On domination problems for permutation and other perfect graphs, *Theor. Comput. Sci.* **54**, 181 – 198, 1987.

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

[14] S. Even, A. Pnueli and A. Lempel, Permutation graphs and transitive graphs, *J. Assoc. Comput. Mach.* **19**, 400 – 410, 1972.

[15] M. Farber and M. Keil, Domination in permutation graphs, *J. Algorithms* **6**, 309 – 321, 1985.

[16] J. Gimbel, D. Kratsch and L. Stewart, On cocolourings and cochromatic numbers of graphs, to appear in: *Disc. Appl. Math.*.

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

[18] M.C. Golumbic, D. Rotem and J. Urrutia, Comparability graphs and intersection graphs, *Disc. Math.* **43**, 37 – 46, 1983.

[19] J. Gustedt, Pathwidth for chordal graphs is NP-complete. To appear in: *Discr. Appl. Math.*

[20] T. Kloks and H. Bodlaender, Approximating treewidth and pathwidth of some classes of perfect graphs. To appear in *3th Annual International Symposium on Algorithms and Computation (ISAAC '92)*.

[21] T. Kloks and D. Kratsch, Treewidth of chordal bipartite graphs, Technical report RUU-CS-92-28, Department of computer science, Utrecht University, Utrecht, The Netherlands, 1992.

[22] J. Lagergren and S. Arnborg, Finding minimal forbidden minors using a finite congruence, *Proceedings of the* 18^{th} *International colloquium on Automata, Languages and Programming*, 532–543, Springer Verlag, Lecture Notes in Computer Science, vol. 510, 1991.

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

[24] C.G. Lekkerkerker and J.Ch. Boland, Representation of a finite graph by a set of intervals on the real line, *Fund. Math.* **51**, 45 – 64, 1962.

[25] B. Reed, Finding approximate separators and computing treewidth quickly, 24^{th} *Annual ACM Symposium on Theory of Computing*, 221 – 228, 1992.

[26] Arunabha Sen, Haiyong Deng and Sumanta Guha, On a graph partition problem with application to VLSI layout, *Information Processing Letters* **43**, 87–94, 1992.

[27] J. Spinrad, On comparability and permutation graphs, *SIAM J. Comp.* **14**, No. 3, 658 – 670 August 1985.

[28] R. Sundaram, K. Sher Singh and C. Pandu Rangan, Treewidth of circular arc graphs, Manuscript 1991.

[29] K. Wagner, Monotonic coverings of finite sets, *Journal of Information Processing and Cybernetics*, EIK, **20**, 633 – 639, 1984.