THE PATHWIDTH AND TREEWIDTH OF COGRAPHS

Hans L. Bodlaender and Rolf H. Möhring

RUU-CS-90-7
February 1990

Utrecht University

Department of Computer Science

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

THE PATHWIDTH AND TREEWIDTH OF COGRAPHS

Hans L. Bodlaender and Rolf H. Möhring

Technical Report RUU-CS-90-7
February 1990

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

The Pathwidth and Treewidth of Cographs

Hans L. Bodlaender*
Department of Computer Science
Utrecht University
P.O. Box 80089 3508 TB Utrecht
The Netherlands

Rolf H. Möhring†
Department of Mathematics
Technical University of Berlin
Straße des 17. Juni 136
1000 Berlin 12
West Germany

Abstract

We show that the pathwidth of a cograph equals its treewidth, and give a linear time algorithm to determine the pathwidth of a cograph and build a corresponding path decomposition.

*The research of this author was partially supported by the ESPRIT II Basic Research Actions Program of the EC under Contract No. 3075 (project ALCOM).

†The research of this author was partially supported by the Deutsche Forschungsgemeinschaft in the Grant No. Mö 446/2-3.

ISSN:0924-3275

1 Introduction

The pathwidth and treewidth of a graph are two notions with a large number of different applications in many areas, like algorithmic graph theory, VLSI-design and others (see e.g. [1, 13]). Unfortunately, determining the pathwidth or treewidth of a given graph is NP-complete [2]. In this paper we show that there are efficient algorithms for determining the pathwidth or treewidth of a cograph. We also derive some technical lemmas, which are not only necessary to prove correctness of the algorithms, but are also interesting in their own right. For instance, we show that the pathwidth of a cograph equals its treewidth.

The complexity of the problems to determine the pathwidth and treewidth of graphs has also been studied for other interesting classes of graphs. Gustedt [10] showed that the pathwidth problem stays NP-complete when restricted to chordal graphs. For fixed k, the problem of determining whether the pathwidth or treewidth of a given graph is at most k can be solved in polynomial time with dynamic programming [2, 8], and in $O(n^2)$ time with graph minor theory [5, 16].

The notions of pathwidth and treewidth have several equivalent characterizations (see e.g. [1, 13, 18].) For instance, a graph is a partial k-tree, if and only if its treewidth is at most k.

This paper is further organized as follows. In section 2 we give most necessary definitions and some preliminary results. In section 3 we prove a number of interesting graph-theoretic lemmas and theorems. In section 4 we show how these can be used to obtain linear time algorithms for pathwidth and related notions on cographs. Some final remarks are made in section 5.

2 Definitions and preliminary results

In this section we give most necessary definitions and some preliminary results. We start with introducing the notion of *cographs.*

Notation

Let $G = (V, E), H = (W, F)$ be undirected graphs.

(i) We denote the disjoint union of G and H by $G \dot{\cup} H = (V \dot{\cup} W, E \dot{\cup} F)$ (where $\dot{\cup}$ is the disjoint union on graphs, and sets, respectively.)

(ii) With $G \times H$ we denote the following type of "product" of G and H: $G \times H = (V \dot{\cup} W, E \dot{\cup} F \cup \{(v, w) | v \in V, w \in W\})$

(iii) The complement of G is denoted by $G^c = \{V, E^c\}$, with $E^c = \{(v, w) | v, w \in V, v \neq w, (v, w) \notin E\}$.

Proposition 2.1 $\dot{\cup}, \times$ *are commutative and transitive.* $G \times H = (G^c \dot{\cup} H^c)^c$.

Definition 2.1 *A graph* $G = (V, E)$ *is a cograph, iff one of the following conditions holds:*

1. $|V| = 1$

2. *There are cographs* $G_1, \ldots, G_k$ *and* $G = G_1 \dot{\cup} G_2 \dot{\cup} \ldots \dot{\cup} G_k$

3. *There are cographs* $G_1, \ldots, G_k$ *and* $G = G_1 \times G_2 \times \ldots \times G_k$

There are other, equivalent characterizations of the class of cographs. Rule 3 can be replaced by:

3'. *There is a cograph* H *and* $G = H^c$.

Also, one can restrict k in rule 2 and 3 to be 2. Alternatively, one can define the class of cographs as the graphs that do not contain P_4, a path with 4 vertices, as an induced subgraph. (See e.g. [6]).

With each cograph $G = (V, E)$, one can associate a labeled tree, called the *cotree* T_G of G. Each vertex of G corresponds to a unique leaf in T_G. Internal vertices of T_G have a label $\in \{0, 1\}$. To each vertex in T_G one can associate a cotree in the following manner: a leaf corresponds to a cotree, consisting of one vertex. The cograph corresponding to a 0-labeled vertex v in T_G is the disjoint union of the cographs, corresponding to the sons of v in T_G. The cograph corresponding to a 1-labeled vertex v in T_G is the product ("$\times$") of the cographs, corresponding to the sons of v in T_G. Note that $(v, w) \in E$, if and only if the lowest common ancestor of v and w in T_G is labeled with a 1. (There are other very similar notions of "cotree").

Corneil, Perl and Stewart [7] gave an $O(n + e)$ algorithm for determining whether a given graph $G = (V, E)$ is a cograph, and if so, building the corresponding cotree.

A cotree T_G can easily be transformed to an equivalent cotree T'_G such that every internal vertex in T'_G has exactly 2 sons. (Note that $G_1 \dot{\cup} \ldots \dot{\cup} G_k = (G_1 \dot{\cup} \ldots \dot{\cup} G_{k-1}) \dot{\cup} G_k$, and $G_1 \times \ldots \times G_k = (G_1 \times \ldots G_{k-1}) \times G_k$. The resulting operation on trees is illustrated in Figure 1).

So, in the remainder of this paper we assume that cographs G are given together with a binary cotree T_G.

Next we give the definitions of pathwidth and treewidth, introduced by Robertson and Seymour [15, 16].

Definition 2.2 *A tree-decomposition of a graph* $G = (V, E)$ *is a pair* $(\{X_i | i \in I\}, T = (I, F))$ *with* $\{X_i | i \in I\}$ *a family of subsets of* V*, and* T *a tree, such that*

- $\cup_{i \in I} X_i = V$
- $\forall\, (v, w) \in E : \exists\, i \in I : v \in X_i \wedge w \in X_i$
- $\forall\, v \in V : \{i \in I | v \in X_i\}$ *forms a subtree of* T

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

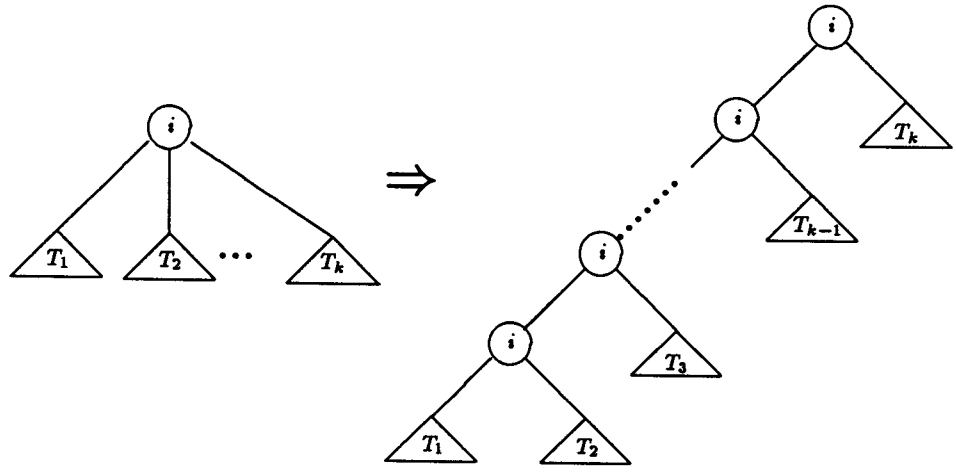

Figure 1: Transformation to a binary co-tree $i \in \{0,1\}$

Note that the third condition can be replaced by:

$$\forall\, i,j,k \in I: \text{ if } j \text{ is on the path from } i \text{ to } k \text{ in } T, \text{ then } X_i \cap X_k \subseteq X_j$$

There are several other notions that are equivalent to the notion of treewidth, e.g. a graph G is a partial k-tree, if and only if *treewidth*$(G) \leq k$. (See [1, 18]).

The notion of pathwidth is obtained from the notion of treewidth by requiring that the tree T in the tree-decompositions is a path.

Definition 2.3 *A path-decomposition of a graph $G = (V,E)$ is a pair $(\{X_i | i \in I\}, I)$, with $\{X_i | i \in I\}$ a family of subsets, and $\exists\, r \in I\!N : I = \{1, 2 \ldots, r\}$, such that*

- $\cup_{i \in I} X_i = V$
- $\forall (v,w) \in E : \exists i \in I : v \in X_i \wedge w \in X_i$
- $\forall v \in V : \exists b_v, e_v \in I : \forall i \in I : v \in X_i \Leftrightarrow b_v \leq i \leq e_v$

The pathwidth of $(\{X_i | i \in I\}, I)$ is $\max_{i \in I} |X_i|$. The pathwidth of G is the minimum pathwidth over all possible path-decompositions of G.

The third condition states that for all $v \in V$ $\{i \in I | v \in X_i\}$ forms an interval in I, and is equivalent to " $\forall\, i,j,k : i < j < k \Rightarrow X_i \cap X_k \subseteq X_j$".

The notion of pathwidth is closely related to several other notions, including node search number and interval thickness.

Definition 2.4 *The node search number of a graph $G = (V,E)$ is the minimum number of searchers needed to clear all edges of G, under the following rules.*

- *Initially all edges are contaminated.*
- *A move can consist of*
  1. *Putting a searcher on a vertex,*
  2. *Removing a searcher from a vertex,*
  3. *Moving a searcher over an edge from a vertex to an adjacent vertex.*
- *A contaminated edge becomes cleared when there is a searcher on both ends of the edge.*
- *A cleared edge becomes recontaminated when there is a path from the edge to a contaminated edge that does not pass through a vertex with a searcher on it.*

Definition 2.5 *A graph $G = (V, E)$ is an interval graph if to each $v \in V$ an interval $[b_v, e_v] \subseteq \mathbb{R}$ can be associated such that $\forall\, v, w \in V : (v, w) \in E \Leftrightarrow [b_v, e_v] \cap [b_w, e_w] \neq \phi$.*

Lemma 2.1 *(See [4, 9])*
Let $G = (V, E)$ be an interval graph. Let the chromatic number of G be $\chi(G)$, let the maximum size of a clique in G be $\omega(G)$. Then $\chi(G) = \omega(G) = \text{treewidth}(G) + 1 = \text{pathwidth}(G) + 1$.

Definition 2.6 *The interval thickness of a graph $G = (V, E)$ is the minimum chromatic number of an interval graph H that contains G as a subgraph.*

Theorem 2.1 *[11, 13]: For every graph $G = (V, E)$, the following three numbers are equal:*

- *the pathwidth of $G + 1$*
- *the node search number of G*
- *the interval thickness of G.*

3 Graph-theoretic results

In this section we derive some new and interesting graph theoretic results which are needed to derive the algorithm, but have also interest on their own. We start with a very short proof of a known result.

Definition 3.1 *A family $\{T_i | i \in I\}$ of subsets of a set T is said to satisfy the Helly-property, if for all $J \subseteq I$ with for all $i, j \in J : T_i \cap T_j \neq \emptyset$ it holds that $\cap_{j \in J} T_j \neq \emptyset$.*

Theorem 3.1 *(See [9], p.92): A family of induced subtrees of a tree satisfies the Helly property.*

Lemma 3.1 *("Clique containment lemma")*
Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree-decomposition of $G = (V, E)$, and let $W \subseteq V$ be a clique in G. Then $\exists\, i \in I : W \subseteq X_i$.

Proof
Let $T_v = \{i \in I | v \in X_i\}$. $\{T_v | v \in W\}$ is a family of subtrees of T. By theorem 3.1: $\exists\, i \in I : \forall u \in W : i \in T_v$. Hence $\exists \in I : W \subseteq X_i$. □

Older and longer proofs of lemma 3.1 can be found in [4, 18]. With the help of the Helly-property for trees we can also prove a variant of the clique-containment lemma for bipartite subgraphs.

Lemma 3.2 *"Complete bipartite subgraph containment lemma"*
Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree-decomposition of $G = (V, E)$. Let $A, B \subseteq V$, and suppose $\{(v, w) | v \in A, w \in B\} \subseteq E, A \cap B = \emptyset$. Then $\exists i \in I : A \subseteq X_i$ or $B \subseteq X_i$.

Proof
Let $(\{X_i | i \in I\}, T = (I, F)), G = (V, E)$, and A and B be given. Suppose that for all $i \in I : B \not\subseteq X_i$. Let $T_v = \{i \in I | v \in X_i\}$. Consider the family $\{T_v | v \in B\}$ of subtrees of T. As $\cap_{v \in B} T_v = \emptyset$, it follows from theorem 3.1 that there are $b_1, b_2 \in B$ such that T_{b_1} and T_{b_2} are vertex disjoint. Consider the unique path of T connecting T_{b_1} and T_{b_2}, and let k and ℓ be the border vertices of this path. (See Figure 2).

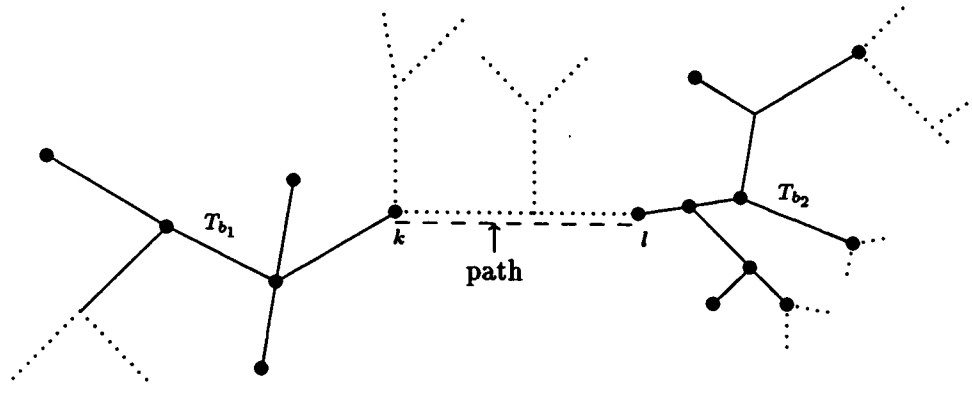

Figure 2. An illustration of the proof of Lemma 3.2

Each $a \in A$ must be contained in a set X_i with $i \in T_{b_1}$ and in a set X_j with $j \in T_{b_2}$. Hence $a \in X_k$. (Use definition of tree-decomposition). So $A \subseteq X_k$. □

Lemma 3.3 *Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition of $G = (V, E)$, and let $A, B \subseteq V$ and suppose $\{(v, w) | v \in A, w \in B\} \subseteq E, A \cap B = \emptyset$. Suppose $\exists i \in I : A \subseteq X_i$. Then there exists an induced subtree $T' = (I', F')$ of T, such that*

(i) $\forall i \in I' : A \subseteq X_i$

(ii) $B \subseteq \cup_{i \in I'} X_i$

(iii) $(\{X'_i | i \in I'\}, T' = (I', F'))$ *with* $X'_i = X_i \cap (A \cup B)$ *is a tree-decomposition of the subgraph of* G *induced by* $A \cup B$.

Proof

Let $I' = \{i \in I | A \subseteq X_i\}$. Take $T' = (I', F')$ to be the subgraph of T induced by I'. By definition of tree-decomposition, T' is again a tree. Clearly, condition (i) is fulfilled.

Because $\exists i : A \subseteq X_i, (\{X_i | i \in I\}, T)$ is a tree-decomposition of $G' = (V, E')$ with $E' = E \cup \{(v, w) | v, w \in A, v \neq w\}$. For all $b \in B, A \cup \{b\}$ forms a clique in G', and hence by the clique-containment lemma: $\exists i \in I : A \cup \{b\} \subseteq X_i \Rightarrow \exists i \in I' : b \in X_i$. So condition (ii) is fulfilled. Consider an edge $(b, c) \in E, b, c \in B$. $A \cup \{b, c\}$ forms a clique in G', and hence by the clique-containment lemma $\exists i \in I : A \cup \{b, c\} \subseteq X_i \Rightarrow \exists i \in I' : \{b, c\} \subseteq X_i$. It now easily follows that condition (iii) is fulfilled. □

Lemma 3.4 *Let* $G = (V, E), H = (W, F)$ *be graphs.*

(i) $\mathit{treewidth}(G \dot{\cup} H) = \max(\mathit{treewidth}(G), \mathit{treewidth}(H))$.

(ii) $\mathit{pathwidth}(G \dot{\cup} H) = \max(\mathit{pathwidth}(G), \mathit{pathwidth}(H))$.

(iii) $\mathit{treewidth}(G \times H) = \min(\mathit{treewidth}(G) + |W|, \mathit{treewidth}(H) + |V|)$.

(iv) $\mathit{pathwidth}(G \times H) = \min(\mathit{pathwidth}(G) + |W|, \mathit{pathwidth}(H) + |V|)$.

Proof

(i), (ii) Trivial.

(iii) First we show that $\mathit{treewidth}(G \times H) \leq \mathit{treewidth}(G) + |W|$. Take a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of G with treewidth $\mathit{treewidth}(G)$. Then $(\{X_i \cup W | i \in I\}, T = (I, F))$ is a tree-decomposition of $G \times H$ with treewidth $\mathit{treewidth}(G) + |W|$. So $\mathit{treewidth}(G \times H) \leq \mathit{treewidth}(G) + |W|$. Similarly one can show $\mathit{treewidth}(G \times H) \leq \mathit{treewidth}(H) + |V|$.

Next we show that $\mathit{treewidth}(G \times H) \geq \min(\mathit{treewidth}(G) + |W|, \mathit{treewidth}(H) + |V|)$. Consider a tree-decomposition $(\{X_i | i \in T\}, T = (I, F))$ of $G \times H$. From the complete bipartite subgraph containment lemma it follows that $\exists\, i : V \subseteq X_i$ or $\exists\, i : W \subseteq X_i$.

Suppose $\exists i : V \subseteq X_i$. Let $T = (I', F')$ be a subtree of T such that $\forall i \in I' : V \subseteq X_i, W \subseteq \cup_{i \in I'} X_i$ and $(\{X_i | i \in I'\}, T' = (I+, F'))$ is a tree-decomposition of $G \times H$. T' exists by lemma 3.3. Note that $(\{X_i \cap W | i \in I'\}, T' = (I', F))$ is a tree-decomposition of H, so $\exists i \in I' : |X_i \cap W| \geq \mathit{treewidth}(H) + 1 \Rightarrow \exists i \in I', |X_i| \geq |V| + \mathit{treewidth}(H) + 1$. So the treewidth of the tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ is at least $|V| + \mathit{treewidth}(H)$.

Similarly, if $\exists i : W \subseteq X_i$, one can show that the treewidth of $(\{X_i | i \in I\}, T = (I, F))$ is at least $|W| + \mathit{treewidth}(G)$. Hence $\mathit{treewidth}(G \times H) \geq \min(|V| + \mathit{treewidth}(H), |W| + \mathit{treewidth}(G))$.

(iv) Similar to (iii). □

Theorem 3.2 *For every cograph $G = (V, E)$: treewidth(G) = pathwidth(G).*

Proof

Use induction on $|V|$.

If G consists of a single vertex, then *treewidth*$(G) = 0 =$ *pathwidth*(G). If $G = G_1 \dot{\cup} G_2$, then *treewidth*(G) = max(*treewidth*(G_1), *treewidth*(G_2)) = $(i.h.)$ max(*pathwidth*(G_1), *pathwidth*(G_2)) = *pathwidth*(G). If $G = G_1 \times G_2$, with $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, then *treewidth*(G) = min(*treewidth*$(G_1) + |V_2|$, *treewidth*$(G_2) + |V_1|$) = $(i.h.)$ min(*pathwidth*$(G_1) + |V_2|$, *pathwidth*$(G_2) + |V_1|$) = *pathwidth*(G). □

4 Algorithms for pathwidth and related notions on cographs

In this section we give linear algorithms for determining treewidth, pathwidth, path-decompositions, optimal node search strategies, and interval graph augmentations with minimum clique size of cographs.

In Figure 3 we give two recursive procedures. COMPUTE-SIZE computes for every vertex in a binary cotree the number of vertices of the corresponding cograph. COMPUTE-PATHWIDTH computes for every vertex in a binary cotree the pathwidth of the cograph corresponding to that vertex. To compute the pathwidth of a cograph G, let r be the root of the binary cotree corresponding to G. Now first call COMPUTE-SIZE(r), and then COMPUTE-PATHWIDTH(r). As per vertex in the cotree a constant number of operations are performed, this costs $O(n)$ time in total. Correctness follows from lemma 3.4.

Theorem 4.1 *The pathwidth and treewidth of a cograph given with a corresponding cotree, can be computed in $O(n)$ time.*

It is not hard to construct corresponding path-decompositions in time, linear in the output, i.e. linear in $\sum_{i \in I} |X_i|$. However, in some cases this may be quadratic in n. (Consider a cograph $G = G_1 \times G_2$, where G_1 is a clique with $n/2$ vertices, and G_2 consists of $n/2$ isolated vertices. The optimal tree-decomposition of G will consist of $n/2$ sets X_i, each containing each vertex of G_1 and one vertex of G_2).

Thus, we are looking for a more compact representation of path-decompositions. We solve this in the following way: for each $v \in V$ we compute numbers *first*(v)= $\min\{i \in I | v \in X_i\}$ and *last*(v) $= max\{i \in I | v \in X_i\}$. These numbers fix the path-decomposition, because for all $v \in V, i \in I : v \in X_i \Leftrightarrow$ *first*$(v) \leq i \leq$ *last*(v).

Note that this representation corresponds by assigning to each $v \in V$ an interval such that the corresponding interval graph contains G; the chromatic number = maximum clique size of this interval graph equals the pathwidth of G plus 1. Thus we also find a representation of G corresponding to its interval thickness.

The numbers *first*(v) and *last*(v) for all $v \in V$ are computed in the procedure MAKE-INTERVALS of Figure 4, which is called with MAKE-INTERVALS $(r, 1, m)$, where r is the root of the binary cotree of G, and m is an integer variable. In the

```
procedure  COMPUTE-SIZE (v: NODE);
   begin if v is a leaf of T_G
            then size (v) := 1
            else begin COMPUTE-SIZE (left son of v)
                       COMPUTE-SIZE (right son of v);
                       size (v) := size (left son of v) + size (right son of v)
                 end
   end

procedure  COMPUTE-PATHWIDTH (v: NODE);
   begin if v is a leaf of T_G
            then pathwidth(v) := 0
            else begin COMPUTE-PATHWIDTH (left son of v);
                       COMPUTE-PATHWIDTH (right son of v);
                       if label(v)=0
                         then pathwidth(v):= max(pathwidth (left son of v),
                                pathwidth (right son of v))
                         else pathwidth(v):= min(size (left son of v) +
                                pathwidth (right son of v),
                                pathwidth (left son of v) + size (right son of v))
                 end
   end
```

Figure 3

procedure, *start* always denotes the smallest value that can be used for *first*(w) with w a leaf in the subtree of the cotree rooted at v, and *finish* will yield the largest value used for *last*(w), with w again leaf in the subtree rooted at v. Correctness of the procedure easily follows. Clearly, the procedure is linear in the size of the cotree $= O(n)$.

Theorem 4.2 *A representation of a path-decomposition with optimal pathwidth of a cograph, given with a corresponding cotree, can be computed in $O(n)$ time.*

Theorem 4.3 *The pathwidth and treewidth of cographs and corresponding path-decompositions or tree-decompositions can be computed in $O(n+e)$ time.*

Proof

Recall that the cotree of a cograph can be found in $O(n+e)$ time (see section 2). We now use the fact that optimal path-decompositions of cotrees fulfill $\sum_{i \in I} |X_i| = O(n+e)$. □

Theorem 4.4 *There exists an algorithm that, given a cograph G and a corresponding cotree of G, determines in $O(n)$ time an interval graph H that contains G as a subgraph and has chromatic number equal to the interval thickness of G.*

```
procedure MAKE-INTERVALS (v: vertex, start:integer, finish: var integer);
   var help: integer:
   begin if v is a leaf of T_G
         then begin first(v) :=start;
                    last(v) :=start;
                    finish(v):=start;
              end
         else if label(v) = 0
              then  begin MAKE-INTERVALS (left son of v, start,help);
                          MAKE-INTERVALS (right son of v, help+1,finish)
                    end
              else (* label(v) = 1)
              if size (left son of v) + pathwidth(right son of v) >
                    size (right son of v) + pathwidth(left son of v)
                    then begin MAKE-INTERVALS (left son of v, start, finish);
                          for each w ∈ V that is a leaf-descendant
                              of the right son of v
                          do begin first(w) :=start;
                                   last(w) := finish;
                              end
                          end
                    else  begin MAKE-INTERVALS (right son of v, start, finish)
                          for each w ∈ V that is a leaf-descendant
                                of the left son of v
                          do begin first(w) := start;
                                   last(w) := finish;
                              end
                          end
   end
```

Figure 4

Theorem 4.5 *There exists an algorithm that, given a cograph G and a corresponding cotree of G, determines the node search number of G and corresponding search strategy in $O(n)$ time.*

Proof

Compute *first*(v) and *last*(v) for all $v \in V$ as described above. Now use the following search strategy:

put a searcher on each vertex v with *first*(v) =1
for $i := 1$ to $\max\{last(v) | v \in V\} - 1$
do begin for all $v \in V$ with *last*$(v) = i$: remove searcher from v;
for all $v \in V$ with *first*$(v) = i + 1$: put a searcher on v
end

With this search strategy, all edges will be cleared, no recontamination can take place, and the optimal number of searchers *(pathwidth(G)+1)* is used. Determining the sets $\{v | first(v) = i\}$, and $\{v | last(v) = i\}$ can be done with bucket sort in $O(n)$ time in total. □

5 Final Remarks

In this paper we gave a linear time algorithm to determine the treewidth and pathwidth of cographs. Currently, we are investigating how to extend the results of this paper to larger classes of graphs, e.g., graphs that are built with modular composition with small neighborhood modules (see [14]). Another interesting problem is whether these results can be extended to distance-hereditary graphs.

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

[4] H.L. Bodlaender. Dynamic programming algorithms on graphs with bounded tree-width. Techn. rep., Lab. for Computer Science, M.I.T., 1987. Ext. abstract in proceedings ICALP 88.

[5] H.L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. Technical Report RUU-CS-88-29, Dept. of Computer Science, Univ. of Utrecht, 1988. To appear in: Proc. Workshop on Graph Theoretic Concepts in Comp. Sc. '89.

[6] D.G. Corneil, H. Lerchs, and L. Stewart Burlingham. Complement reducible graphs, Disc. Appl. Math. 3: 163-174, 1981.

[7] D.G Corneil, Y. Perl and L. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 4:926-934, 1985.

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

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

[10] J. Gustedt. Path width for chordal graphs is NP-complete. Preprint TU Berlin, 1989.

[11] L.M. Kirousis and C.H. Papadimitriou. Interval graphs and searching. *Disc. Math.*, 55:181-184, 1985.

[12] L.M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theor. Comp. Sc.*, 47:205-218, 1986.

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

[14] M.B. Novick. Fast parallel algorithms for the modular decomposition. Technical Report TR 89-1016, Department of Computer Science, Cornell University, 1989.

[15] N. Robertson and P. Seymour. Graph minors. I. Excluding a forest. *J. Comb. Theory Series B*, 35:39-61, 1983.

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

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

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