REPORTS
IN
INFORMATICS

ISSN 0333-3590

Mixed search number of permutation graphs

Pinar Heggernes **Rodica Mihai**

REPORT NO 365 **January 2008**

Department of Informatics

UNIVERSITY OF BERGEN

Bergen, Norway

This report has URL
http://www.ii.uib.no/publikasjoner/texrap/pdf/2008-365.pdf

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

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 of permutation graphs

Pinar Heggernes* Rodica Mihai*

Abstract

Search games in graphs have attracted significant attention in recent years, and they have applications in securing computer networks against viruses and intruders. Since graph searching is an NP-hard problem, polynomial-time algorithms have been given for solving it on various graph classes. Most of these algorithms concern computing the node search number of a graph, and only few such algorithms are known for computing the mixed search or edge search numbers of specific graph classes. In this paper we show that the mixed search number of permutation graphs can be computed in linear time, and we describe an algorithm for this purpose. In addition, we give a complete characterization of the edge search number of complete bipartite graphs.

1 Introduction

The graph searching problem concerns a team of searchers who are trying to capture a fugitive moving along the edges of the graph. The fugitive is assumed to be very fast and invisible, and he knows the search strategy of the searchers. The minimum number of searchers that can guarantee the capture of the fugitive under this worst case scenario for the searchers is the search number of the graph, and the problem is to compute this number. The study of the graph searching problem started in 1970s when it was independently introduced by Parsons [25] and Petrov [28], and since that time it has been studied extensively [3, 2, 21, 22, 18, 26]. It fits into the broader class of pursuit-evasion, search, and rendezvous problems on which a large number of results have appeared [1].

In a computer network setting, the graph searching problem serves as a mathematical model for protecting networks against viruses and other unwanted agents, like spyware or eavesdroppers [2, 13]. A practical example is the problem of finding a successful strategy for a group of collaborating software programs that are designed to clean the network from a virus [11].

In the above mentioned original version of graph searching by Parsons and Petrov, later called edge searching [19], a search step consists of placing a searcher on a vertex or removing a searcher from a vertex or sliding a searcher along an edge. An edge is cleared by sliding a searcher from one of its endpoints to the other endpoint. Kirousis and Papadimitriou [19] introduced a variant of graph searching called node searching. In this version an edge is cleared if both its endpoints contain searchers. A new version of the graph searching was introduced by Bienstock and Seymour in [3]. This version, called mixed searching, combines features of both edge searching and node searching. An edge is cleared either by sliding or by placing searchers at each endpoint. In the mixed searching game, a contaminated edge of the graph is cleared if either both its two endpoints

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

contain searchers or a searcher is slided along it. The allowable moves are placing a searcher on a vertex, removing a searcher from a vertex and sliding a searcher along an edge.

The minimum number of the searchers sufficient to perform searching and ensure the capture of the fugitive for each of the models are respectively the edge, node, and mixed search numbers, and computations of these are all NP-hard [3, 22, 18]. The node search number of a graph is known to be equal to its pathwidth plus one; the mixed search number of a graph is equal to its proper pathwidth [31].

Polynomial-time algorithms are known for computing the node search number of trees [27, 29], interval graphs [7], cographs [6], k-starlike graphs for fixed k [26], d-trapezoid graphs [5], block graphs [9], split graphs [17], circular-arc graphs [30], and permutation graphs [4, 23]. However, only for a few of these graph classes polynomial-time algorithms are known for computing mixed search or edge search numbers. Edge search number of trees [22, 27], interval graphs and split graphs [26, 15] can be computed in polynomial time. For computing the mixed search number, polynomial-time algorithms exist so far only for interval graphs and split graphs [12].

In this paper we show that the mixed search number of permutation graphs can be computed in linear time, thereby resolving the computational complexity of this problem on this graph class. Permutation graphs are a well-studied graph class with significant theoretical importance [16, 8]. In addition, we show how to compute the edge search number for a subclass of permutation graphs, namely complete bipartite graphs. In fact we give a complete characterization of both edge and mixed search numbers on complete bipartite graphs.

2 Preliminaries

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 subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$.

A *path* is a sequence $v_1, v_2, ..., v_p$ 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 $v_1, v_2, ..., v_p$ 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 vertex set $S \subset V$ is a *separator* if $G[V \setminus S]$ is disconnected. Given two vertices u and v, S is a u, v-*separator* if u and v belong to different connected components of $G[V \setminus S]$, and S is then said to *separate* u and v. Two separators S and T are said to be *crossing* if S is a u, v-separator for a pair of vertices $u, v \in T$, in which case T is an x, y-separator for a pair of vertices $x, y \in S$ [20, 24]. A u, v-separator S is *minimal* if no proper subset of S separates u and v. In general, S is a *minimal separator* of G if there exist two vertices u and v in G such that S is a minimal u, v-separator. It can be easily verified that S is a minimal separator if and only if $G[V \setminus S]$ has two distinct connected components C_1 and C_2 such that $N_G(C_1) = N_G(C_2) = S$. In this case, C_1 and C_2 are called *full components*.

2.1 Chordal graphs, interval graphs, and pathwidth

Permutation graphs and complete bipartite graphs will be introduced in the sections in which they are studied. In this subsection we mention the graph classes and graph parameters that are central

in graph searching.

A graph is *chordal* if every cycle of length at least 4 has a chord. A *triangulation* of a graph G is a chordal graph H on the same vertex set as G such that G is a subgraph of H. If there is no proper subgraph of H that is a triangulation of G then H is said to be a *minimal triangulation* of G. The following central characterization of minimal triangulations is useful for understanding our results on permutation graphs. A triangulation of G is minimal if and only if it is obtained by adding edges to make into cliques a maximal set of non-crossing minimal separators of G [24]. A set C of vertices of G is a *potential maximal clique* if there is a minimal triangulation of G in which C is a maximal clique.

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. A path decomposition of width $pw(G)$ is called an *optimal* path decomposition of G.

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. An important characterization of interval graphs is that 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 [14]. Such an optimal path decomposition is 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 [7]. For an arbitrary graph G, every path decomposition of G corresponds to an interval graph obtained by adding edges to G until each bag of the path decomposition is a clique. The mentioned path decomposition is then a clique path of this interval graph.

2.2 Search games

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 winning mixed search program using $ms(G)$ steps (analogously, a winning edge search program using $es(G)$ steps) is called *optimal*. The following result is central and gives the relation between the three graph searching parameters.

Lemma 1 ([31]) *Let G be an arbitrary graph.*

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

Note that, although the node search number of a graph is known, it might be difficult to decide its mixed search number or edge search number. Hence although $pw(G)$ of a graph G can be computed easily, it might be difficult to decide whether $ms(G) = pw(G)$ or $ms(G) = pw(G)+1$.

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 {edge, mixed, node} search number k there exists a winning monotone {edge, mixed, node} search program using k searchers. Thus in this paper we consider only monotone search programs.

3 Mixed search number of permutation graphs

Let π be a permutation of $\{1, ..., n\}$. We define $G(\pi)$ to be the graph with vertex set $\{1, ..., n\}$ and edge set $\{ij \mid (i - j) \cdot (\pi^{-1}(i) - \pi^{-1}(j)) < 0\}$. Hence, two vertices i, j of $G(\pi)$ are adjacent if and only if the permutation π changes their natural order. An undirected graph G is called a *permutation graph* if there exists a permutation π such that G is isomorphic to $G(\pi)$.

In this section we give a linear-time algorithm to compute the mixed search number of permutation graphs. Permutation graphs are a well studied graph class with subject to many theoretical results, and they have many characterizations [16]. If G is a permutation graph then all minimal triangulations of G are interval graphs [4]. In addition, permutation graphs have a linear number of minimal separators [23]. Pathwidth of permutation graphs, and hence their node search number, can be computed in linear time [4, 23]. No polynomial-time algorithm has been known for computing their mixed search number.

We start by relating mixed search number to proper pathwidth, and then giving a new general result, before we move to permutation graphs. A path decomposition is called *proper* if no three bags of the same size s all intersect in the same $s - 1$ vertices. The *proper pathwidth* of a graph G, denoted by $ppw(G)$ is the minimum width over all proper path decompositions of G.

Theorem 2 ([31]) *For any graph G, $ms(G) = ppw(G)$.*

Thus computing the mixed search number and the proper pathwidth are equivalent problems. This, in combination with the following result, is the main tool that we use to compute the mixed search number of permutation graphs.

Theorem 3 ([12]) *For an interval graph G, $ms(G) = pw(G)$ if and only if no three maximum cliques intersect in $pw(G)$ vertices.*

We define a *good path decomposition* to be an optimal path decomposition that does not contain three consecutive bags intersecting in the same $pw(G)$ vertices. We now add the following new result for general graphs that strengthens Theorem 2.

Theorem 4 *For any graph G, $ms(G) = pw(G)$ if and only if G has a good path decomposition. (Otherwise $ms(G) = pw(G) + 1$.)*

Proof. First we show that in any optimal path decomposition P, if there are three bags of maximum size intersecting in the same $pw(G)$ vertices then there are three consecutive bags intersecting in the same $pw(G)$ vertices. Let B_i, B_j, B_k be three bags from left to right (not necessarily consecutive) in P such that $|S = B_i \cap B_j \cap B_k| = pw(G)$. Then by the definition of a path decomposition, S is a subset of every bag of P between B_i and B_k. Since no bag is a subset of another bag, it means that all bags between B_i and B_k must be of maximum size and contain S. Hence any three consecutive bags between B_i and B_k are of size $pw(G) + 1$ and contain the same $pw(G)$ vertices.

If $ms(G) = pw(G)$ then by Theorem 2, $pw(G) = ppw(G)$, and by the above argument there exists a good path decomposition of G.

If G has a good path decomposition P then let H be the interval graph obtained by making each bag of P into a clique by adding edges. Since P is an optimal path decomposition of G, $pw(H) = pw(G)$. Since G is a subgraph of H, $ms(G) \leq ms(H)$. No three bags of P of maximum size overlap in the same $pw(G)$ vertices, and since there is a one-to-one correspondence between the bags of P and the maximal cliques of H, no three maximum cliques of H overlap on the same $pw(G)$ vertices. Hence $ms(H) = pw(H)$ by Theorem 3. Combining all of the above, we obtain that $ms(G) \leq pw(G)$, and the result follows from Lemma 1. □

With this general result, we are now ready to move to permutation graphs, and the computation of their mixed search number.

Lemma 5 *Let G be a permutation graph. If $ms(G) = pw(G)$ then G has a good path decomposition that corresponds to a minimal triangulation of G.*

Proof. Since $ms(G) = pw(G)$, by Theorem 4 G has a good path decomposition P. Assume that the interval graph H that has P as a clique path is not a minimal triangulation of G. Then H has a chordal subgraph H' which is a minimal triangulation of G. Since all minimal triangulations of permutation graphs are interval graphs, H' is an interval graph and has a clique path P' which is a path decomposition of G. We argue that P' is a good path decomposition of G. Observe that the size of the largest bag in P' cannot be larger than the size of the largest bag in P since H' is a subgraph of H, and thus P' is an optimal path decomposition. Removal of edges from H might create three new bags of size s that intersect at the same $s - 1$ vertices. However, if this happens then s cannot be equal to $pw(G) + 1$ because if the removal of an edge splits a maximal clique

into two new maximal cliques, then the new maximal cliques will be of size at least 1 less than the size of the old maximal clique. Hence, P' is a good path decomposition of G that corresponds to the minimal triangulation H', and the proof is complete. □

In the remaining of this section, let $G = G(\pi)$ be a permutation graph for a permutation π of $\{1, ..., n\}$. A *permutation diagram* of G is obtained in the following way. Take two copies of the real line between 0.5 and $n + 0.5$; place one of them below the other; put consecutive labels from 1 to n on the integer points of the above one; put consecutive labels from $\pi(1)$ to $\pi(n)$ on the integer points of the other; draw lines between a point on the upper line and a point on the lower line if and only if the two points have the same labels. Each line $(i, \pi^{-1}(i))$ will be called the *line of vertex* i. It is easy to see that two vertices i and j are adjacent in G if and only if their lines intersect in the permutation diagram for G.

A *scanline* of G is a pair (a, e) where $a, e \in \{0.5, 1.5, ..., n + 0.5\}$. We also define the following two scanlines $s_0 = (0.5, 0.5)$ and $s_e^n = (n+0.5, n+0.5)$. Each scanline s_i is associated with a set of vertices S_i of G such that S_i consists of exactly those vertices whose lines cross s_i. For each scanline s_i, the corresponding vertex set S_i is a separator of G [4]. A *special scanline* is a scanline s_i such that S_i is a minimal separator of G and s_i is between two full components of $G[V \setminus S_i]$ in the permutation diagram [23].

Meister defined the potential maximal clique graph of a permutation graph, and used this to compute the pathwidth of permutation graphs in linear time [23]. We will heavily rely on this algorithm. The *potential maximal clique graph* of a permutation graph $G(\pi)$ is a directed graph $\mathcal{PC}(\pi)$ defined as follows: $\mathcal{PC}(\pi)$ has a vertex for every special scanline of G, and there is an arc from vertex s_i to vertex s_j if and only if the corresponding special scanline s_i is on the left side of the special scanline s_j and there is no other special scanline strictly between them (non-intersecting with any of them). Hence the set of vertices of $\mathcal{PC}(\pi)$ correspond exactly to the set of minimal separators of $G(\pi)$, and arcs go between non-crossing minimal separators. For each arch $s_i s_j$ of $\mathcal{PC}(\pi)$ we define C_{ij} to be the set of vertices whose lines cross s_i or s_j and vertices whose lines are between s_i and s_j in the permutation diagram of G. For each arc $s_i s_j$ in $\mathcal{PC}(\pi)$, C_{ij} is a potential maximal clique of G, and each potential maximal clique of G corresponds to an edge in $\mathcal{PC}(\pi)$. Furthermore, the number of vertices of $\mathcal{PC}(\pi)$ is $O(n + m)$, $\mathcal{PC}(\pi)$ is acyclic and can be generated in linear time [23]. A *source-sink path* in $\mathcal{PC}(\pi)$ is any directed path from s_0 to s_e^n.

Theorem 6 ([23]) *Let $G = G(\pi)$ be a permutation graph. An interval graph H is a minimal triangulation of G if and only if there is a source-sink path $s_0, ..., s_{k-1}, s_e^n$ in $\mathcal{PC}(\pi)$ such that $C_{01}, ..., C_{k-1,k}$ is a clique path of H.*

We will call a source-sink path $s_0, ..., s_{k-1}, s_e^n$ a *good path* if $|C_{i-1,i}| \leq pw(G) + 1$, and $S_{i-1} \neq S_i$ whenever $|S_{i-1}| = |S_i| = pw(G)$, for all $i \in \{1, 2, \ldots, k\}$. We can now give the following main structural result.

Theorem 7 *Let $G = G(\pi)$ be a permutation graph. Then $ms(G) = pw(G)$ if and only if there exists a good path in $\mathcal{PC}(\pi)$. (Otherwise $ms(G) = pw(G) + 1$.)*

Proof. If there exists a good path in $\mathcal{PC}(\pi)$ then by Theorem 6 there exists a good path decomposition of G. Hence by Theorem 4, $ms(G) = pw(G)$. If $ms(G) = pw(G)$ then by Lemma 5 there exists a good path decomposition P of G that corresponds to a minimal triangulation. Observe that the condition that no three consecutive bags of P intersect in the same $pw(G)$ vertices is equivalent to the condition that no two consecutive minimal separators of size $pw(G)$ of P are

equal. Hence by Theorem 6, there exists a good path in $\mathcal{PC}(\pi)$ corresponding to this minimal triangulation. □

Thus we will search for good paths in $\mathcal{PC}(\pi)$ to decide whether $ms(G) = pw(G)$ or $ms(G) = pw(G) + 1$. The algorithm is described in the proof of the following theorem.

Theorem 8 *The mixed search number of a permutation graph can be computed in linear time.*

Proof. We describe such an algorithm. By the results of [23], $\mathcal{PC}(\pi)$ can be computed, a topological search on it can be performed, and $pw(G)$ can be computed in linear time. In fact, the algorithm of Meister [23] computes the size of each minimal separator corresponding to a vertex and each potential maximal clique corresponding to an arc of $\mathcal{PC}(\pi)$ within the linear time bound, and the data structure on which this algorithm is based allows checking the equivalence of two minimal separators in constant time. During the construction of $\mathcal{PC}(\pi)$, minimal separators that are equal are detected and only one copy of the list of vertices in these minimal separators is kept, whereas all minimal separators that contain exactly these vertices are set to point to the same location, all within the linear time bound. Hence to check whether two vertices of $\mathcal{PC}(\pi)$ correspond to two minimal separators that are equal, can be done in constant time by comparing the pointers, after the preprocessing during the construction of $\mathcal{PC}(\pi)$.

We now describe our algorithm, based on the above. First we run the algorithm of [23] to construct $\mathcal{PC}(\pi)$ as described above and to compute $pw(G)$. After this, we delete all arcs corresponding to potential maximal cliques of size larger than $pw(G)+1$, and all vertices corresponding to minimal separators of size larger than $pw(G)$ from $\mathcal{PC}(\pi)$, since these can never be involved in paths corresponding to optimal path decompositions. After this we traverse the graph in a topological order, and delete every arc s_is_j such that $|S_i| = |S_j| = pw(G)$ and $S_i = S_j$. Such arcs can never be part of a good path, and can thus be deleted safely. Let us call $\mathcal{PC}'(\pi)$ the potential maximal cliques graph that we obtain after the described deletions from $\mathcal{PC}(\pi)$. We claim that there is a good path in $\mathcal{PC}(\pi)$ if and only if there is a source-sink path in $\mathcal{PC}'(\pi)$. Clearly, if there is a source-sink path P in $\mathcal{PC}'(\pi)$ then P is a good path in $\mathcal{PC}(\pi)$ since P is a path in $\mathcal{PC}(\pi)$ that does not contain any of the forbidden substructures of a good path. If there is a good path P in $\mathcal{PC}(\pi)$ then this path survives all the deletions described above since it does not contain any of the deleted substructures. Hence P is a source-sink path in $\mathcal{PC}'(\pi)$.

The algorithm, after the deletions, is to simply search for a source-sink path in $\mathcal{PC}'(\pi)$. The algorithm returns such a path if it is found and outputs $ms(G) = pw(G)$. If no such path is found, then the algorithm returns $ms(G) = pw(G) + 1$. The correctness of the algorithm follows from the proof of the claim in the previous paragraph and Theorem 7.

For the running time, after computing $\mathcal{PC}(\pi)$ and $pw(G)$ in linear time with Meister's algorithm [23], deletion of arcs and vertices that correspond to too big potential maximal cliques and minimal separators can be done in linear time, too, since we only check sizes. For each arc s_is_j, whether $|S_i| = |S_j| = pw(G)$ can be checked in constant time by the same arguments, and checking whether $S_i = S_j$ can be done in constant time, too, as explained in the first paragraph, comparing the pointers from s_i and s_j. Looking for source-sink paths in $\mathcal{PC}'(\pi)$ takes also clearly linear-time, since this is just simple graph traversal. Hence the total running time is $O(n + m)$. □

4 Edge search number of complete bipartite graphs

A *bipartite graph* is a graph whose vertex set can be partitioned into two independent sets. We denote such a graph by $G = (A, B, E)$ where $A \cup B$ is the vertex set of G, and A and B are independent sets. A bipartite graph $G = (A, B, E)$ is a *complete bipartite graph* if every vertex of A is adjacent to every vertex of B. Such a graph is denoted by $K_{a,b}$, where $a = |A|$ and $b = |b|$.

It is known that $pw(K_{a,b}) = \min\{a, b\}$ [6], hence the node search number of complete bipartite graphs is completely characterized. By the results of the previous section, their mixed search number can be computed in linear time, since complete bipartite graphs are a subset of permutation graphs. Here, we give a complete characterization of their edge search number, hence completing the knowledge of searching in complete bipartite graphs.

Lemma 9 *If* $\min\{a, b\} \geq 3$ *then* $es(K_{a,b}) = \min\{a, b\} + 2$.

Proof. Let A and B be the two independent sets of $G = K_{a,b}$ with $|A| = a$ and $|B| = b$. Assume without loss of generality that $a \leq b$. By the result mentioned above, $pw(G) = a$. Thus by Lemma 1 we have $a \leq es(G) \leq a + 2$. We will show that $es(G) = a + 2$. Hence we have to show that $es(G) \geq a + 2$.

For this lower bound, assume for a contradiction that there is an edge search program that clears the graph with $a + 1$ searchers without recontamination. If all vertices of A are occupied by searchers initially, then one searcher is left to clear all edges, and since all vertices of B are uncleared and without searchers, there is no way to continue without recontamination. Hence initially at least one vertex v of B is occupied with a searcher. This searcher can only be removed after the clearance of all edges incident to B but one, so the first move cannot be to slide the searcher on v. The same is true for the vertices of A as well, so at most $a - 1$ vertices of A are occupied with searchers initially, and the first move must be to use the last searcher to clear an edge whose both endpoints are occupied by searchers. When all edges incident to v are cleared except one, the searcher on v can be slided to the other endpoint of this remaining edge, and all edges incident to v will be cleared. This can be done only if at most one vertex of A was without searcher. After this, all vertices of A are occupied with searchers, we have one idle searcher, and at least two vertices of B remain without searchers. Hence each vertex of A has at least two uncleared edges incident to it. We can slide the idle searcher from a vertex u of A to a vertex of B different from v and leave it there, and if $b = 3$ we can even slide the searcher on u to the last vertex of B and leave it there, without recontamination. These are the only possible allowed moves at this stage. But after that still we are left with at least two vertices from each side that each have at least two uncleared edges incident to it, and we have no idle searcher available to slide between them. Since none of the searchers can be moved without recontamination, we obtain the desired contradiction, and conclude that the search cannot be completed with at most $a + 1$ searchers. □

For the cases not covered by the above lemma, it can be easily verified that $es(K_{1,1}) = es(K_{1,2}) = 1$, $es(K_{1,b}) = 2$ for $b \geq 3$, $es(K_{2,2}) = 2$, and $es(K_{2,b}) = 3$ for $b \geq 3$.

Hence we can conclude that if a complete bipartite graph is given as a pair of integers, representing the sizes of the two independent sets, its edge search number can be computed in constant time. This is also true for its node search number by the results of [6], and its mixed search number by our next result, which we include for completeness.

Lemma 10 *If* $\max\{a, b\} \geq 3$ *then* $ms(K_{a,b}) = \min\{a, b\} + 1$.

Proof. Let A and B be the two independent sets of $G = K_{a,b}$ with $|A| = a$ and $|B| = b$. Assume without loss of generality that $a \leq b$. By the result mentioned above, $pw(G) = a$. Thus by Lemma 1 we have $a \leq ms(G) \leq a + 1$. We will show that $ms(G) = a + 1$. Hence we have to show that $ms(G) \geq a + 1$.

Assume for a contradiction that there is a mixed search program to clear the graph using a searchers without allowing recontamination. If initially all searchers are placed on the vertices of A, each of the vertices is adjacent to at least two uncleared edges. Therefore none of the searchers can be removed or slided without allowing recontamination. Hence initially at least one searcher is placed on a vertex v of B so that at most $a - 1$ are placed on the vertices of A. Let $u \in A$ be a vertex without a searcher. All the edges between v and the vertices of $A \setminus \{u\}$ are thus cleared. Note that there are at least two uncleared vertices in B without searchers. (If $a = b$ and all searchers are placed on B the same argument above on A applies on B.) Hence the next step of this search program cannot be to move a searcher from a vertex of A to a vertex of B, because this would recontaminate that vertex of A since every vertex of A is adjacent to at least two uncleared vertices in B. Hence the next move is to move the searcher on v. If it is moved to another vertex of B, v will be recontaminated because of u, which is still uncleared. So to avoid recontamination, the search has to continue by sliding the searcher on v to u along the edge vu. At this moment v and all edges incident to it are cleared, all vertices of A are occupied by searchers, but each vertex of A is adjacent to at least two uncleared edges. Therefore no searcher can be slided or removed without allowing recontamination, which contradicts the existence of the assumed search program. Thus, $ms(G) \geq a + 1$. □

For the cases not covered by this lemma, it can be easily verified that $ms(K_{1,1}) = ms(K_{1,2}) = 1$ and $ms(K_{2,2}) = 2$.

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. 1

[2] 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. 1

[3] D. BIENSTOCK AND P. SEYMOUR, *Monotonicity in graph searching*, J. Algorithms, 12 (1991), pp. 239–245. 1, 2.2

[4] H. L. BODLAENDER, T. KLOKS, AND D. KRATSCH, *Treewidth and pathwidth of permutation graphs*, SIAM J. Disc. Math., 8 (1995), pp. 606–616. 1, 3, 3

[5] 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). 1

[6] 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. 1, 4, 4

[7] 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. 1, 2.1

[8] A. BRANDSTÄDT, V. B. LE, AND J. P. SPINRAD, *Graph classes: a survey*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999. 1

[9] H. CHOU, M. KO, C. HO, AND G. CHEN, *Node-searching problem on block graphs*, Disc. Appl. Math., 156 (2008), pp. 55–75. 1

[10] D. G. CORNEIL, H. LERCHS, AND L. STEWART BURLINGHAM, *Complement reducible graphs*, Annals Discrete Math., 1 (1981), pp. 145–162.

[11] P. FLOCCHINI, M. J. HUANG, AND F. L. LUCCIO, *Contiguous search in the hypercube for capturing an intruder*, Proceedings of IPDPS 2005, IEEE Computer Society, 2005. 1

[12] 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. 1, 3

[13] M. FRANKLIN, Z. GALIL, AND M. YUNG, *Eavesdropping games: a graph-theoretic approach to privacy in distributed systems*, J. ACM, 47 (2000), pp. 225–243. 1

[14] P. C. GILMORE AND A. J. HOFFMAN, *A characterization of comparability graphs and of interval graphs*, Canad. J. Math., 16 (1964), pp. 539–548. 2.1

[15] 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. 1

[16] M. C. GOLUMBIC, *Algorithmic Graph Theory and Perfect Graphs (Annals of Discrete Mathematics, Vol 57)*, North-Holland, 2004. 1, 3

[17] J. GUSTEDT, *On the pathwidth of chordal graphs*, Disc. Appl. Math., 45 (1993), pp. 233–248. 1

[18] L. M. KIROUSIS AND C. H. PAPADIMITRIOU, *Interval graphs and searching*, Disc. Math., 55 (1985), pp. 181–184. 1

[19] M. KIROUSIS AND C. H. PAPADIMITRIOU, *Searching and pebbling*, Theor. Comput. Sci., 47 (1986), pp. 205–218. 1

[20] T. KLOKS, D. KRATSCH, AND J. SPINRAD, *On treewidth and minimum fill-in of asteroidal triple-free graphs*, Theor. Comp. Sc., 175 (1997), pp. 309–335. 2

[21] A. S. LAPAUGH, *Recontamination does not help to search a graph*, J. ACM, 40 (1993), pp. 224–245. 1, 2.2

[22] 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. 1

[23] 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. 1, 3, 3, 6, 3

[24] A. PARRA AND P. SCHEFFLER, *Characterizations and algorithmic applications of chordal graph embeddings*, Disc. Appl. Math. 79 (1997), pp. 171–188. 2, 2.1

[25] T. PARSONS, *Pursuit-evasion in a graph*, in Theory and Applications of Graphs, Springer-Verlag, 1976. 1

[26] 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. 1

[27] 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. 1

[28] N. N. PETROV, *A problem of pursuit in the absence of information on the pursued*, Differentsialnye Uravneniya, 18 (1982), pp. 1345–1352, 1468. 1

[29] 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. 1

[30] K. SUCHAN AND I. TODINCA, *Pathwidth of circular-arc graphs*, Proceedings of WG 2007, Lecture Notes in Computer Science 4769, 2007, pp. 258–269. 1

[31] A. TAKAHASHI, S. UENO, AND Y. KAJITANI, *Mixed searching and proper-path-width*, Theor. Comput. Sci., 137 (1995), pp. 253–268. 1, 1, 2