

REPORTS IN INFORMATICS

ISSN 0333-3590

Mixed search number and linear-width of
interval and split graphs

Fedor V. Fomin Pinar Heggernes
Rodica Mihai

REPORT NO 347

February 2007



Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL <http://www.ii.uib.no/publikasjoner/textrap/pdf/2007-347.pdf>

Reports in Informatics from Department of Informatics, University of Bergen, Norway, is available at
<http://www.ii.uib.no/publikasjoner/textrap/>.

Requests for paper copies of this report can be sent to:

Department of Informatics, University of Bergen, Høyteknologisenteret,
P.O. Box 7800, N-5020 Bergen, Norway

Mixed search number and linear-width of interval and split graphs

Fedor V. Fomin* Pinar Heggernes* Rodica Mihai*

Abstract

We show that the mixed search number and the linear-width of interval graphs and of split graphs can be computed in linear time and in polynomial time, respectively.

1 Introduction

In the graph searching problem, a team of searchers (pursuers) is trying to catch a fugitive moving along the edges of a graph. The problem is to find the minimum number of searchers that can guarantee the capture of the fugitive in the worst case scenario for the searchers (assuming that fugitive is very fast, invisible, and knows the strategy of the searchers). The study of this problem started in 1970s when it was independently introduced by Parsons [23] and Petrov [28], and since that time it has been studied extensively [2, 3, 7, 10, 9, 11, 17, 16, 21, 22, 24, 27]. 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 classical Parson-Petrov formulation of the problem is referred to as edge searching, and there are two modifications of it giving two other models. The first model, node searching, was introduced by Kirousis and Papadimitriou [16], and the second model, mixed searching, by Bienstock and Seymour [3]. The difference between the models is in the way the searchers are allowed to catch the fugitive or clear the edges of the graph. (We give formal definitions of the problems in the next section.) The minimum number of the searchers sufficient to perform searching and ensure capture for each of the models are respectively the edge, node, and mixed search numbers, and computations of these are all NP hard [3, 22, 16].

The node search number of a graph is known to be equal to its pathwidth plus one. Similarly, the mixed search number of a graph is equal to its proper pathwidth [29]. Also for every graph G of minimum vertex degree at least 2, the proper pathwidth of G is equal to the linear-width of G . While pathwidth can be seen as a “linear” variant of treewidth, linear-width is a “linear” variant of branchwidth. Whereas the computation of treewidth and pathwidth of interval and split graphs is almost straightforward, the polynomial time algorithms [19, 25]

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Emails: fedor.fomin@ii.uib.no, pinar.heggernes@ii.uib.no, rodica.mihai@ii.uib.no. Supported by the Norwegian Research Council.

computing the branchwidth of an interval graph are not trivial and computing the branchwidth of a split graph is NP-hard [19]. Such a difference between computational complexities of computing treewidth and branchwidth of split graphs was one of the motivations for our study.

Graph searching problems can be seen as vertex and edge ordering problems and computational complexity of different ordering problems on interval and split graphs is a well studied area (see e.g., the survey [6]). For example, the bandwidth minimization problem is solved in polynomial time on interval graphs [18] and is NP hard on split graphs [20]. Profile (or SumCut) problem is trivially solvable in polynomial time on interval graphs and is NP hard on split graphs [26], while Optimal Linear Arrangement is NP hard on interval graphs [5].

It is easy to show that the node search number of an interval or split graph can be computed in linear time. Peng et al. [27] show that similar result holds for edge search number. In this paper we show that the mixed search number (the proper pathwidth and the linear-width) of interval and split graphs can be computed in polynomial time. In fact, for interval graphs we obtain a linear time algorithm. For interval graphs our algorithm resembles the algorithm for the edge search number [27], but for split graphs there is a substantial difference between edge search and mixed search, and the situation with mixed search number is more involved.

2 Background, definitions, and notations

We work with simple and undirected graphs $G = (V, E)$, with vertex set $V(G) = V$ and edge set $E(G) = E$, and we let $n = |V|$, $m = |E|$. The set of *neighbors* of a vertex x is denoted by $N(x) = \{y \mid xy \in E\}$. A vertex set C is a *clique* if every two vertices in C are adjacent, and a *maximal clique* if no superset of C is a clique. The maximum size of a clique in G is denoted by $\omega(G)$. A vertex set I is an *independent set* if no two vertices of I are adjacent. The *degree* of a vertex v is $d(v) = |N(v)|$. A vertex is *isolated* if it has degree 0.

A *path* is a sequence v_1, v_2, \dots, v_p of distinct vertices of G , where $v_i v_{i+1} \in E$ for $1 \leq i < p$, in which case we say that this is a path *between* v_1 and v_p . A path v_1, v_2, \dots, v_p is called a *cycle* if $v_1 v_p \in E$. A *chord* of a cycle (path) is an edge connecting two non-consecutive vertices of the cycle (path).

A graph is *chordal* if every cycle of length at least 4 has a chord. A graph is an *interval graph* if intervals of the real line can be associated to its vertices such that two vertices are adjacent if and only if their corresponding intervals overlap. Interval graphs are chordal. A *caterpillar* is a graph that contains a chordless path such that if the vertices on this path are deleted only isolated vertices remain. Caterpillars are interval graphs and trees. A graph is a *split graph* if its vertices can be partitioned into a clique C and an independent set I , in which case (C, I) is called a *split partition* of G . A split partition of a split graph is not unique, but it is always possible to choose a partition such that C is a clique of maximum size. In this paper we will always assume that C is a clique of maximum size. For a vertex $u \in C$, we denote by $N_I(u)$ the neighbors of u in I . Split graphs are also chordal. Let us remark that split and interval graphs can be recognized in linear time. A clique of maximum size in these graphs can be found in linear time as well. (See e.g., [14]).

The parameters that we study in this paper, mixed search number and linear-width, are both closely related to a parameter called pathwidth, defined through path decompositions.

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.

- Every vertex $x \in V$ appears in some bag.
- For every edge $xy \in E$ there is a bag containing both x and y .
- 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. A path decomposition of width $pw(G)$ is called an *optimal* path decomposition of G .

By a classical result of Gillmore and Hoffman [13], a graph G is an interval graph if and only if it has an optimal path decomposition where every bag is a maximal clique of G . Such an optimal path decomposition is often called a *clique-path*. It is well known that the pathwidth of an interval graph is one less than the size of its largest clique. Clique-paths of interval graphs can be computed in linear time [4]. Consequently, the pathwidth of interval graphs can also be computed in linear time.

The *mixed search game* can be formally defined as follows. Let $G = (V, E)$ be a graph to be searched. A *search program* 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 program 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 program either if both its endpoints contain searchers, or 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 program is *winning* if after its termination all edges are cleared. The *mixed search number* of a graph G , denoted by $ms(G)$, is the minimum number of searchers required for a winning program of mixed 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 edge searching an edge can be cleared only by sliding. So mixed searching can be seen as a combination of node and edge searching. The *edge* and *node search*

numbers of a graph G are defined similarly to the mixed search number, and are denoted by $es(G)$ and $ns(G)$, respectively.

A search program is called *monotone* if at any step of this program no recontamination occurs. For all three versions of graph searching, recontamination does not help to search the graph with fewer searchers [3, 21], i.e., on any graph with $\{\text{edge, mixed, node}\}$ search number k there exists a winning monotone $\{\text{edge, mixed, node}\}$ search program using k searchers. Thus in this paper we consider only monotone search programs.

The linear-width was introduced by Thomas [31]. The *linear-width* of an arbitrary graph G , $lw(G)$, is defined to be the smallest integer $k \geq 0$ such that the edges of G can be ordered e_1, \dots, e_m in such a way that for every $i = 1, 2, \dots, m - 1$, there are at most k vertices incident both to an edge in $\{e_1, \dots, e_i\}$ and to an edge in $\{e_{i+1}, \dots, e_m\}$.

Takahashi et al. [29] proved that the mixed search number of any graph is equal to its proper pathwidth. Fomin [8] showed that proper pathwidth is also equivalent to another graph parameter, namely split bandwidth. Thus our algorithms for computing mixed search numbers of interval and split graphs can be applied to these parameters as well.

The next proposition follows directly from the results of Bienstock and Seymour [3], Fomin and Thilikos [12], and Takahashi, Ueno, and Kajitani [29].

Proposition 1 *For any graph G , $pw(G) \leq lw(G) \leq ms(G) \leq pw(G) + 1$.*

Furthermore, if G has no vertices of degree 1 then $lw(G) = ms(G)$ [3]. In fact, Thilikos showed the following stronger result [30].

Proposition 2 ([30]) *Let G be any graph, and let G' be the graph obtained by removing all vertices of degree 1 from G . Then $lw(G) = ms(G')$.*

3 The mixed search number of interval graphs

By the characterization of interval graphs through clique paths, $pw(G) = \omega(G) - 1$ for every interval graph G . Hence, by Proposition 1 we know that the mixed search number of an interval graph G is either $\omega(G) - 1$ or $\omega(G)$. In this section we characterize interval graphs G that have $ms(G) = \omega(G)$. Consequently all other interval graphs have mixed search number equal to one less than the maximum clique size.

Observation 3 *If G is a complete graph then $ms(G) = n - 1$.*

Proof. Since G is complete, $pw(G) = n - 1$. Thus by Proposition 1 we need at least $n - 1$ searchers. To see that $n - 1$ searchers are enough, place searchers on $n - 1$ vertices of G . All edges between pairs of vertices among these $n - 1$ vertices are now cleared. Slide a searcher from any cleared vertex to the remaining contaminated vertex. This clears the slided edge since all neighbors of the vacated vertex are guarded. After this, all edges and vertices of G are cleared. ■

Lemma 4 *Let G be a graph consisting of three cliques of size $n - 2$ that intersect at the same $n - 3$ vertices. Then $ms(G) = \omega(G) = n - 2$.*

Proof. Clearly G is an interval graph and $\omega(G) = n - 2$. By Proposition 1, $n - 2$ searchers are enough to clear G . Let us show that $n - 3$ searchers are not enough. Let C be the set of $n - 3$ vertices in the intersection of the three cliques, and let x, y, z be the remaining vertices. We need $n - 3$ searchers to clear $C \cup \{x\}$ by Observation 3. If we start by placing all $n - 3$ searchers on vertices of C , then we will need to slide one of them to x , and this will allow recontamination of a vertex of C from y or z . Since we know that there is always an optimal program without recontamination, we can discard this approach. Let us place all $n - 3$ searchers on x and $n - 4$ vertices of C . Then slide the searcher on x to the single unguarded vertex of C . Now all edges between pairs of vertices in $C \cup \{x\}$ are cleared. To clear y or z , we need to slide a searcher from a vertex of C to one of these vertices, say y . But then this vertex of C will become recontaminated from x . Thus it is not possible to continue the search without recontamination with only $n - 3$ searchers, and hence $ms(G) = n - 2$. ■

For the following results, we need to give more details about clique-paths. Every interval graph has a clique-path which is simply an ordering of all maximal cliques of G [13]. An interval graph has at most n maximal cliques. We will denote a clique-path of G by $(B_1, S_1, B_2, S_2, \dots, S_{c-1}, B_c)$, where B_i is a bag of the clique-path and a maximal clique of G for each $1 \leq i \leq c$, and $S_i = B_i \cap B_{i+1}$ represents the edge between B_i and B_{i+1} for $1 \leq i < c$.

Lemma 5 *Let G be an interval graph with $\omega(G) = k + 1$. Then G contains three maximal cliques of size $k + 1$ that intersect at the same k vertices if and only if there are two consecutive edges S_{i-1} and S_i of cardinality k satisfying $S_{i-1} = S_i$ in every clique-path of G .*

Proof. Assume that G is an interval graph with $\omega(G) = k + 1$ and a clique-path of G contains two consecutive edges S_i and S_{i+1} of cardinality k satisfying $S_i = S_{i+1}$. This means that the maximal cliques of G appearing as bags B_{i-1}, B_i, B_{i+1} share the same k vertices. Since each such maximal clique is distinct, each has a vertex that is not in this intersection, and since $\omega(G) = k + 1$, each has exactly $k + 1$ vertices.

For the other direction, assume that G is an interval graph with $\omega(G) = k + 1$ and that G has three maximal cliques of size $k + 1$ that intersect at the same k vertices. Pick any clique-path of G , and let bags B_i, B_j, B_ℓ be the representatives of these three maximal cliques. They do not necessarily appear consecutively in the chosen clique-path. Let B_i be the one furthest to the left, and let B_j be the one furthest to the right. By the definition of clique-paths, the intersection $B_i \cap B_j$ must be a subset of every edge on the path between B_i and B_j . Since $\omega(G) = k + 1$ and $|B_i \cap B_j| = k$, this means that every edge on the path between B_i and B_j has cardinality exactly k and thus must be equal to $B_i \cap B_j$. Since B_ℓ also appears on the path between B_i and B_j , there are at least two such consecutive edges in the clique-path, and the proof is complete. ■

Theorem 6 *Let G be an interval graph. Then $ms(G) = \omega(G)$ if and only if G has three maximal cliques of size $\omega(G)$ that intersect at the same $\omega(G) - 1$ vertices. Otherwise, $ms(G) = \omega(G) - 1$.*

Proof. If G has three maximal cliques of size $\omega(G)$ that intersect at the same $\omega(G) - 1$ vertices, then the subgraph of G induced by the vertices of these three maximal cliques has mixed search number $\omega(G)$ by Lemma 4. Consequently, $ms(G)$ cannot be smaller than $\omega(G)$. By Proposition 1, $ms(G) \leq pw(G) + 1 = \omega(G)$. Hence we can conclude that $ms(G) = \omega(G)$ in this case.

Assume now that there are no three maximal cliques of size $\omega(G)$ in G that intersect at the same $\omega(G) - 1$ vertices. We give a program to search the graph using $\omega(G) - 1$ searchers. Since $\omega(G) - 1$ searchers are the fewest possible by Proposition 1, the result will follow. Let $P = (B_1, S_1, B_2, S_2, \dots, B_c)$ be any clique-path of G . By Lemma 5, we know that there is no index i with $2 \leq i \leq c - 1$ satisfying $S_{i-1} = S_i$ and $|S_{i-1}| = |S_i| = \omega(G) - 1$. We start searching the graph with bag B_1 . Since $B_1 \not\subseteq B_2$, there is a vertex $x \in B_1 \setminus B_2$, and since $B_1 \cap B_2 = S_1 \neq \emptyset$, there is a vertex $y \in S_1$ with $y \neq x$. Furthermore $|B_1| \leq \omega(G)$. If $|B_1| \leq \omega(G) - 1$, then simply place searchers on all vertices of B_1 . If $|B_1| = \omega(G)$, then place searchers on all vertices of $B_1 \setminus \{y\}$, and then slide the searcher on x to y . In both cases, all vertices of B_1 and all edges between them are cleared, since x has no neighbors in any other bag by the definition of a clique-path. Actually, by the same argument, no vertex of $B_1 \setminus B_2$ has any neighbors in any other bag than in B_1 , hence searchers placed on these vertices can now be safely removed as long as we keep searchers on all vertices of S_1 .

Observe that vertices of $(B_1 \cup \dots \cup B_i) \setminus S_i$ have no neighbors belonging to $(B_{i+1} \cup \dots \cup B_c) \setminus S_i$. Hence we can assume by induction that we have already cleared the subgraph induced by the vertices of B_1, \dots, B_i that do not belong to S_i , and we have searchers placed on all vertices of $S_i = B_i \cap B_{i+1}$. We will now show how to proceed so that the subgraph induced by the vertices of B_1, \dots, B_i, B_{i+1} is cleared and searchers are kept on all vertices of S_{i+1} . If $|B_{i+1}| \leq \omega(G) - 1$ then $|S_i| \leq \omega(G) - 2$ and there are available searchers not guarding the vertices of S_i such that we can place them on vertices of B_{i+1} and all vertices of B_{i+1} will be occupied by searchers. Consequently, all vertices of S_{i+1} are also occupied by searchers and the proof for this case is complete. Assume for the remaining part that $|B_{i+1}| = \omega(G)$. Thus we have enough available searchers to occupy all vertices of B_{i+1} except one, without removing any searcher from S_i . We distinguish between two cases: $S_i = S_{i+1}$ and $S_i \neq S_{i+1}$. If $S_i = S_{i+1}$, then we know by our assumption and Proposition 1 that $|S_i| = |S_{i+1}| \leq \omega(G) - 2$. Hence there are at least two vertices $x, y \in B_{i+1} \setminus S_{i+1}$. We place the available searchers on all unguarded vertices of B_{i+1} except y . Then we can safely slide the searcher on x to y , which clears whole B_{i+1} while guarding all vertices of S_{i+1} by keeping searchers on them. What remains is the case when $S_i \neq S_{i+1}$. Assume first that there is a vertex $x \in S_i \setminus S_{i+1}$. We place searchers on all unguarded vertices of B_{i+1} except one in an arbitrary way. Then we can safely slide the searcher on x to the single vertex of B_i that is not occupied by a searcher, since x has no neighbors in the bags appearing on the other side of S_{i+1} . This will again clear whole B_{i+1} while keeping searchers on all vertices of S_{i+1} . If there is no vertex $x \in S_i \setminus S_{i+1}$ then $S_i \subset S_{i+1}$, which means that $|S_i| \leq \omega(G) - 2$, and there is at least one vertex x in B_{i+1} that does not belong to any other bag since $S_{i+1} \subset B_{i+1}$. Hence we have at least one available searcher that we can place on x without removing any searchers from S_i . We place the available searchers on all vertices of B_{i+1} except one vertex different from x . Now, we can safely slide the searcher on x to the single unguarded vertex of B_{i+1} , which clears whole

B_{i+1} , and we have searchers on all vertices of S_{i+1} , since $x \notin S_{i+1}$. ■

We have thus arrived at the main result of this section.

Theorem 7 *The mixed search number of an interval graph can be computed in linear time.*

Proof. Let G be an interval graph. A clique-path of G can be computed in linear time, and it has at most n maximal cliques [4]. This means that the sum of the sizes of all bags and all edges of the clique-path is $O(n + m)$. Given the clique-path, to find $ms(G)$, by Theorem 6 we need to check for every triple of consecutive maximal cliques B_{i-1}, B_i, B_{i+1} having each $\omega(G)$ vertices whether $B_{i-1} \cap B_i = B_i \cap B_{i+1}$.

We argue that this can be done in overall linear time. First sort all vertices in all bags according to the same order. Since the sum of the sizes of all bags is $O(n + m)$ and the largest value is n , this can be done in $O(n + m)$ time. Then comparing three bags can be done in $O(\omega(G))$ time. We do this only when all three bags have $\omega(G)$ vertices. There are $\omega(G) - 1$ edges in B_{i-1} that do not appear in B_i or B_{i+1} since B_{i-1} has one vertex not appearing in B_i or B_{i+1} . Hence we can let these edges of B_{i-1} “pay for” the comparison between B_{i-1}, B_i, B_{i+1} . Thus each edge of G will pay for at most 3 of comparisons, and therefore we can bound the total time of all comparisons by $O(m)$. ■

We would like to mention that, interestingly, the cases that distinguish the different values of mixed search number of interval graphs are exactly the same cases that distinguish the different values of their edge search number [27]. However, as we will see in Section 5, this is not true for split graphs. Difference between mixed search and edge search seems more substantial for split graphs.

4 The linear-width of interval graphs

Since any induced subgraph of an interval graph is also interval, by Proposition 2 and Theorem 7, we have readily the following result.

Corollary 8 *The linear-width of an interval graph can be computed in linear time.*

In this section, as a structural result, we use the results of the previous section to characterize interval graphs G with $lw(G) = ms(G) - 1$. Observe first that by the definition of linear-width, if G is a caterpillar with at least two edges then $pw(G) = lw(G) = 1$.

Theorem 9 *Let G be an interval graph with at least two edges. Then $lw(G) = ms(G) - 1$ if and only if G is a caterpillar with maximum degree at least 3. Otherwise $lw(G) = ms(G)$.*

Proof. By Proposition 1, $lw(G)$ is either equal to $ms(G)$ or to $ms(G) - 1$.

Let G be a caterpillar with maximum degree at least 3. Then it has three maximal cliques of size 2 that intersect at the same 1 vertex. By Theorem 6, $ms(G) = \omega(G) = 2$. Since G is a caterpillar, $lw(G) = pw(G) = 1$. Therefore $lw(G) = ms(G) - 1$.

For the other direction, let G be an interval graph with $lw(G) \neq ms(G)$. By Proposition 1, $ms(G) = pw(G) + 1 = \omega(G)$ and $lw(G) = pw(G) = \omega(G) - 1$. Since $ms(G) = \omega(G)$ by Theorem 6, G has three cliques of size $\omega(G)$ that intersect at the same $\omega(G) - 1$ vertices. If $\omega(G) = 2$ then G is a caterpillar with maximum degree at least three, and the proof is complete. Assume for contradiction that $\omega(G) > 2$. Then the three cliques mentioned above are of size at least 3. The subgraph of G induced by the union of the vertices of these three cliques has mixed search number $\omega(G)$ by Lemma 4. Since this subgraph has no vertices of degree 1, its linear-width is also $\omega(G)$. Consequently, $lw(G)$ and $ms(G)$ cannot be less than $\omega(G)$, and we conclude that they are equal, which gives a contradiction. Hence $\omega(G)$ cannot be more than 2 if $lw(G) \leq ms(G)$. ■

Since a caterpillar has at most $n - 1$ edges, we can decide in $O(n)$ time whether a given input graph is caterpillar and compute the degrees of all vertices.

5 The mixed search number of split graphs

It is easy to show that the pathwidth of a split graph G is either $\omega(G) - 1$ or $\omega(G)$ [15]. Hence it follows from this and Proposition 1 that $ms(G)$ is equal to one of the following: $\omega(G) - 1$, $\omega(G)$, or $\omega(G) + 1$. In this section we characterize each of these three cases, and show that mixed search number of split graphs can be computed in polynomial time.

Theorem 10 *Let $G = (V, E)$ be a split graph with a split partition (C, I) of its vertices where C is a clique of maximum size. Then $ms(G) = \omega(G) - 1 = |C| - 1$ if and only if one of the following three conditions holds.*

1. *There are 2 vertices $u, v \in C$ such that $N_I(u) \cap N_I(v) = \emptyset$ and $|N_I(v)| \leq 1$ and one of the following is true:*
 - (a) $|N_I(u)| \leq 1$, or
 - (b) *there is an additional vertex $x \in C$ such that $|N_I(u) \cap N_I(x)| \leq 1$.*
2. *There are 3 vertices $x, u, v \in C$ such that $N_I(u) \cap N_I(x) = \emptyset$ and $|N_I(v) \cap N_I(x)| \leq 1$ and one of the following is true:*
 - (a) $|N_I(u)| \leq 1$, or
 - (b) $|N_I(v)| \leq 2$ and $|N_I(u) \cap N_I(v)| \leq 1$.
3. *There are 4 vertices $x, y, u, v \in C$ such that $(N_I(x) \cup N_I(y)) \cap (N_I(u) \cup N_I(v))$ contains at most 2 vertices u_1, v_1 , and $u_1x \notin E$ and $v_1u \notin E$.*

Proof. Assume that $ms(G) = |C| - 1$. Hence there is a monotone mixed search program that is able to clear G with $|C| - 1$ searchers. At some point of this search, all $|C| - 1$ searchers must occupy $|C| - 1$ vertices of C , since only from such a situation, by sliding one of the searchers to the single vertex of C without a searcher, we can clear any clique C and the edges with both endpoints in C . Let us consider the first time when $|C| - 1$ vertices of C are occupied by searchers during this search program. Let v be the vertex that got occupied by a searcher at this point, and let u be the vertex of C without a searcher. Without loss of generality,

we can assume that all vertices of I that are not adjacent to u or v are cleared, as well as all edges between such vertices and C , since we can always do this before placing the last searcher on v . Furthermore, no neighbors of u in I are cleared (otherwise recontamination would be allowed), and at most one neighbor of v in I is cleared (if we placed the last searcher on a neighbor and slid to v). Let v_1 this possible neighbor of v ; consequently v_1 is not adjacent to u . The next step must be to slide a searcher from one of the cleared vertices of C to u , because all vertices of C are pairwise adjacent and u is not cleared, so we would get recontamination otherwise. Let *step i* be this next step. Hence *step i* is either to slide from v to u , or to slide from another vertex $x \neq v$ of C to u . We will show that in either case, the possible ways of completing this search successfully without any more searchers all lead to the conditions of the theorem, proving the *only if* direction. For each condition, we will also explain how to complete the search with $|C| - 1$ searchers, hence the *if* direction will be proved at the same time.

Assume that *step i* of the search is to slide from v to u . This can only be done if v_1 is cleared (or does not exist), hence a searcher must have slid from v_1 to v before this step. Then we know that $|N_I(v)| \leq 1$ and that u and v have no common neighbors. After *step i* , v is the only vertex of C without a searcher.

If *step $i + 1$* is to slide from u to a neighbor u_1 of u in I , then u cannot have more neighbors (otherwise recontamination), and we have Condition 1 (a). The search is complete.

If *step $i + 1$* is to slide (or move) from another vertex $x \neq u$ to a neighbor u_1 of u in I , then x and u can have at most this single neighbor u_1 in common in I . Also x and v can have at most v_1 as a common neighbor in I , but this is already covered since $|N_I(v)| \leq 1$. Hence we have Condition 1 (b). Now any other neighbor that u might have in I can be cleared with the searcher on u_1 , to complete the search.

Assume that *step i* of the search is to slide from a vertex $x \neq v$ of C to u . Then we know that $N_I(v) \cap N_I(x) \subseteq \{v_1\}$ and $N_I(u) \cap N_I(x) = \emptyset$. Hence, if x is adjacent to v_1 , we must have slid a searcher from v_1 to v to place this searcher on v , and v_1 is cleared along with all edges between v_1 and C . After *step i* , x is the only vertex of C without a searcher.

If *step $i + 1$* is to slide from u to a neighbor $u_1 \in I$ of u , then $|N_I(u)| \leq 1$. Hence we have Condition 2 (a). Now we can use the searcher on u_1 to clear all neighbors of v and all edges between these and C , one by one.

If *step $i + 1$* is to move from u to a neighbor of v in I which is not adjacent to u , then $N_I(u) = \emptyset$ and we can use the searcher on u to clear all neighbors of v in I one by one. This case is covered by the previous case, since u could have a neighbor in I which we could have cleared before moving on to the neighbors of v . Hence we have again Condition 2 (a).

If *step $i + 1$* is to slide from v to a neighbor of v in I then $|N_I(v)| \leq 2$. Furthermore, if v has two neighbors in I , then at most one of these can be adjacent to u , because to clear both neighbors of v we must have slid to v from v_1 , and v_1 is not adjacent to u as argued above. Hence we have Condition 2 (b). Now we can use the searcher on the second neighbor of v to search all remaining neighbors of u in I one by one.

If *step $i + 1$* is to move from v to a neighbor $u_1 \in I$ of u with $u_1 \notin N_I(v)$, then $N_I(v)$ does not contain any other vertices than v_1 . Now we can use the searcher on u_1 to clear all

neighbors of u . Note that this situation is just a special case of the previous situation, hence it is covered by Condition 2 (b).

If step $i + 1$ is to move from a vertex $y \neq u, x$ of C to a neighbor $u_1 \in I$ of u , then $N_I(y) \cap N_I(u) \subseteq \{u_1\}$ and $N_I(y) \cap N_I(v) \subseteq \{v_1, u_1\}$. In this case, we must have slid from v_1 to v before placement on v , and after the slide (or move) from y to u_1 , u_1 will be cleared and it can be adjacent to both u and v . After step $i + 1$, the searcher on u_1 can be used to clear all remaining uncleared vertices of $N_I(u) \cup N_I(v)$. We should also remember that $N_I(v) \cap N_I(x) \subseteq \{v_1\}$ and $v_1 u \notin E$. Hence we have Condition 3.

If step $i + 1$ is to move from a vertex $y \neq u, x$ of C to a neighbor of v in I then have the exact same situation as the previous situation (because u_1 can be adjacent to v and hence serve as the assumed neighbor of v), and hence again Condition 3. The search can be completed in the same way.

This covers all possible cases, and completes the proof. ■

Lemma 11 *Let G be a split graph with a split partition (C, I) of its vertices where C is a clique of maximum size. If there are two vertices $u, v \in C$ such that $|N_I(u) \cap N_I(v)| \leq 2$ then $ms(G) \leq \omega(G) = |C|$.*

Proof. Let us describe a search program to clear G with $|C|$ searchers. Place searchers on all vertices of $C \setminus \{u\}$. Use the last available searcher to clear all vertices of $I \setminus N(u)$ and edges between these vertices and $C \setminus \{u\}$. Let x and y be the at most two vertices of I that are both adjacent to u and v . Place the last searcher on x . Now, x and all edges between x and $C \setminus \{u\}$ are cleared. Slide this searcher from x to u . Now all edges between the vertices of C and between x and C are cleared. Slide the searcher on v to the second common neighbor y . Now, all edges between y and C are cleared and no uncleared vertex of $N_I(u)$ is adjacent to v . We can use the searcher on y to search the remaining vertices of $N_I(u)$ one by one. ■

Theorem 12 *Let G be a split graph with a split partition (C, I) of its vertices where C is a clique of maximum size. Then $ms(G) = \omega(G) + 1 = |C| + 1$ if and only if $|N_I(u) \cap N_I(v)| \geq 3$ for every pair of vertices $u, v \in C$.*

Proof. We know that $|C| + 1$ searchers are always enough to clear any split graph, hence if $ms(G) > |C|$ then $ms(G) = |C| + 1$. Consequently Lemma 11 readily states that if $ms(G) = |C| + 1$ then every pair of vertices in C must have at least three common neighbors in I . For the opposite direction we need to show that $|C|$ searchers cannot clear the graph under this condition. Assume on the contrary that every two vertices in C have at least three common neighbors in I , and that there is a monotone mixed search program that is able to clear the whole graph with $|C|$ searchers. As argued previously, at some point of the search searchers must be placed on $|C| - 1$ vertices of C . Consider the first point in time when this happens, and let u be the vertex of C without a searcher. Without loss of generality, we can use the last searcher to clear all vertices of I that are not neighbors of u , and assume that all edges between pairs of vertices in $(C \setminus \{u\}) \cup (I \setminus N(u))$ are cleared. We need to clear the edges

between u and its neighbors, and the edges between $N_I(u)$ and $C \setminus \{u\}$. We know that there can be at most one vertex in $N_I(u)$ that can have a searcher on it, and no other vertices in $N_I(u)$ are cleared, because otherwise they would be subject to recontamination from u . We know that every vertex in $C \setminus \{u\}$ has at least 3 neighbors in $N_I(u)$. If we slide a searcher from a vertex of C to another vertex, recontamination will occur. Assuming without loss of generality that the last searcher is placed on a vertex u_1 of $N_I(u)$, the next step must be to slide the last searcher from u_1 to u . This will clear all edges between u_1 and C , and all edges with both endpoints in C . After this u has still uncleared neighbors, and for each $v \in C \setminus \{u\}$, u and v have at least two common uncleared neighbors v_1 and v_2 . If we slide a searcher from v to v_1 , v will get recontaminated from v_2 , and if we slide a searcher from u to v_1 , u will become recontaminated from v_2 . Hence we cannot complete the search with only $|C|$ searchers, which gives the desired contradiction. ■

Consequently, we are also able to characterize split graphs G for which $ms(G) = \omega(G)$.

Corollary 13 *For a split graph G , $ms(G) = \omega(G)$ if and only if neither any of the three conditions of Theorem 10 nor the condition of Theorem 12 are satisfied.*

We are ready to state the main result of this section.

Theorem 14 *The mixed search number of a split graph can be computed in polynomial time.*

Proof. By the above results we have the following algorithm to compute the mixed search number of a split graph G with split partition (C, I) where C is a maximum clique: For every pair of vertices $u, v \in V$, check if $|N_I(u) \cap N_I(v)| \leq 2$. If no such pair is found, then output $ms(G) = \omega(G) + 1$. This takes $O(n^3)$ time. Whenever such a pair is found, the loop over all pairs can be terminated. If such a pair exists, then the algorithm must check if any of the conditions of Theorem 10 is fulfilled in addition. If so, output $ms(G) = \omega(G) - 1$; if not, output $ms(G) = \omega(G)$. The time required to check each condition of Theorem 10 is clearly polynomial. The first and second conditions require examining every triple of vertices and comparing their neighborhoods, which can be done in total $O(n^4)$ time. The most time consuming condition to check is the last condition, which is checked only if all other conditions above fail. In a straight forward way, by examining every quadruple of vertices and comparing their neighborhoods, this condition can be checked in $O(n^5)$ time, which dominates the total running time of the algorithm. ■

6 The linear-width of split graphs

Since any induced subgraph of a split graph is also split, by Proposition 2 and Theorem 14, we have the following result.

Corollary 15 *The linear-width of a split graph can be computed in polynomial time.*

In this section, like Theorem 9, we a structural result on the linear-width of split graphs.

Lemma 16 * *Let G be a split graph.*

1. *If $lw(G) \neq ms(G)$ then $lw(G) = \omega(G) - 1$ and $ms(G) = \omega(G)$.*
2. *If G' is the graph obtained by removing all vertices of degree 1 from G , and $\omega(G) \geq 3$, then $lw(G) = lw(G')$.*

Proof. Split graphs G having $ms(G) = \omega(G) + 1$ are characterized by Theorem 12. Definitely, if G satisfies the condition of Theorem 12 then G has an induced subgraph G' with no vertices of degree 1 such that $ms(G') = ms(G) = \omega(G) + 1$ by the proof of Theorem 12. Since G' has no vertices of degree 1, $lw(G') = lw(G)$. And since G' is a subgraph of G , $lw(G) = lw(G') = ms(G') = ms(G) = \omega(G) + 1$. Hence, Case 2 mentioned above can never occur, and the only possibility is Case 1.

2. Since G is split and $\omega(G) \geq 3$, G' has no vertices of degree 1. Consequently, by Proposition 2, $lw(G) = ms(G') = lw(G')$. ■

7 Concluding remarks

We have shown that mixed search number of interval graphs and split graphs can be computed in linear and polynomial time, respectively. Regarding the algorithm described in the proof of Theorem 14, probably a better running time is possible to achieve by carefully organizing the computation. An interesting open question is whether there exists a different algorithm for the mixed search number of split graphs with better running time.

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] D. BIENSTOCK, *Graph searching, path-width, tree-width and related problems (a survey)*, in Reliability of computer and communication networks (New Brunswick, NJ, 1989), vol. 5 of DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Amer. Math. Soc., Providence, RI, 1991, pp. 33–49.
- [3] D. BIENSTOCK AND P. SEYMOUR, *Monotonicity in graph searching*, J. Algorithms, 12 (1991), pp. 239–245.
- [4] 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.
- [5] J. COHEN, F. V. FOMIN, P. HEGGERNES, D. KRATSCH, AND G. KUCHEROV, *Optimal linear arrangement of interval graphs.*, in Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006), vol. 4162 of Lecture Notes in Computer Science, Springer, 2006, pp. 267–279.

- [6] J. DÍAZ, J. PETIT, AND M. SERNA, *A survey of graph layout problems*, ACM Computing Surveys, 34 (2002), pp. 313–356.
- [7] J. A. ELLIS, I. H. SUDBOROUGH, AND J. S. TURNER, *The vertex separation and search number of a graph*, Inform. and Comput., 113 (1994), pp. 50–79.
- [8] F. V. FOMIN, *A generalization of the graph bandwidth*, Vestnik St. Petersburg Univ. Math., 34 (2001), pp. 15–19 (2002).
- [9] F. V. FOMIN AND P. A. GOLOVACH, *Graph searching and interval completion*, SIAM J. Discrete Math., 13 (2000), pp. 454–464 (electronic).
- [10] F. V. FOMIN, P. A. GOLOVACH, AND N. N. PETROV, *Search problems on 1-skeletons of regular polyhedrons*, Int. J. Math. Game Theory Algebra, 7 (1998), pp. 101–111.
- [11] F. V. FOMIN AND D. M. THILIKOS, *On the monotonicity of games generated by symmetric submodular functions*, Discrete Appl. Math., 131 (2003), pp. 323–335.
- [12] F. V. FOMIN AND D. M. THILIKOS, *A 3-approximation for the pathwidth of halin graphs.*, J. Discrete Algorithms, 4 (2006), pp. 499–510.
- [13] P. C. GILMORE AND A. J. HOFFMAN, *A characterization of comparability graphs and of interval graphs*, Canad. J. Math., 16 (1964), pp. 539–548.
- [14] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs*, Second edition, Annals of Discrete Mathematics 57. Elsevier, 2004.
- [15] J. GUSTEDT, *On the pathwidth of chordal graphs*, Disc. Appl. Math., 45 (1993), pp. 233–248.
- [16] L. M. KIROUSIS AND C. H. PAPADIMITRIOU, *Interval graphs and searching*, Discrete Math., 55 (1985), pp. 181–184.
- [17] ———, *Searching and pebbling*, Theoret. Comput. Sci., 47 (1986), pp. 205–218.
- [18] D. J. KLEITMAN AND R. V. VOHRA, *Computing the bandwidth of interval graphs*, SIAM J. Disc. Math., 3 (1990), pp. 373–375.
- [19] T. KLOKS, J. KRATOCHVÍL, AND H. MÜLLER, *Computing the branchwidth of interval graphs*, Disc. Appl. Math., 145 (2005), pp. 266–275.
- [20] T. KLOKS, D. KRATSCH, Y. L. BORGNE, AND H. MÜLLER, *Bandwidth of split and circular permutation graphs*, in Proceedings of the 26th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2000), vol. 1928 of Lecture Notes in Computer Science, Springer, 2000, pp. 243–254.
- [21] A. S. LAPAUGH, *Recontamination does not help to search a graph*, J. Assoc. Comput. Mach., 40 (1993), pp. 224–245.

- [22] N. MEGIDDO, S. L. HAKIMI, M. R. GAREY, D. S. JOHNSON, AND C. H. PAPADIMITRIOU, *The complexity of searching a graph*, J. Assoc. Comput. Mach., 35 (1988), pp. 18–44.
- [23] T. D. PARSONS, *Pursuit-evasion in a graph*, in Theory and applications of graphs (Proc. Internat. Conf., Western Mich. Univ., Kalamazoo, Mich., 1976), Springer, Berlin, 1978, pp. 426–441. Lecture Notes in Math., Vol. 642.
- [24] ———, *The search number of a connected graph*, in Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing (Florida Atlantic Univ., Boca Raton, Fla., 1978), Congress. Numer., XXI, Winnipeg, Man., 1978, Utilitas Math., pp. 549–554.
- [25] C. PAUL AND J. A. TELLE, *New tools and simpler algorithms for branchwidth.*, in Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005), 2005, pp. 379–390.
- [26] S.-L. PENG AND C.-K. CHEN, *On the interval completion of chordal graphs*, Discrete Appl. Math., 154 (2006), pp. 1003–1010.
- [27] 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. Treewidth.
- [28] N. N. PETROV, *A problem of pursuit in the absence of information on the pursued*, Differential'nye Uravneniya, 18 (1982), pp. 1345–1352, 1468.
- [29] A. TAKAHASHI, S. UENO, AND Y. KAJITANI, *Mixed searching and proper-path-width*, Theoret. Comput. Sci., 137 (1995), pp. 253–268.
- [30] D. M. THILIKOS, *Algorithms and obstructions for linear-width and related search parameters*, Discrete Appl. Math., 105 (2000), pp. 239–271.
- [31] R. THOMAS, *Tree-decompositions of graphs (lecture notes)*. School of Mathematics. Georgia Institute of Technology, Atlanta, Georgia 30332, USA, 1996.