Edge search number of cographs*

Petr A. Golovach† Pinar Heggernes‡ Rodica Mihai‡

Abstract

We give a linear-time algorithm for computing the edge search number of cographs, thereby resolving the computational complexity of edge searching on this graph class. To achieve this we give a characterization of the edge search number of the join of two graphs. 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 [3, 4, 14, 16, 26, 28, 29, 30, 33, 40] 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 [32] and by Petrov [36] 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, i.e., 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 [27] 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 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 slid 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, 26, 30] on general graphs.

*Preliminary versions of some of the results of this this work have appeared as [17] (in Russian), [18, 19] (without proofs), and [25] (extended abstract).

†School of Engineering and Computing Sciences, Durham University, South Road, DH1 3LE Durham, UK. Email: petr.golovach@durham.ac.uk. Supported by EPSRC grant EP/G043434/1.

‡Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: pinar.heggernes@ii.uib.no, rodica.g.mihai@iris.no. Supported by the Research Council of Norway.

Polynomial-time algorithms are known for computing the node search number of trees [34, 37], interval graphs [8], k-starlike graphs for fixed k [33], d-trapezoid graphs [6], block graphs [10], split graphs [22], circular-arc graphs [38], permutation graphs [5, 31], biconvex bipartite graphs [35], 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 [24] can be computed in polynomial time. Edge search number of trees [30, 34], interval graphs, split graphs [20, 33], unicyclic graphs [40], and complete multipartite graphs [2] can be computed in polynomial time.

In this paper, we give a characterization of the edge search number of the join of two graphs. Using this characterization we give a linear-time algorithm for computing the edge search number of cograph. The computational complexity of edge searching on cographs has been open until now. Since cographs are subclass of permutation graphs [9, 21], 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. Observe that 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.

2 Preliminaries

We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$. In general we use $n = |V|$. The set of *neighbors*, or the *(open) neighborhood*, of a vertex v is denoted by $N_G(v) = \{u \mid uv \in E\}$. The *closed neighborhood* of v is the set $N_G[v] = N_G(v) \cup \{v\}$. The *degree* of a vertex v is $d_G(v) = |N(v)|$. We omit the subscripts if graphs under consideration are clear from the context. A vertex v is *universal* if $d(v) = n - 1$, *pendant* if $d(v) = 1$, and *isolated* if $d(v) = 0$. We say that an edge is *pendant* if it has an endpoint of degree 1.

A vertex set is a *clique* if all of its vertices are pairwise 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 $G-u$.

By K_n we denote the complete graph on n vertices, I_n is the graph on n isolated vertices with no edges, and the complete bipartite graph (X, Y, E) with $|X| = n$ and $|Y| = m$ is denoted by $K_{n,m}$.

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 [12, 23].

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* $\mathcal{S}_e = (s_1, \ldots, s_r)$ consists of a sequence of discrete steps (moves) $s_1, \ldots, s_r$ which involves searchers. Initially there is no searcher on the graph. Every step is one of the following three types:

- One searcher is placed on some vertex of G (there can be several searchers located on one vertex).
- One searcher is 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. Note that a vertex is cleared when a searcher is placed on it.

A cleared edge e is recontaminated 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 as well as the vertex u.

A search strategy is *winning* if after its termination all edges and vertices 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; i.e., edges cannot be cleared by sliding. Clearly, for node searching, sliding of searchers along edges is not necessary, and each step in the search strategy here is either placing a searcher on some vertex or removing a searcher. In mixed searching, an edge can be cleared both by sliding and by placing searchers on both its endpoints. 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. For the graph I_1, it is assumed that $es(I_1) = ns(I_1) = ms(I_1) = 1$.

The following result is central; it gives the relation between the three graph searching parameters and relates them to pathwidth.

Lemma 1 ([39]) *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 the complexity of computing the difference between these two parameters and the edge search number is unknown. Hence even if $pw(G)$ 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 $es(G)$ searchers, or a winning node search strategy using $ns(G)$ searchers, 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, 28], i.e., on any graph with search number k, there exists a winning monotone search strategy using k searchers. Hence, to find an optimal strategy it is sufficient to consider only monotone strategies.

In this paper we consider only monotone search strategies. Furthermore, to be able to give a precise description of the search strategies in our results, we describe more restrictions on the search strategies that we consider. We call a search strategy $\mathcal{S}$ *efficient* if the following statements are true for $\mathcal{S}$: searchers are never placed on vertices that are incident to only cleared edges, searchers are removed as soon as they become unnecessary, and in the last steps of $\mathcal{S}$ all searchers are removed from the graph. If $\mathcal{S}$ is a node search strategy then we have the additional requirement that searchers are never placed on a vertex already occupied by a searcher. If $\mathcal{S}$ is an edge search strategy then we have the additional requirement that searchers never slide along cleared edges. It is easy to check that if there is an optimal edge or node search strategy with k searchers there is also an optimal edge or node search strategy with k searchers that is efficient. We consider only efficient search strategies. Note in particular that, in an efficient search strategy, if after a step a searcher occupies a vertex incident to only cleared edges then in subsequent steps such searchers are removed from the graph.

3 A closer relation between edge and node searching

From Lemma 1 we see that for a given graph G, we might have $es(G) > ns(G)$ or $es(G) < ns(G)$. In this section, we will distinguish between the cases when the edge search number is at most the node search number and when the edge search number is at least the node search number. In particular, we will describe some graphs G such that $es(G) \geq ns(G)$ and some graphs G such that $es(G) \leq ns(G)$. To do so, we will generate a particular node search strategy from a given edge search strategy, and generate a particular edge search strategy from a given node search strategy. Hence we will obtain a closer relation between edge and node searching on specific graphs. These results will be used to give a characterization of the edge search number of the join of two graphs in the next section.

First we study graphs G for which $es(G) \geq ns(G)$. Let G be a graph, and let $\mathcal{S}_e = (s_1, \ldots, s_p)$ be a winning edge search strategy for k searchers on G. We consecutively consider steps s_i for $1 \leq i \leq p$ and construct the node search strategy $\mathcal{N}(\mathcal{S}_e) = (s'_1, \ldots, s'_q)$ as follows:

- If, at step s_i of $\mathcal{S}_e$, a searcher is placed on a vertex u which is not occupied by other searchers, then a searcher is placed on u in $\mathcal{N}(\mathcal{S}_e)$.
- If, at step s_i of $\mathcal{S}_e$, a searcher is removed from a vertex u and no searchers remain on u, then a searcher is removed from u in $\mathcal{N}(\mathcal{S}_e)$.
- If at, step s_i of $\mathcal{S}_e$, a searcher slides from a vertex u to v along uv, then
  - if v is not occupied by searchers before step s_i then a searcher is placed on v in $\mathcal{N}(\mathcal{S}_e)$;
  - if no searchers remain on u after step s_i then a searcher is removed from u at the next step in $\mathcal{N}(\mathcal{S}_e)$.
- Otherwise we do not include new steps in $\mathcal{N}(\mathcal{S}_e)$.

Notice that it can happen that the search strategy $\mathcal{N}(\mathcal{S}_e)$ does not satisfy the efficiency condition that searchers are removed from the graph as soon as they become unnecessary. In this case the search strategy $\mathcal{N}(\mathcal{S}_e)$ can be modified afterwards by applying the following rule. If, at step s in $\mathcal{N}(\mathcal{S}_e)$, a searcher is placed on a vertex, then all searchers that occupy vertices which are incident only with cleared edges, are removed from the graph before s (in any order), and then step s is complete. Correspondingly, if there were steps at which searchers were removed from these vertices after step s, then these steps are deleted from the search strategy. We call these modifications the *finalization of* $\mathcal{N}(\mathcal{S}_e)$.

It is easy to observe that $\mathcal{N}(\mathcal{S}_e)$ is a winning node search strategy on G for $k+1$ searchers. For some cases it is possible to choose $\mathcal{S}_e$ so that $\mathcal{N}(\mathcal{S}_e)$ is in fact a winning node search strategy for k searchers.

Lemma 2 *Let G be a connected graph different from K_2 and without vertices of degree two. If $es(G) \leq k$ then there is a winning edge search strategy $\mathcal{S}_e$ for k searchers on G such that $\mathcal{N}(\mathcal{S}_e)$ is a winning node search strategy for k searchers on G.*

Proof. Let $\mathcal{S}_e = (s_1, \ldots, s_p)$ be a winning edge search strategy for k searchers on G which is monotone and efficient. Consider $\mathcal{N}(\mathcal{S}_e) = (s'_1, \ldots, s'_q)$ before finalization. If $\mathcal{N}(\mathcal{S}_e)$ is a winning node search strategy for k searchers, we are done. Assume that we need $k+1$ searchers for $\mathcal{N}(\mathcal{S}_e)$. We modify $\mathcal{S}_e$ inductively to get a strategy of the type we are looking for.

Let s'_i be the first step of $\mathcal{N}(\mathcal{S}_e)$ at which there are $k+1$ searchers present on the graph. Clearly, a searcher is placed on some vertex at this step of $\mathcal{N}(\mathcal{S}_e)$. Assume that this step was included in $\mathcal{N}(\mathcal{S}_e)$ while we considered the step s_j in $\mathcal{S}_e$. Observe that step s_j of $\mathcal{S}_e$ has then the following properties:

1. at step s_j, a searcher slides from some vertex u to a vertex v along edge $uv \in E(G)$;
2. there are no searchers on u after step s_j;
3. v was not occupied by searchers before step s_j;

4. before step s_j, uv was contaminated and it was the unique contaminated edge incident with u;

5. the following was true before step s_j and is true after step s_j: exactly k searchers are present on G, and each vertex is occupied by at most one searcher.

We call a step with these properties an *extremal* step. Notice that s_j is the first extremal step in $\mathcal{S}_e$, and it is easy to see that $j > 0$ since there are no searchers on G in the beginning of the game.

Consider step s_{j-1} of $\mathcal{S}_e$. This step is not extremal, since s_j is the first extremal step. Furthermore searchers cannot be removed from the graph at step s_{j-1}, since otherwise we would have at most $k-1$ searchers on the graph before step s_j. Hence, at step s_{j-1} a searcher either is placed on some vertex x or is slid from a vertex x to y along edge xy. In the last case at least one searcher remains in x, and y is not occupied by any searcher before s_{j-1}; otherwise either s_{j-1} would be extremal or we would be able to remove a searcher contradicting that there are k searchers before step s_j.

If $x \neq u$ then we modify $\mathcal{S}_e$ by changing the order of steps s_j and s_{j-1}. If a searcher was placed on x at step s_{j-1} of $\mathcal{S}_e$ then at most $k-1$ searchers were occupying vertices of G before step s_{j-1}. Therefore, if we do step s_j before step s_{j-1}, there will be at most $k-1$ searchers on G after the completion of the first, and at most k searchers after the completion of both. Note that the set of cleared edges after these steps s_j and s_{j-1} is same as before. Furthermore, neither of the steps is extremal anymore. Consequently, by swapping the two steps, we reduced the number of extremal steps. Suppose that a searcher slides along xy at step s_{j-1} of $\mathcal{S}_e$. Since $d(u) \neq 2$ and uv is the unique contaminated edge incident with u before step s_j, $y \neq u$. Since there was no searcher on v before step s_j, $y \neq v$ either. Furthermore, there are two searchers on x before step s_{j-1}; otherwise s_{j-1} would be extremal. Therefore we can again change the order of steps s_j and s_{j-1}, and the set of cleared edges remains the same after the completion of both steps. However, in this new strategy s_j and s_{j-1} are not extremal.

Assume now that $x = u$. Now we choose the first non-extremal step s_r after step s_j in $\mathcal{S}_e$. Suppose for contradiction, that s_r does not exist. This means that the last step in $\mathcal{S}_e$ is extremal. Recall that we assumed that searchers are removed from the graph if they occupy vertices incident only with cleared edges. Hence $k = 1$, and G is a path. Since G has no vertices of degree two, $G = K_2$ but this contradicts the premises of the lemma. Hence s_r does exist. Now we consider two cases.

Case 1: $r > j+1$. Then step s_{r-1} is extremal. Hence at this step a searcher slides from some vertex w. Observe that this searcher was placed or slid to w before s_{j-1}. Indeed, $w \neq u$ since a searcher was moved from u along the unique contaminated edge at step s_j, and hence a searcher could not be placed or moved to w at step s_{j-1}. Also the subsequent steps $s_j, \ldots, s_{r-2}$ are extremal, and a searcher could not be slid to w at these steps since $d(w) \neq 2$. We modify $\mathcal{S}_e$ by replacing the steps $s_{j-1}, s_j, \ldots, s_{r-1}$ with the steps $s_{r-1}, s_{j-1}, s_j, \ldots, s_{r-2}$. By same arguments as were used for the case $x \neq u$, we conclude that the step s_{r-1} is not extremal in the new strategy, and the set of cleared edges after $s_{r-1}, s_{j-1}, s_j, \ldots, s_{r-2}$ is the same as the set of cleared edges after s_{r-1} before.

Case 2: $r = j + 1$. Then at step s_r a searcher either is removed from some vertex w or is slid from some vertex z to a vertex w.

Suppose that a searcher is removed from w. Recall that we assumed that searchers are removed from the graph as soon as they occupy a vertex incident with only cleared edges. The

removal of a searcher from w became possible when a searcher was moved to v at step s_j. It follows that $w = v$, and $d(v) = 1$ since uv was contaminated before step s_j and there were no searchers on v before this step. Observe that it could not happen that at step s_{j-1} a searcher was placed on u, because in this case $d(u) = 1$ and therefore $G = K_2$. Hence at step s_{j-1} a searcher was moved from u to y. This means that we can reorder the steps s_{j-1}, s_j, s_{j+1} as follows without changing the set of cleared edges: s_j, s_{j+1}, s_{j-1}. Now these steps are not extremal.

Assume now that a searcher slides from z to w along an edge zw. Note that after step s_{j+1}, there are no searchers on vertex z and there are two searchers on vertex w. Note also that $z \neq u$, and $z \neq v, y$ since $d(z) \neq 2$. We modify $\mathcal{S}_e$ as follows:

- if $w = y$ then the steps s_{j-1}, s_j, s_{j+1} are replaced by the steps s_{j-1}, s_{j+1}, s_j;
- if $w = v$ then the steps s_{j-1}, s_j, s_{j+1} are replaced by the steps s_j, s_{j+1}, s_{j-1};
- if $w \neq v, y$ then the steps s_{j-1}, s_j, s_{j+1} are replaced by the steps s_{j+1}, s_{j-1}, s_j.

Again, it is easy to verify that the set of cleared edges remains the same after the completion of these steps in the changed order. Observe that if $w = y$ or $w \neq v, y$ then y is occupied by two searchers before step s_j is performed and hence this step is not extremal, and if $w = v$ then u is occupied by two searchers before step s_j and there are two searchers on v before s_{j-1}, so these steps are not extremal. It follows that the number of extremal steps is reduced.

We showed that it is possible to reduce the number of extremal steps in $\mathcal{S}_e$. It can happen that after our modifications the strategy does not satisfy our condition that searchers are removed from a vertex immediately after all incident edges are cleared, but it can easily be corrected by obvious changes in $\mathcal{S}_e$. By repeating these modifications, if necessary, we get a winning edge search strategy for k searchers without extremal steps. Then $\mathcal{N}(\mathcal{S}_e)$ is a winning node search strategy for k searchers. □

Lemma 2 immediately gives us the following theorem.

Theorem 3 *Let G be a connected graph different from K_2 and without vertices of degree two. Then $es(G) \geq ns(G)$.*

We say that a node search strategy $\mathcal{S}_n$ for k searchers on a graph G is *reducible* if the following is true for every step s_i of $\mathcal{S}_n$: if a searcher is placed on a vertex u of degree at least 3 at step s_i, and this searcher is removed from u at step s_{i+1}, then there are at most $k-1$ searchers on G after step s_i (before the removal of the searcher on u at step s_{i+1}). In other words, G is reducible if, whenever $k-1$ searchers are on G, the next two steps are not to place a searcher at a vertex of degree at least 3 all neighbors of which are occupied, and to remove this searcher immediately afterwards.

Observe now that node search strategies $\mathcal{N}(\mathcal{S}_e)$ that we generated from edge search strategies $\mathcal{S}_e$ are reducible.

Lemma 4 *Let G be a graph with $es(G) \leq k$, and let $\mathcal{S}_e$ be a winning edge search strategy for k searchers on G such that $\mathcal{N}(\mathcal{S}_e)$ is a node search strategy for k searchers. Then $\mathcal{N}(\mathcal{S}_e)$ is reducible.*

Proof. The claim follows from the construction of $\mathcal{N}(\mathcal{S}_e)$ and the following observation. If a searcher is placed on u in $\mathcal{N}(\mathcal{S}_e)$, then a searcher is placed or slid to u at the corresponding step of $\mathcal{S}_e$, and the edges incident to u were contaminated before this step. Then at least two edges incident with u need to be cleaned, and to do it we have to place or slide one more searcher to u before we can remove a searcher from this vertex. □

Next we give some conditions for a graph G to satisfy $ns(G) \geq es(G)$. Let G be a graph, and let $\mathcal{S}_n = (s_1, \ldots, s_p)$ be a winning node search strategy for k searchers on G. We consecutively consider steps s_i for $1 \leq i \leq p$, and construct an edge search strategy $\mathcal{E}(\mathcal{S}_n) = (s'_1, \ldots, s'_q)$ as follows:

- If, at step s_i of $\mathcal{S}_n$, a searcher is removed from a vertex u, then a searcher is removed from u in $\mathcal{E}(\mathcal{S}_n)$.
- If, at step s_i of $\mathcal{S}_n$, a searcher is placed on a vertex u, then we consider four cases:
  1. $d(u) = 0$. A searcher is placed on u in $\mathcal{E}(\mathcal{S}_n)$.
  2. $d(u) = 1$. Let v be the vertex adjacent to u. If $d(v) = 1$ or v is occupied by a searcher, then a searcher is placed on v and then slid to u in $\mathcal{E}(\mathcal{S}_n)$. Otherwise a searcher is placed on u in $\mathcal{E}(\mathcal{S}_n)$.
  3. $d(u) = 2$. Let v and w be the vertices adjacent to u. If for one of these vertices, say v, $d(v) = 1$ or v is occupied by a searcher, then a searcher is placed on v and then slid to u in $\mathcal{E}(\mathcal{S}_n)$. If the same condition holds for w, then this searcher is slid to w and then returned to u in $\mathcal{E}(\mathcal{S}_n)$. Otherwise a searcher is placed on u in $\mathcal{E}(\mathcal{S}_n)$.
  4. $d(u) \geq 3$. A searcher is placed on u in $\mathcal{E}(\mathcal{S}_n)$. If there is a vertex $w \in N(u)$ occupied by a searcher such that uw is the unique contaminated edge incident to w before step s_i, then the searcher from w is slid to u at the next step of $\mathcal{E}(\mathcal{S}_n)$, and then at subsequent steps of $\mathcal{E}(\mathcal{S}_n)$, for each vertex $v \in N(u)$ with $v \neq w$, such that $d(v) = 1$ or v is occupied by a searcher, this searcher is slid along uv and then returned to u, and finally the searcher is removed from u and placed on w. Otherwise, for each vertex $v \in N(u)$, such that $d(v) = 1$ or v is occupied by a searcher, we put an additional searcher on v, slide this searcher to u, and then remove him from the graph, at subsequent steps of $\mathcal{E}(\mathcal{S}_n)$.

It is easy to see that $\mathcal{E}(\mathcal{S}_n)$ is a winning edge search strategy on G for $k+1$ searchers. We will show that it is in fact a winning strategy on G for k searchers. Moreover, it is even a winning edge search strategy on particular supergraphs of G. Let H be a supergraph of a graph G. Then a node search strategy $\mathcal{S}_n$ on G can be considered as a (not necessarily winning) node search strategy on H, and it is possible to construct an edge search strategy for H from this node search strategy, as described above. To distinguish these cases, we denote the edge search strategy generated for H from $\mathcal{S}_n$ as described above, by $\mathcal{E}(\mathcal{S}_n, H)$.

Lemma 5 *Let G be a graph, let G' be a graph obtained by adding some pendant vertices adjacent to some vertices of G, and let $\mathcal{S}_n$ be a reducible winning node search strategy on G for k searchers. Then*

- *$\mathcal{E}(\mathcal{S}_n)$ is a winning edge search strategy for k searchers on G, and*

- *if $\mathcal{S}_n$ is a reducible node search strategy on G', then $\mathcal{E}(\mathcal{S}_n, G')$ is a winning edge search strategy for k searchers on G'.*

Proof. Observe that $\mathcal{E}(\mathcal{S}_n, G')$ is a winning edge search strategy on G' for $k+1$ searchers. This follows from the construction of the edge strategy and the way pendant edges are cleared. It remains to prove that $\mathcal{E}(\mathcal{S}_n)$ and $\mathcal{E}(\mathcal{S}_n, G')$ are winning edge search strategies for k searchers on G and G' respectively. It is sufficient to prove it for $\mathcal{E}(\mathcal{S}_n, G')$. Notice that we need the searcher number $k+1$ in $\mathcal{E}(\mathcal{S}_n, G')$ only if at some step s_i in $\mathcal{S}_n$ we place a searcher on a vertex u of degree at least 3 in G' (observe that this vertex can have degree 1 or 2 in G) such that all vertices $v \in N_{G'}(u)$ occupied by searchers are incident to at least one contaminated edge different from uv. Also s_i is not the last step in $\mathcal{S}_n$ since the searchers are removed from the graph in concluding steps. Consider the step s_{i+1} of $\mathcal{S}_n$ on G'. If a searcher is placed on some vertex at this step then we have at most $k-1$ searchers on G after step s_i and we do not need the searcher number $k+1$. If a searcher is removed from u at this step then there are at most $k-1$ searchers on G' after step s_i because of the reducibility of $\mathcal{S}_n$. Searchers cannot be removed from the vertices of $N_{G'}(u)$ due to monotonicity, since each of them is incident to at least one contaminated edge after step s_i. Hence, a searcher is removed from some vertex $w \notin N_{G'}[u]$. This can be done only if all edges incident with w are cleared, but then by the efficiency of the considered search strategies, the searcher should have been removed from w before step s_i. □

4 Edge search number of the join of two graphs

In this section we characterize the edge search number of the join of two arbitrary graphs. Given an arbitrary graph G and a positive integer c, we define G_c to be the supergraph of G obtained by adding c pendant vertices adjacent to each vertex of G. Hence G_c has $c \cdot |V(G)|$ vertices in addition to the $|V(G)|$ vertices of G.

Lemma 6 *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: $\{I_1, I_1\}$, $\{I_1, I_2\}$, $\{I_2, I_2\}$, $\{I_2, K_p\}$ for $p \geq 2$. Then $es(G \otimes H) = \min\{es(G_m)+m,\ es(H_n)+n\}$.*

Proof. Let $r = \min\{es(G_m)+m,\ es(H_n)+n\}$. The inequality $es(G \otimes H) \leq r$ is easy to obtain. Assume that $r = es(G_m)+m$. We place a searcher on each vertex of H, in total m searchers. We clear G and all edges between G and H using the remaining $r - m = es(G_m)$ searchers. Then we remove searchers from the vertices of G, and use one of them to clear all edges of H by sliding on each of them one by one. The case $r = es(H_n)+n$ is symmetric.

Now we prove that $es(G \otimes H) \geq r$. Let $k = es(G \otimes H)$ and $F = G \otimes H$. Assume first that F has no vertices of degree two. By Lemma 2, there is a winning edge search strategy $\mathcal{S}_e$ for k searchers on F, such that $\mathcal{N}(\mathcal{S}_e)$ is a winning node strategy on F for k searchers. As argued earlier, we can assume that $\mathcal{S}_e$ is monotone and efficient. Observe that no searchers can be removed from F in $\mathcal{N}(\mathcal{S}_e)$ until searchers are placed either on all vertices of G or on all vertices of H; otherwise some edges would be recontaminated. Assume that after some step in $\mathcal{N}(\mathcal{S}_e)$ all vertices of G are occupied by searchers. Note that searchers cannot be removed from G in $\mathcal{N}(\mathcal{S}_e)$ until all edges of F are cleared. Otherwise G would be recontaminated.

We modify the search strategy $\mathcal{N}(\mathcal{S}_e)$ on F to construct a node search strategy $\mathcal{S}_n = (s_1, \ldots, s_p)$ on H as follows:

- remove the steps of $\mathcal{N}(\mathcal{S}_e)$ at which searchers are either placed on or removed from the vertices of G;
- remove "obsolete" steps at which searchers are placed on, and subsequently removed from, vertices not incident with contaminated edges.

Observe that $\mathcal{S}_n$ is a winning node search strategy on H for $k-n$ searchers. Now we consider two cases.

Case 1: $k-n>1$. Consider the strategy $\mathcal{S}_n$ on the supergraph H_n of H. If $\mathcal{S}_n$ is reducible on H_n, then by Lemma 5, $es(H_n)\leq k-n$ and hence $k\geq es(H_n)+n\geq r$. Suppose that $\mathcal{S}_n$ is not reducible on H_n, and there is a step s_i in $\mathcal{S}_n$, such that a searcher is placed on a vertex $u\in V(H)$ with $d_F(u)\geq 3$, this searcher is removed from u at the next step, and there are $k-n$ searchers on H after step s_i. Notice that this can happen only once, if at the steps $s_1,\ldots,s_i$ searchers were placed on vertices of H, because the strategy $\mathcal{N}(\mathcal{S}_e)$, from which $\mathcal{S}_n$ was constructed, is a reducible strategy on F by Lemma 4. Observe also that $i>1$, since $k-n\geq 2$. In this case we modify $\mathcal{S}_n$ by changing the order of steps s_{i-1} and s_i. Now we can apply Lemma 5 and obtain that $es(H_n)\leq k-n$. Hence $k\geq es(H_n)+n\geq r$.

Case 2: $k-n=1$. Since $\mathcal{S}_n$ is a winning node search strategy for one searcher on H, $H=I_\ell$ for some $\ell\geq 1$.

We claim that either $\ell\leq 2$, or $\ell\geq 3$ and $G=K_1$. To prove it, assume that $\ell\geq 3$ and $G\neq K_1$. Since F has no vertices of degree 2, $n\geq 3$. Now recall that according to $\mathcal{N}(\mathcal{S}_e)$, no searchers are removed from F until searchers are placed on all vertices of G, and these searchers cannot be removed from G until all edges of F are cleared. When searchers are placed on G, at most one vertex of H is occupied by a searcher, and we have only one searcher to clear all contaminated edges of F. To clear them, we have to consecutively place a searcher on at least two remaining vertices of H, and this cannot be done without placing and immediately removing a searcher. This means that $\mathcal{N}(\mathcal{S}_e)$ on F is not reducible, which contradicts Lemma 4.

If $G=K_1$ and $H=I_\ell$ for $\ell\geq 3$ then $es(F)=2=es(H_1)+1=es(H_n)+n\geq r$. If $H=I_1$ then $es(F)=k=n+1=|V(F)|$. It means that $F=K_{n+1}$ and $n\geq 3$. Hence, $k=es(G_1)+1=es(G_m)+m\geq r$. Suppose now that $H=I_2$. Since F has no vertices of degree two, $n\geq 3$. By the premises of the lemma, H is different from K_n. It follows immediately that $es(G_2)\leq n-1$ and $k=n+1\geq es(G_2)+2=es(G_m)+m\geq r$.

Assume now that F contains vertices of degree two. Clearly, $\min\{n,m\}\leq 2$ for this case. Suppose $n\leq m$ and $n\leq 2$. Consider two cases.

Case 1: $n=1$. Denote the single vertex of G by z. We prove that $k\geq es(H_1)+1$. Let U be the set of vertices of degree at least two in H. If $U=\emptyset$ then H is a disjoint union of isolated vertices and copies of K_2, and by the premises of the lemma, either at least one component is K_2 or there are at least 3 isolated vertices. Therefore, $es(H_1)=1$ and $es(F)\geq 2$. Now it is sufficient to prove that $k\geq es(H_1)+1$ for connected H; otherwise we can consider connected components of H separately. If $|U|=1$ or $|U|=2$ then it can be verified directly that $es(F)=3=es(H_1)+1$.

For $|U|\geq 3$, we can use similar arguments as for the case when F has no vertices of degree two. Let F' be the graph obtained from F by deleting the edge zx for every $x\in V(H)\setminus U$. This graph has no vertices of degree two. Clearly $es(F')\leq es(F)$. Using Lemma 2 we conclude that there is a winning edge search strategy $\mathcal{S}_e$ for k searchers on F', which is efficient and monotone, such that $\mathcal{N}(\mathcal{S}_e)$ is a winning node strategy for k searchers. Denote by H' the graph

$H[U]$. Since we assumed that H is connected, the graph H' is connected, too. Consider the node search strategy $\mathcal{S}_n$ on H' obtained by removing the steps of $\mathcal{N}(\mathcal{S}_e)$ operating on removed vertices of F'.

Now we prove that $\mathcal{S}_n$ demands only $k-1$ searchers. Assume the opposite, and that after some step there are k searchers on H'. At this step some searcher is placed on a vertex u incident with contaminated edges. Notice that z could not be occupied after the corresponding step of $\mathcal{N}(\mathcal{S}_e)$, since the edges incident with z in F' are contaminated. It remains to note that now it is impossible to remove any searcher from the graph without violating monotonicity. This contradiction proves our claim.

Since H' is a connected graph with at least 3 vertices, $2 \leq ns(H') \leq k-1$ and $k-1 = k-n \geq 2$. Let $p = m - |U| + 1$. Consider the node search strategy $\mathcal{S}_n$ on the supergraph H'_p of H'. If $\mathcal{S}_n$ is reducible on H'_p, then by Lemma 5, $es(H'_p) \leq k-1$. Observe that H_1 can be obtained by selecting some pendant vertices in a subgraph of H'_p and adding a pendant vertex adjacent to each of these vertices. It follows that $es(H_1) \leq es(H'_p)$ and we conclude that $es(H_1) \leq k-1$. Suppose that $\mathcal{S}_n$ is not reducible and there is a step s_i in $\mathcal{S}_n$ such that a searcher is placed on a vertex $u \in V(H')$ with $d_{H'_p}(u) \geq 3$, this searcher is removed from u at the next step, and there are $k-1$ searchers on H' after step s_i. Notice that this can happen only once, if at the steps $s_1, \ldots, s_i$ searchers were placed on vertices of H', because the strategy $\mathcal{N}(\mathcal{S}_e)$, from which $\mathcal{S}_n$ was constructed, is a reducible strategy on F' by Lemma 4. Observe also that $i > 1$, since $k-1 \geq 2$. In this case we modify $\mathcal{S}_n$ by changing the order of steps s_{i-1} and s_i. Now we can apply Lemma 5 and obtain that $es(H_1) \leq es(H'_p) \leq k-1$.

Case 2: $n = 2$. We prove that $es(F) = es(H_2) + 2$. If $m = 2$ then $F = K_4$, since $\{G, H\}$ differs from the pairs $\{I_2, I_2\}$ and $\{I_2, K_2\}$, and hence $es(H_2) = 2$. Clearly $es(F) = 4 = es(H_2) + 2$. If $m = 3$ then either $H = I_1 \oplus K_2$ and $es(F) = 4 = es(H_2) + 2$, or $H = I_3$ and $es(F) = 3 = es(H_2) + 2$. Suppose that $m \geq 4$. Notice that H has an isolated vertex u and $es(F) = es(F-u)$, $es(H_2) = es(H_2-u)$, and our claim follows by induction. □

We will relate the above lemma to edge search strategies. To be able to use the above lemma for algorithmic purposes we will now relate $es(G)$ to $es(G_c)$ for a graph G and a positive integer c. We define a boolean function *extra* on an arbitrary graph G as follows. If there is an optimal edge search strategy on G, such that at every step where the maximum number of searchers is used, a searcher is slid through a contaminated edge both of whose endpoints are occupied by searchers, then $extra(G) = 1$. Otherwise $extra(G) = 0$. Hence $extra(G) = 0$ if every optimal edge search strategy avoids sliding a searcher on a contaminated edge whose endpoints are occupied by searchers, at least once when using the maximum number of searchers.

Lemma 7 *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 slid 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 slid 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 two contaminated edges incident to some vertex u occupied by a searcher. One edge can be cleared 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. If after this step u is incident with exactly one contaminated edge uv, this edge can be cleared by sliding the searcher from u to v along uv. By the same arguments as before, the vertex v was not occupied before this step, v is not a degree one vertex, and all edges between v and its neighbors of degree one are contaminated. Consequently, from now on no searcher can be removed or slid 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 is, $k + 1$ 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 6 and 7, 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 8 *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 $es(G \otimes H) = \min\{n+es(H)+1-extra(H),\ m+es(G)+1-extra(G)\}$.*

5 Edge search number of cographs

In this section we show how to compute the edge search number of a cograph. By the results of the previous section, if we know how to compute $extra(G)$ for every graph G then we can compute the edge search number of the join of two graphs whose edge search numbers we know, using the Lemma 8. Computing the *extra* value might be a difficult task for general graphs, but here we will show that we can compute $extra(G)$ efficiently when G is a cograph.

Before we continue with computing the edge search number of cographs, 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 9 *Let $G^{(1)}$ and $G^{(2)}$ be two arbitrary disjoint 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 $es(G^{(2)}) < es(G^{(2)})$ or $es(G^{(2)}) = es(G^{(1)})$. For the first case, $extra(G^{(1)} \oplus G^{(2)}) = extra(G^{(1)})$, regardless of $extra(G^{(1)})$, since we can search $G^{(1)}$ 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 [20, 33].

Lemma 10 *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 [20, 33] 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. □

Lemma 11 *If G has a universal vertex u, and the number of vertices 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 lemma. 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 lemmas, together with Lemma 9, handle the *extra* parameter of all (and more) graphs that are not covered by Lemma 8.

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 12 *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 Lemmas 10 and 11 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 9. 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 10 and 11, and we are at a join-node, we know that we are not dealing with one of the cases not covered by Lemma 8. Thus by Lemma 8 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 through 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 $es(G^{(l)} \otimes G^{(r)}) = \min\{|V(G^{(l)})| + es(G^{(r)}), |V(G^{(r)})| + es(G^{(l)}) + 1\} = |V(G^{(r)})| + es(G^{(l)}) + 1$. We construct the desired edge search strategy in the following manner. We place $|V(G^{(r)})|$ searchers on the vertices of $G^{(r)}$. After that we construct the edge search strategy in the same fashion as 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 $es(G^{(l)} \otimes G^{(r)}) = \min\{|V(G^{(l)})| + es(G^{(r)}), |V(G^{(r)})| + es(G^{(l)})\}$. This is similar to the case when $extra(G^{(l)}) = 0$ and $extra(G^{(r)}) = 1$ and $es(G^{(l)} \otimes G^{(r)}) = |V(G^{(l)})| + es(G^{(r)})$. 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 12:

Corollary 13 *If G is a connected cograph, and G is not one of the graphs covered by Observations 10 and 11, then $extra(G) = 1$.*

Theorem 14 *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 10 and 11. By starting subtrees we mean subtrees corresponding to graphs not covered by Lemma 8. By the discussion just before Observation 10 we know how to compute the edge search number of these subgraphs. By Observations 10 and 11 we also know how to compute their *extra* values. After this initialization, 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 8 and the parameter *extra* according to Lemma 12. For each 0-node we compute the edge search number and the parameter *extra* according to Lemma 9 and the discussion preceding it. Thus we have that the edge search number of a cograph can be computed in linear time. □

6 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. An answer to this question in either direction 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.

Acknowledgments

We are grateful to the anonymous referees for their constructive suggestions and remarks.

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, 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. (In Russian.)

[18] P. A. Golovach, *Node-search and search number of a combination of graphs*, Vestn. St. Petersbg. Univ., Ser. I, Mat. Mekh. Astron. 2 (1990), pp. 90–91. (In Russian.)

[19] P. A. Golovach, *Search number, node search number, and vertex separator of a graph*, Vestn. Leningr. Univ., Math. 24, No.1, pp. 88–90 (1991); translation from Vestn. Leningr. Univ., Ser. I, Mat. Mekh. Astron. 1991, 1 (1991), pp. 110–111.

[20] 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.

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

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

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

[24] 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.

[25] P. Heggernes, and R. Mihai, *Edge search number of cographs in linear time*, Proceedings of FAW 2009, Lecture Notes in Computer Science 5598, 2009, pp. 16–26.

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

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

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

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

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

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

[37] 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.

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

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

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