Edge search number of cographs in linear time

Pinar Heggernes* Rodica Mihai*

Abstract

We give a linear-time algorithm for computing the edge search number of cographs, thereby proving that this problem can be solved in polynomial time on this graph class. With our result, the knowledge on graph searching of cographs is now complete: node, mixed, and edge search numbers of cographs can all be computed efficiently. Furthermore, we are one step closer to computing the edge search number of permutation graphs.

1 Introduction

Graph searching has been subject to extensive study [4, 3, 26, 28, 23, 31, 38, 16, 14, 27] and it fits into the broader class of pursuit-evasion/search/rendezvous problems on which hundreds of papers have been written (see e.g., the book [1]). The problem was introduced by Parsons [30] and by Petrov [34] independently, and the original definition corresponds exactly to what we today call edge searching. In this setting, a team of searchers is trying to catch a fugitive moving along the edges of a graph. The fugitive is very fast and knows the moves of the searchers, whereas the searchers cannot see the fugitive until they capture him (when the fugitive is trapped and has nowhere to run). An edge is cleared by sliding a searcher from one endpoint to the other endpoint, and a vertex is cleared when a searcher is placed on it; we will give the formal definition of clearing a part of the graph in the next section. The problem is to find the minimum number of searchers that can guarantee the capture of the fugitive, which is called the edge search number of the graph.

There are two modifications of the classical Parsons-Petrov model, namely node searching and mixed searching, introduced by Kirousis and Papadimitriou [24] and Bienstock and Seymour in [4], respectively. The main difference between the three variants is in the way an edge is cleared. In the node searching version an edge is cleared if both its endpoints contain searchers. The mixed searching version

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: {pinar.heggernes, rodica.mihai}@ii.uib.no.

combines the features of node and edge searching, namely an edge is cleared if either both its two endpoints contain searchers or a searcher is slided along it. The minimum number of searchers sufficient to perform searching and ensure the capture of the fugitive for each of the three variants are respectively the edge, node, and mixed search numbers, and computations of these are all NP-hard [4, 28, 23].

Polynomial-time algorithms are known for computing the node search number of trees [32, 35], interval graphs [8], k-starlike graphs for fixed k [31], d-trapezoid graphs [6], block graphs [10], split graphs [20], circular-arc graphs [36], permutation graphs [5, 29], biconvex bipartite graphs [33], and unicyclic graphs [13]. However, only for a few of these graph classes polynomial-time algorithms are known for computing mixed search and edge search numbers. Mixed search number of interval graphs, split graphs [15] and permutation graphs [22] can be computed in polynomial time. Edge search number of trees [28, 32], interval graphs, split graphs [31, 18], unicyclic graphs [38], and complete multipartite graphs [2] can be computed in polynomial time.

In this paper, we give a linear-time algorithm for computing the edge search number of cographs, and thereby we prove that the edge search number of cographs can be computed in polynomial time, which has been an open problem until now. Cographs are an important and well studied subclass of permutation graphs [19, 9]. Hence, by the mentioned results on permutation graphs above, their node search and mixed search numbers were already known to be computable in polynomial time. An especially designed algorithm for the node search number of cographs also exists [7]. Our new results complete the knowledge on the graph searching on cographs, showing that node, mixed, and edge search numbers of cographs can all be computed efficiently. Apart from cographs, we see from the above list that interval and split graphs are the only graph classes for which polynomial-time algorithms are known for computing their node, mixed and edge search numbers. For permutation graphs, we still do not know how to compute their edge search number in polynomial time. With our results, we extend the knowledge on permutation graphs in the sense that we know at least how to compute the edge search number of some permutation graphs, namely cographs.

2 Preliminaries

We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$. The set of *neighbors* of a vertex x is denoted by $N(x) = \{y \mid xy \in E\}$. The *degree* of a vertex v is $d(v) = |N(v)|$. A vertex is *universal* if $N(v) = V \setminus \{v\}$ and *isolated* if $N(v) = \emptyset$. A vertex set is a *clique* all of its vertices are pairwise are adjacent, and an *independent set* if all of its vertices are pairwise non-adjacent. The subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$. For a given vertex $u \in V$, we denote $G[V \setminus \{u\}]$ simply by $G{-}u$.

K_n denotes the complete graph on n vertices. I_n denotes the graph on n isolated vertices (hence no edge), and by $K_{n,m}$ we denote the complete bipartite graph (X, Y, E) such that $|X| = n$ and $|Y| = m$.

Let $G = (V, E)$ and $H = (W, F)$ be two undirected graphs with $V \cap W = \emptyset$. The *(disjoint) union* of G and H is $G \oplus H = (V \cup W,\ E \cup F)$, and the *join* of G and H is $G \otimes H = (V \cup W,\ E \cup F \cup \{vw \mid v \in V, w \in W\})$. *Cographs* are defined recursively through the following operations:

- A single vertex is a cograph.
- If G and H are vertex disjoint cographs then $G \oplus H$ is a cograph.
- If G and H are vertex disjoint cographs then $G \otimes H$ is a cograph.

Consequently, complements of cographs are also cographs. If G is a cograph then either G is disconnected, or its complement $\overline{G}$ is disconnected, or G consists of a single vertex. Using the corresponding decomposition rules one obtains the modular decomposition tree of a cograph which is called a cotree. A *cotree* T of a cograph G is a rooted tree with two types of interior nodes: 0-nodes and 1-nodes. The vertices of G are assigned to the leaves of T in a one-to-one manner. Two vertices u and v are adjacent in G if and only if the lowest common ancestor of the leaves u and v in T is a 1-node. A graph is a cograph if and only if it has a cotree [11]. Cographs can be recognized and their corresponding cotrees can be generated in linear time [21, 12].

A *path-decomposition* of a graph $G = (V, E)$ is a linearly ordered sequence of subsets of V, called *bags*, such that the following three conditions are satisfied: 1. Every vertex $x \in V$ appears in some bag. 2. For every edge $xy \in E$ there is a bag containing both x and y. 3. For every vertex $x \in V$, the bags containing x appear consecutively. The *width* of a decomposition is the size of the largest bag minus one, and the *pathwidth* of a graph G, $pw(G)$, is the minimum width over all possible path decompositions.

The *edge search game* can be formally defined as follows. Let $G = (V, E)$ be a graph to be searched. A *search strategy* consists of a sequence of discrete steps which involves searchers. Initially there is no searcher on the graph. Every step is one of the following three types

- Some searchers are placed on some vertices of G (there can be several searchers located in one vertex);
- Some searchers are removed from G;
- A searcher slides from a vertex u to a vertex v along edge uv.

At every step of the search strategy the edge set of G is partitioned into two sets: *cleared* and *contaminated* edges. Intuitively, the agile and omniscient fugitive with unbounded speed who is invisible for the searchers, is located somewhere on a contaminated territory, and cannot be on cleared edges. Initially all edges of G are contaminated, i.e., the fugitive can be anywhere. A contaminated edge uv becomes cleared at some step of the search strategy if at this step a searcher located in u slides to v along uv.

A cleared edge e is (re)contaminated at some step if at this step there exists a path P containing e and a contaminated edge and no internal vertex of P contains a searcher. For example, if a vertex u is incident to a contaminated edge e, there is only one searcher at u and this searcher slides from u to v along edge $uv \neq e$, then after this step the edge uv, which is cleared by sliding, is immediately recontaminated.

A search strategy is *winning* if after its termination all edges are cleared. The *edge search number* of a graph G, denoted by $es(G)$, is the minimum number of searchers required for a winning strategy of edge searching on G. The differences between mixed, edge, and node searching are in the way the edges can be cleared. In node searching an edge is cleared only if both its endpoints are occupied (no clearing by sliding). In mixed searching an edge can be cleared both by sliding and if both its endpoints are occupied by searchers. The *mixed* and *node search numbers* of a graph G are defined similarly to the edge search number, and are denoted by $ms(G)$ and $ns(G)$, respectively. The following result is central; it gives the relation between the three graph searching parameters and relates them to pathwidth.

Lemma 1 ([37]) *Let G be an arbitrary graph.*

- $ns(G) = pw(G) + 1$.
- $pw(G) \leq ms(G) \leq pw(G) + 1$.
- $pw(G) \leq es(G) \leq pw(G) + 2$.

Hence computing the pathwidth and the node search number are equivalent tasks. However, note that, although $pw(G)$ of a graph G can be computed easily, it might be difficult to decide whether $es(G) = pw(G)$ or $es(G) = pw(G) + 1$ or $es(G) = pw(G) + 2$.

A winning edge search strategy using $es(G)$ steps is called *optimal.* A search strategy is called *monotone* if at any step of this strategy no recontamination occurs. For all three versions of graph searching, recontamination does not help to search the graph with fewer searchers [4, 26], i.e., on any graph with edge search number k there exists a winning monotone edge search strategy using k searchers. Thus in this paper we consider only monotone edge search strategies.

3 Edge search number of cographs

In this section we show how to compute the edge search number of a cograph. We start by giving general results on the disjoint union and join of two arbitrary graphs. Given an arbitrary graph G and an integer c, following Golovach [17], we define G_c to a supergraph of G, obtained from G by attaching c new vertices of degree 1 to each vertex of G. Hence G_c has $c \cdot |V(G)|$ vertices in addition to the $|V(G)|$ vertices of G.

Lemma 2 ([17]) *Let G and H be two arbitrary graphs with $|V(G)| = n$ and $|V(H)| = m$, such that the pair $\{G, H\}$ is not one of the following pairs $\{I_1, I_1\}$, $\{I_1, I_2\}$, $\{I_2, I_2\}$, $\{I_2, K_k\}$. Then $es(G \otimes H) = \min\{es(G_m)+m,\ es(H_n)+n\}$.*

We will relate the above lemma to edge search strategies. To have an easy notion of the number of searchers that are *used* at each step of the search, assume for the rest of this section that every searcher which is not necessary is removed from the graph as soon as it becomes unnecessary. We define $extra(G) = 1$ if there is an optimal edge strategy on G such that every time the maximum number of searchers is used the following operation is executed: sliding a searcher through an edge whose both endpoints are occupied by two other searchers. Hence $extra(G) = 0$ if every optimal edge search strategy avoids this operation at least once when using the maximum number of searchers.

Lemma 3 *Let G be an arbitrary graph and $c > 2$ be an integer. Then $es(G_c) = es(G)+1-extra(G)$.*

Proof. Clearly, $es(G_c) \geq es(G)$. Let us study the two cases $extra(G) = 1$ and $extra(G) = 0$ separately.

Let $extra(G) = 1$. Then it follows directly that $es(G_c) \geq es(G)+1-extra(G) = es(G)$. Let us show that $es(G_c) \leq es(G)+1-extra(G) = es(G)$. We will do this by turning any optimal edge strategy for G into an edge strategy for G_c using at most the same number of searchers. We run the each search strategy of G on G_c. Since at each step of the search at least one searcher is available to be slided between two already occupied vertices, whenever the strategy of G clears a vertex v, we keep the searcher on v, and we use the extra available searcher to clear all the vertices of degree 1 adjacent to v, one by one. Thus we conclude that $es(G_c) = es(G) = es(G) + 1 - extra(G)$ when $extra(G) = 1$.

Let $extra(G) = 0$ and $es(G) = k$. First we show that $es(G_c) \geq es(G) + 1 - extra(G) = k + 1$. We know that at least k searchers are necessary to clear G_c, by the first sentence of the proof. So assume for a contradiction that $es(G_c) = k$. Consider any optimal edge search strategy for G_c; let us study the last step before the k'th searcher is used for the first time. To get rid of some simple cases, without loss of generality we can use the k'th searcher to clear all edges whose both endpoints are occupied by searchers. In addition, if a degree one vertex contains a searcher, we can slide it to the single neighbor v of this vertex, and then use the k'th searcher to clear all edges between v and its neighbors of degree 1. Hence, for each vertex u of degree at least 2 containing a searcher, we can use the k'th searcher to clear all edges between u and its neighbors of degree 1. Furthermore, if a vertex containing a searcher is incident to only one contaminated edge, then we can slide its searcher to the other endpoint of the contaminated edge, clearing the edge. After repeating this for as long as possible, if some vertices are incident to only cleared edges, we can remove their searcher and place it on an uncleared vertex. Hence we can assume that there is a step in this search strategy where $k-1$ searchers are placed on the vertices of G, all edges between vertices of degree one and their neighbors containing searchers are cleared, all edges containing searchers on both endpoints are cleared, and G_c is not yet cleared since $extra(G) = 0$ and we have so far only slided the k'th searcher between vertices of G occupied with searchers. At this point, every vertex containing a searcher is incident to at least two contaminated edges of G. After this point, we can clear at most one contaminated edge incident to some vertex occupied by a searcher, by sliding the k'th searcher from the occupied endpoint towards the endpoint w not occupied by a searcher. Note that w is not a degree one vertex, and all edges between w and its neighbors of degree one are contaminated. Consequently, from now on no searcher can be removed or slided without allowing recontamination, and the search cannot continue successfully without increasing the number of searchers. Thus $es(G_c) \geq k+1 = es(G)+1-extra(G)$. Let us now show that $es(G_c) \leq es(G)+1$, that $k+1$ searchers are enough to clear G_c. We construct an optimal edge search strategy for G_c by following the steps of an optimal edge search strategy for G. At each step where we place a searcher on a vertex v of G we use the extra searcher to clear all the edges between v and vertices of degree 1. Thus $es(G_c) = es(G) + 1 - extra(G)$ if $extra(G) = 0$. □

By Lemmas 2 and 3, the next lemma follows immediately. For the cases that are not covered by this lemma, it is easy to check that $es(I_1 \otimes I_1) = es(I_1 \otimes I_2) = es(I_2 \otimes I_2) = 1$ and $es(I_2 \otimes K_k) = k + 1$ for $k \geq 2$.

Lemma 4 *Let G and H be two arbitrary graphs with $|V(G)| = n$ and $|V(H)| = m$, such that the pair $\{G, H\}$ is not one of the following pairs $\{I_1, I_1\}$, $\{I_1, I_2\}$, $\{I_2, I_2\}$, $\{I_2, K_k\}$. Then $es(G \otimes H) = \min\{n+es(H)+1-extra(H),\ m+es(G)+1-extra(G)\}$.*

Consequently, if we know how to compute $extra(G)$ for a graph G then we can compute the edge search number of the join of two graphs using the above lemma.

This might be a difficult task for general graphs, but here we will show that we can compute $extra(G)$ efficiently if G is a cograph.

Before we continue with this, we briefly mention that the disjoint union operation on two arbitrary graphs is easy to handle with respect to edge search number and the parameter *extra*. If G and H are two arbitrary disjoint graphs, then clearly $es(G \oplus H) = \max\{es(G), es(H)\}$. Furthermore we have the following observation on $extra(G \oplus H)$.

Lemma 5 *Let G_1 and G_2 be two arbitrary graphs. Then*
$extra(G_1 \oplus G_2) = \min_{i\in\{1,2\}}\{extra(G_i) \mid es(G_i) = es(G_1 \oplus G_2)\}$.

Proof. Without loss of generality let $es(G_1 \oplus G_2) = es(G_1)$. We have two possibilities: either $es(G_2) < es(G_1)$ or $es(G_2) = es(G_1)$. For the first case, $extra(G_1 \oplus G_2) = extra(G_1)$, regardless of $extra(G_2)$, since we can search G_2 first and then move all the searchers to G_2. For the second case, the lemma claims that if $extra(G_1) = 0$ or $extra(G_2) = 0$ then extra $extra(G_1 \oplus G_2) = 0$. This is easy to see, since regardless of where we start the search, there will be a point of the search where all searchers are used without the use of the sliding operation between two vertices occupied by searchers. □

We continue by listing some simple graphs G with $extra(G) = 0$. For the graphs covered by the next lemma, it is known that $es(I_n) = 1$, $es(K_2) = 1$, $es(K_3) = 2$, and $es(K_n) = n$ for $n \geq 4$. Furthermore, $es(K_{n,m}) = \min\{n, m\}+1$ when $\min\{n, m\} \leq 2$ and since $(I_2 \otimes K_n)$ is an interval graph, $es(I_2 \otimes K_n) = n + 1$ for $n \geq 1$, by the results of [31, 18].

Observation 6 *If G is one of the following graphs then $extra(G) = 0$: I_n, K_n with $n \leq 3$, $K_{n,m}$ with $\min\{n, m\} \leq 2$, or $(I_2 \otimes K_n)$.*

Proof. The optimal edge search strategies for these graphs are known, as listed before the lemma, from previous results [2, 15]. Using these results and by the definition of the parameter *extra* it follows immediately that $extra(G) = 0$ if G is one of the following graphs: I_n, K_n, or $K_{n,m}$ with $min\{n, m\} < 3$. If $G = I_2 \otimes K_n$ then since G an interval graph, it follows from [31, 18] that $es(G) = n+1$. It follows also that $extra(G) = 0$ since in every optimal edge search strategy for G, when the maximum number of searchers are required, at least one edge uv is cleared by sliding the searcher from u towards v when all adjacent edges to u are cleared except uv. □

Observation 7 *If G has a universal vertex u, and the size of the largest connected component of $G-u$ is at most 2, then $extra(G) = 0$.*

Proof. If all connected components of $G-u$ are of size 1, then $G = K_{1,n}$ and covered by the previous observation. Otherwise, a graph G that satisfies the premises of the lemma consists of edges and triangles all sharing a common vertex u, and sharing no other vertices. Such a graph is an interval graph, and it is known that it can be cleared with 2 searchers: place one searcher on u, and clear every edge or triangle attached at u by sliding the second searcher from u to the other vertices of the edge or the triangle. Clearly $extra(G) = 0$. □

Notice that the above two observations, together with Lemma 5, handle the *extra* parameter of all (and more) graphs that are not covered by Lemma 4.

We are now ready to show how to compute $extra(G)$ when G is a cograph. This will be explained algorithmically in the proof of the next lemma. For this we will use the cotree as a data structure to store G. Note that due to the decomposition rules on cographs explained in Section 2, we may assume that each interior node of a cotree has exactly two children. As an initialization, note that a single vertex is a special case of I_n, and hence for a single vertex u we define $extra(u) = 0$. Consequently, in our algorithm every leaf l of the cotree of a cograph will have $extra(l) = 0$ before we start the computations.

Lemma 8 *Let G be a cograph. Then $extra(G)$ can be computed in linear time.*

Proof. Let G be a cograph and let T be its cotree. If G is one of the special cographs covered by Observations 6 and 7 then $extra(G) = 0$. We assume now we initialized all the subtrees corresponding to the special cases covered by these observations. Let us consider now the first node in the cotree which corresponds to a graph which is not one of those cases. If we are dealing with a 0-node then we can compute the value for the parameter *extra* by Lemma 5. We will show now how to compute *extra* for a 1-node. Let T_l and T_r be the the left subtree and the right subtree of the 1-node considered and let G_l and G_r be the corresponding cographs that have T_l and T_r as their cotrees, respectively.

We first consider the case when $extra(G_l) = extra(G_r) = 0$. Since we already initialized all the special cases covered by Observations 6 and 7, and we are at a join-node, we know that we not dealing with one of the cases not covered by Lemma 4. Thus by Lemma 4 we have that $es(G_l \otimes G_r) = \min\{|V(G_l)| + es(G_r) + 1 - extra(G_r), |V(G_r)| + es(G_l) + 1 - extra(G_l)\} = \min\{|V(G_l)| + es(G_r) + 1, |V(G_r)| + es(G_l) + 1\}$. Let us assume without loss of generality that $es(G_l \otimes G_r) = |V(G_l)| + es(G_r) + 1$. We will show now that there is an optimal edge search strategy for $G_l \otimes G_r$ using at every step that requires the maximum number of searchers in the strategy the following operation: an edge is cleared by sliding a searcher from one endpoint towards the other endpoint when both endpoints are occupied by searchers.

We place $|V(G_l)|$ searchers on the vertices of G_l, and we use one more searcher to clear all the edges inside G_l. At this point the only edges not cleared are the edges of G_r and the edges between the vertices of G_r and the vertices of G_l. The following step in the edge search strategy for $G_l \otimes G_r$ is the same as the first step in the edge search strategy for G_r. At each point when we place a new searcher on a vertex v of G_r we use one searcher to clear the edges between v and G_l. This is possible to do also when using the maximum number of searchers in G_r which is $es(G_r)$. At this point $|V(G_l)|$ searchers are placed on the vertices of G_l and we have $es(G_r)$ searchers on some vertices of G_r. Since $es(G_l \otimes G_r) = |V(G_l)| + es(G_r) + 1$ we have one more searcher available to clear the edges between G_l and G_r by sliding. This is true for each step when using the largest number of searchers in G_r. Thus, by the definition of $extra$ we have $extra(G_l \otimes G_r) = 1$.

We consider now the case when $extra(G_l) = 0$ and $extra(G_r) = 1$. First we consider the case when $es(G_l \otimes G_r) = \min\{|V(G_l)| + es(G_r), |V(G_r)| + es(G_l) + 1\} = |V(G_l)| + es(G_r)$. We give a corresponding edge search strategy such that $extra(G_l \otimes G_r) = 1$. We place as before $|V(G_l)|$ searchers on the vertices of G_l and use one more searcher to clear the edges inside G_l. Next steps are to imitate the optimal edge search strategy of G_r. We know that $extra(G_r) = 1$ which means that at every step when using $es(G_r)$ searchers on G_r, one searcher is used only to slide trough an edge uv whose both endpoints are occupied by two other searchers. Thus we can use the same sliding searcher to clear the edges between u and the vertices of G_l and the edges between v and the vertices of G_l. Thus $extra(G_l \otimes G_r) = 1$. Let assume now that $es(G_l \otimes G_r) = min\{|G_l| + es(G_r), |G_r| + es(G_l) + 1\} = |G_r| + es(G_l) + 1$. We construct the desired edge search strategy in the following manner. We place $|G_r|$ searchers on the vertices of G_r. After that we construct the edge search strategy similar to the first case consider when $extra(G_l) = extra(G_r) = 0$. Thus $extra(G_l \otimes G_r) = 1$.

The last case we need to consider is $extra(G_l) = extra(G_r) = 1$. Then $es(G_l \otimes G_r) = min\{|G_l| + es(G_r), |G_r| + es(G_l)\}$. This is similar to the case when $extra(G_l) = 0$ and $extra(G_r) = 1$ and $es(G_l \otimes G_r) = |G_l| + es(G_r)$. Thus we have $extra(G_l \otimes G_r) = 1$ also in this situation.

All the previous cases can be checked in constant-time. For each node of the cotree we compute the value of $extra$ in constant-time using a bottom-up strategy. Therefore, we can conclude that $extra(G)$ can be computed in linear-time for a cograph. □

In fact, a stronger result follows immediately by the proof of Lemma 8:

Corollary 9 *If G is a connected cograph, and G is not one of the graphs covered by Observations 6 and 7, then $extra(G) = 1$.*

Theorem 10 *Let G be a cograph. Then the edge search number of G can be computed in linear time.*

Proof. In order to compute the edge search number of a cograph G we do the following. First we compute the cotree T of G in linear time. The next step is to initialize all starting subtrees according to Observations 6 and 7. After that we use a bottom-up strategy to compute the edge search number of G. For each 1-node we compute the edge search number according to Lemma 4 and the parameter *extra* according to Lemma 8. For each 0-node we compute the edge search number and the parameter *extra* according to Lemma 5. Thus we have that the edge search number of a cograph can be computed in linear time. □

4 Conclusions

We have shown how to compute the edge search number of cographs in linear time. It remains an open problem whether the edge search number of permutation graphs can be computed in polynomial time. Both answers to this questions would be interesting. If it turns out that the edge search number for permutation graphs is NP-hard, this would give the first graph class where node and mixed search number are computable in polynomial time and the edge search number computation is NP-hard.

Acknowledgment

The authors would like to thank Petr Golovach for useful discussions.

References

[1] S. Alpern and S. Gal, *The theory of search games and rendezvous*, International Series in Operations Research & Management Science, 55, Kluwer Academic Publishers, Boston, MA, 2003.

[2] B. Alspach and D. Dyer and D. Hanson and B. Yang, *Lower Bounds on Edge Searching*, Proceedings of ESCAPE 2007, Lecture Notes in Computer Science 4614, Springer, 2007, pp. 516–527.

[3] D. Bienstock, *Graph searching, path-width, tree-width and related problems (a survey)*, DIMACS Ser. in Discrete Mathematics and Theoretical Computer Science, 5 (1991), pp. 33–49.

[4] D. Bienstock and P. Seymour, *Monotonicity in graph searching*, J. Algorithms, 12 (1991), pp. 239–245.

[5] H. L. Bodlaender, T. Kloks, and D. Kratsch, *Treewidth and pathwidth of permutation graphs*, SIAM J. Disc. Math., 8 (1995), pp. 606–616.

[6] H. L. Bodlaender, T. Kloks, D. Kratsch and R. H. Möhring, *Treewidth and Minimum Fill-in on d-Trapezoid Graphs*, J. Graph Algorithms Appl., 2 (1998).

[7] H. L. Bodlaender and R. H. Möhring, *The pathwidth and treewidth of cographs*, Proceedings of SWAT 1990, Lecture Notes in Computer Science 447, Springer, 1990, pp. 301–310.

[8] K. S. Booth and G. S. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms*, J. Comp. Syst. Sc., 13 (1976), pp. 335–379.

[9] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph classes: a survey*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[10] H. Chou, M. Ko, C. Ho, and G. Chen, *Node-searching problem on block graphs*, Disc. Appl. Math., 156 (2008), pp. 55–75.

[11] D. G. Corneil, H. Lerchs, and L. Stewart Burlingham, *Complement reducible graphs*, Annals Discrete Math., 1 (1981), pp. 145–162.

[12] D. G. Corneil, Y. Perl, and L. K. Stewart. *A linear recognition algorithm for cographs.* SIAM Journal on Computing 14, 1985, pp. 926–934.

[13] J. Ellis and M. Markov, *Computing the vertex separation of unicyclic graphs*, Inf. Comput. 192, 2004, pp. 123–161.

[14] F.V. Fomin and P. Fraigniaud and N. Nisse, *Nondeterministic Graph Searching: From Pathwidth to Treewidth*, Algorithmica, 2008.

[15] F. Fomin, P. Heggernes, and R. Mihai, *Mixed search number and linear-width of interval and split graphs*, Proceedings of WG 2007, Lecture Notes in Computer Science 4769, 2007, pp. 304–315.

[16] F. Fomin and D. Thilikos, *An annotated bibliography on guaranteed graph searching*, Theor. Comput. Sci. 399, 2008, pp. 236–245.

[17] P. A. Golovach, *Extremal search problems on graphs*, Ph.D Thesis, Leningrad, 1990.

[18] P. A. Golovach and N. N. Petrov, *Some generalizations of the problem on the search number of a graph*, Vestn. St. Petersbg. Univ., Math. 28, 3 (1995), pp. 18–22 ; translation from Vestn. St-Peterbg. Univ., Ser. I, Mat. Mekh. Astron. 1995, 3 (1995), pp. 21–27.

[19] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*, North-Holland, 2004.

[20] J. Gustedt, *On the pathwidth of chordal graphs*, Disc. Appl. Math., 45 (1993), pp. 233–248.

[21] M. Habib and C. Paul. A simple linear time algorithm for cograph recognition. *Discrete Applied Mathematics*, 145:183–197, 2005.

[22] P. Heggernes, and R. Mihai, *Mixed search number of permutation graphs*, Proceedings of FAW 2008, Lecture Notes in Computer Science 5059, 2008, pp. 196–207.

[23] L. M. Kirousis and C. H. Papadimitriou, *Interval graphs and searching*, Disc. Math., 55 (1985), pp. 181–184.

[24] M. Kirousis and C. H. Papadimitriou, *Searching and pebbling*, Theor. Comput. Sci., 47 (1986), pp. 205–218.

[25] T. Kloks, D. Kratsch, and J. Spinrad, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theor. Comp. Sc., 175 (1997), pp. 309–335.

[26] A. S. LaPaugh, *Recontamination does not help to search a graph*, J. ACM, 40 (1993), pp. 224–245.

[27] F. Mazoit and N. Nisse, *Monotonicity of non-deterministic graph searching*, Theor. Comput. Sci., 3, 399 (2008), pp. 169–178.

[28] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, *The complexity of searching a graph*, J. ACM, 35 (1988), pp. 18–44.

[29] D. Meister, *Computing treewidth and minimum fill-in for permutation graphs in linear time*, Proceedings of WG 2005, Lecture Notes in Computer Science 3787, Springer, 2005, pp. 91–102.

[30] T. Parsons, *Pursuit-evasion in a graph*, in Theory and Applications of Graphs, Springer-Verlag, 1976.

[31] S.-L. Peng, M.-T. Ko, C.-W. Ho, T.-s. Hsu, and C. Y. Tang, *Graph searching on some subclasses of chordal graphs*, Algorithmica, 27 (2000), pp. 395–426.

[32] S.-L. Peng, C.-W. Ho, T.-s. Hsu, M.-T. Ko, and C. Y. Tang, *Edge and node searching problems on trees*, Theor. Comput. Sci., 240 (2000), pp. 429–446.

[33] S.-L. Peng, Y.-C. Yang, *On the Treewidth and Pathwidth of Biconvex Bipartite Graphs*, Proceedings of TAMC 2007, pp. 244–255.

[34] N. N. Petrov, *A problem of pursuit in the absence of information on the pursued*, Differentsialnye Uravneniya, 18 (1982), pp. 1345–1352, 1468.

[35] K. Skodinis. *Construction of linear tree-layouts which are optimal with respect to vertex separation in linear time*, J. Algorithms, 47 (2003), pp. 40–59.

[36] K. Suchan and I. Todinca, *Pathwidth of circular-arc graphs*, Proceedings of WG 2007, Lecture Notes in Computer Science 4769, 2007, pp. 258–269.

[37] A. Takahashi, S. Ueno, and Y. Kajitani, *Mixed searching and proper-path-width*, Theor. Comput. Sci., 137 (1995), pp. 253–268.

[38] B. Yang, R. Zhang, and Y. Cao, *Searching Cycle-Disjoint Graphs*, Proceedings of COCOA 2007, pp. 32–43.