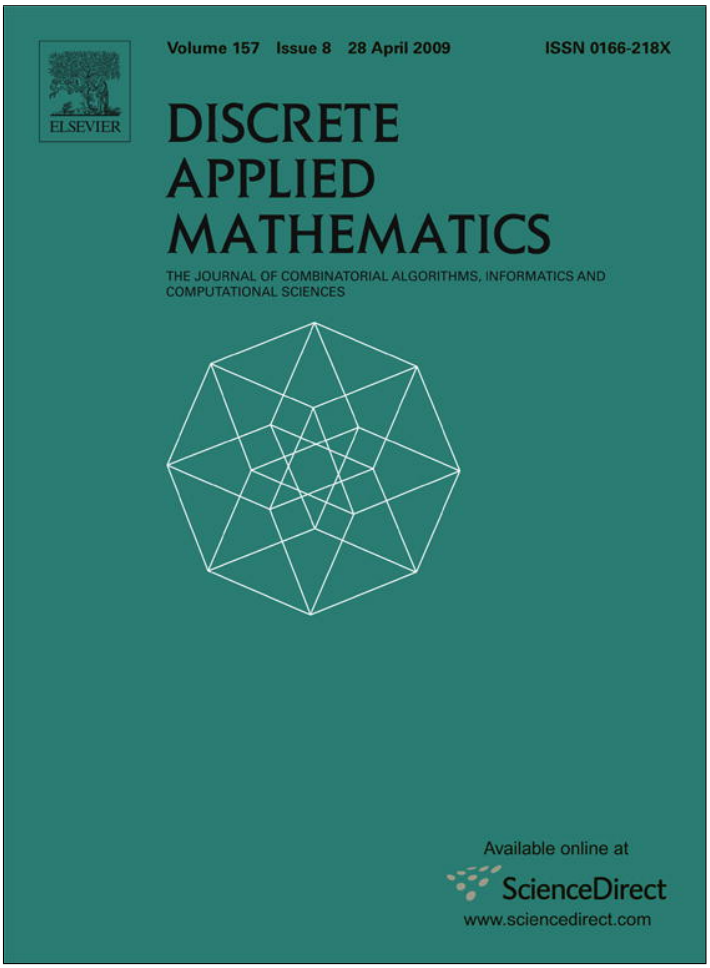

Discrete Applied Mathematics 157 (2009) 1913–1923

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

Edge searching weighted graphs

Öznur Yaşar*, Danny Dyer, David A. Pike, Margo Kondratieva

Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's, NL, Canada

ARTICLE INFO

Article history:
Received 26 November 2007
Received in revised form 3 September 2008
Accepted 25 November 2008
Available online 6 January 2009

Keywords:
Edge searching
Monotonicity
Pathwidth

ABSTRACT

In traditional edge searching one tries to clean all of the edges in a graph employing the least number of searchers. It is assumed that each edge of the graph initially has a weight equal to one. In this paper we modify the problem and introduce the *Weighted Edge Searching Problem* by considering graphs with arbitrary positive integer weights assigned to its edges. We give bounds on the weighted search number in terms of related graph parameters including pathwidth. We characterize the graphs for which two searchers are sufficient to clear all edges. We show that for every weighted graph the minimum number of searchers needed for a not-necessarily-monotonic weighted edge search strategy is enough for a monotonic weighted edge search strategy, where each edge is cleaned only once. This result proves the NP-completeness of the problem.

1. Introduction

The graph searching problem is an extensively studied graph theoretical problem. Its origins date back to the late 1960s in works of Parsons [14] and Breisch [4]. It was first faced by a group of spelunkers who were trying to find a person lost in a system of caves. They were interested in the minimum number of people they needed in the searching team.

Assume that we use a graph to represent a system of gates (which correspond to vertices) and pipes (which correspond to edges) where pipes may have different priorities (depending on size or location). Let us consider these pipes to be full of poison gas. Then we can think of edge-searching as cleaning the system of poison gas. If one gate is left open and if gas leakage can occur through that gate then it will contaminate every pipe that it can reach; that is, all connected pipes with open gates. When a pipe is recontaminated, it will do so to its capacity. That is, even if a recontaminated pipe had been partially (or entirely) cleaned, it must now be fully cleaned again.

Consider a team of searchers and a finite connected graph G with positive integer weights which represent the maximum amount of contamination of edges. We assume the graph to be contaminated initially and our aim is to decontaminate or clean the whole graph by a sequence of steps. At each step we are allowed to do one of the following moves: (1) place a searcher at a vertex, (2) remove a searcher from a vertex, (3) slide a searcher from a vertex along an edge, to an adjacent vertex.

If a searcher slides along an edge $e = uv$ from u to v, then the current positive weight of the edge e is decreased by one if (i) another searcher is stationed at u, or (ii) all other edges incident to u have weight 0 and the current weight of e is 1, or (iii) u is a leaf. When a searcher moves from a leaf u to an adjacent vertex, it is not possible to contaminate the graph through u due to the nature of the system and hence we do not need to place a searcher at u.

If a searcher is stationed at a vertex v, then we say that v is *guarded.* If a path does not contain any guarded vertices, then it is called an *unguarded path*. Assume that the weight of an edge e is decreased after some steps. Then we say that e is *clean* if its weight is reduced to zero and *partially clean* otherwise. If there is an unguarded path that contains one end point

* Corresponding author. Tel.: +1 7097378733; fax: +1 709 737 30 10.
E-mail addresses: oyasar@mun.ca (Ö. Yaşar), dyer@mun.ca (D. Dyer), dapike@mun.ca (D.A. Pike), mkondra@mun.ca (M. Kondratieva).

0166-218X/$ – see front matter
doi:10.1016/j.dam.2008.11.011

of a contaminated or a partially clean edge and one end point of e, then e gets *recontaminated. If e gets recontaminated, its weight goes back to* $w(e)$ *which is its original value.* If in the system there occurs a gap in which an intruder (which may have a diffused form as in the gas leakage scenario) can enter a pipe (i.e. contaminate an edge), then we can no longer consider that pipe as clean (and not even partially clean). Recontamination occurs instantly and there is no order of recontamination.

An *edge search strategy* is a combination of the moves defined above that reduces all edge weights to zero. We say that the graph is *cleaned* when the state of all edge weights being zero simultaneously is achieved. The problem becomes cleaning the graph using the least number of searchers. Such a number for graphs with all edge weights initially equal to one is called the *search number* of the graph and it is denoted by $s(G)$.

We introduce a new version of edge searching called *weighted edge searching*. In our setting we allow the edge weights to be arbitrary positive integers. We call a strategy to clean a weighted graph a *weighted edge search strategy*, and the least number of searchers required in the strategy the *weighted search number* which is denoted by $ws(G)$.

Weighted searching is a reasonable extension of the searching problem, as in many "real-world" situations, an edge in a graph may represent a pipe or a corridor. Traditional edge searching is not robust enough to deal with situations where particular edges may be more important or may require more effort (be it cost or time) to be cleaned. To return to Breisch's original problem, a tunnel in a cave may be quite constricted, allowing only a single searcher through, or broad, requiring several passes to search effectively.

A graph G is said to be *k-searchable* if $s(G) \leq k$. Similarly, a weighted graph is said to be *k-searchable* if $ws(G) \leq k$. It has been shown in [13] that, finding whether a graph G is *k-searchable*, i.e. solving the EDGE SEARCHING problem for G, is NP-complete. Our decision problem can be stated as below.

WEIGHTED EDGE SEARCHING:

Instance. A weighted graph $G = (V, E, w)$ and a positive integer k.
Question. Is G k-searchable?

By transformation from MINIMUM CUT INTO EQUAL SIZED SUBSETS which is known to be NP-complete we see that WEIGHTED EDGE SEARCHING is NP-hard. We will show in Section 5 that our decision problem is in fact NP-complete.

In some situations, we insist that once an edge becomes clean it must be kept clean until the end of the searching strategy. If a strategy follows this rule, then it will be called *monotonic*. If the monotonic strategy is for a weighted graph, then we further insist that a partially cleaned edge will also not get recontaminated. The minimum number of searchers needed for a strategy that is monotonic is denoted as $ms(G)$ for an unweighted graph and by $mws(G)$ for a weighted graph. It has been shown [3,11] that forcing the strategy to be monotonic does not change the search number, hence $s(G) = ms(G)$.

If we are not allowed to remove a searcher from the graph, in which case the searchers will not be able to "jump" from one vertex to another, then the strategy will be called an *internal* strategy [2]. If, on the other hand, the set of clean edges is a connected subgraph of G after each step of the strategy, then the strategy will be a *connected* one. The relationships between these different strategies is examined in [2].

A different weighted model where edges and vertices have dissimilar weights is considered in [1] for internal connected search. Our model together with the rules of cleaning are entirely different from this model.

The problem and its variants are related to many applications such as network security [1]. It has strong connections with the cutwidth of a graph which arises in VLSI design [5] and with pebble games [10]. Because of its closeness with the layout problems, it is related to graph parameters such as pathwidth [7,9], bandwidth [8] and topological bandwidth [12].

In this paper we will first give the preliminaries and illustrate the problem by examples in Section 2. In Section 3, we will give some results that bound the weighted search number, including a bound with pathwidth. Characterization of two-searchable graphs will be done in Section 4. Section 5 is devoted to monotonicity. Finally, we mention some related open problems in Section 6.

2. Preliminaries

In this paper, we will consider finite connected simple graphs unless otherwise stated. Definitions omitted in this paper can be found in [16]. Given a graph $G = (V, E)$, we denote its vertex set by V and edge set by E. A graph $G = (V, E, w)$ is called a *weighted graph* if each edge e of G is assigned a positive integer $w(e)$ called the *weight* of e.

Given two weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, if G and G' have the same underlying graphs, i.e. $V = V'$ and $E = E'$, then G' is said to be *lighter than* G when $w'(e) \leq w(e)$ $\forall e \in E$.

We say that $G' = (V', E', w')$ is a *subgraph* of $G = (V, E, w)$ when $V' \subseteq V$, $E' \subseteq E$ and $w'(e) = w(e)$ $\forall e \in E'$. For given weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, if $V' \subseteq V$, $E' \subseteq E$ and $w'(e) \leq w(e)$ $\forall e \in E'$ then G' is a *lighter subgraph* of G.

In order to define a minor of a given graph we need two operations: *edge deletion*, which corresponds to deleting an edge e, and *edge contraction*, which corresponds to deleting an edge $e = uv$ and identifying the vertices u and v. For given graphs $G = (V, E)$ and $G' = (V', E')$, G' is called a *minor* of G if G' can be obtained from G by a series of edge deletions or contractions. Similarly, given weighted graphs $G = (V, E, w)$ and $G' = (V', E', w')$, we say that G' is a *lighter minor* of G, if G' is a minor of G, considering the corresponding underlying graphs, and $w'(e) \leq w(e)$ $\forall e \in E'$.

A *multigraph* is a graph which may have multiple edges, i.e. $\exists e_1, e_2 \in E$ where $e_1 \neq e_2$ but both have the same end vertices. A *reflexive graph* is a graph which may have loops i.e. $\exists e \in E$ where $e = uu$.

A *path of length n*, denoted as $\mathcal{P}_n$, is a graph with vertex set $V = \{v_0, v_1, \ldots, v_n\}$ and edges $e_i = v_i v_{i+1}$ for every $i = 0, \ldots, n-1$, hence $\mathcal{P}_n = e_0 e_1 \ldots e_{n-1}$. A *suspended path* in a graph G is a path of length at least 2 such that all internal vertices of the path have degree 2 in G.

A *path addition* [16] to G is the addition to G of a path of length at least $n \geq 1$, between two vertices of G introducing $n-1$ new vertices; the added path is called an *ear*. An *ear decomposition* is a partition of E into sets $H_0, H_1, H_2, \ldots, H_k$ such that H_0 is a cycle, and H_i is a path addition to the graph formed by $H_0, H_1, \ldots, H_{i-1}$.

A vertex is called a *cut vertex* if its removal makes the graph disconnected. A graph is *biconnected* or *two-connected* if it has no cut vertices. A *biconnected component* of a graph is a maximal biconnected subgraph.

Theorem 1 (*[17]*). *A graph is biconnected if and only if it has an ear decomposition.*

The set of biconnected components of a graph G forms a graph, called the *block graph* which has as its vertices the biconnected components and cut vertices of G, and there is an edge between two vertices if one of them is a cut vertex and the other is a biconnected component containing that vertex.

Theorem 2 (*[16]*). *The block graph of a connected graph is a tree.*

Let $w_0(e) := w(e)$ denote the initial weight or contamination of the edge e. We denote the contamination of e at step i of a search strategy as $w_i(e)$. Initially all edges are assumed to be contaminated, therefore $w(e) \geq 1\ \forall e \in E$.

If $w_i(e) = 0$, then the weight of edge e is zero at step i and we say that e is *clean* at step i. Note that even if the weight of an edge is zero at some step the edge may be recontaminated at a later point. A vertex u will be said to be *clean* if all edges incident to u are clean.

An *exposed vertex* is a vertex that has at least two edges incident with it, one of which is either clean or partially clean and the other is not clean.

Note that for an unweighted graph G the weighted edge search number, $ws(G)$, will be computed by taking all edge weights equal to one. Similarly, given a weighted graph, $s(G)$ will correspond to the search number of the underlying unweighted graph. Observe that for any weighted graph G we have:

$$s(G) \leq ws(G). \tag{1}$$

Example 1 (*Path of Length n*). The search number is $s(\mathcal{P}_n) = 1$ whereas

$$ws(\mathcal{P}_n) = \begin{cases} 1, & \text{if } w(e) = 1\ \forall e \in E \text{ or when } n = 1 \text{ and } w(e) \text{ is arbitrary;} \\ 2, & \text{if } n \geq 2, \exists e \in E \text{ such that } w(e) \geq 2 \text{ and } w(e_i) \leq 2 \text{ where} \\ & i = 1, \ldots, (n-2), \text{ and when } w(e_0) \text{ or } w(e_{n-1}) \text{ are arbitrary;} \\ 3, & \text{otherwise.} \end{cases}$$

Example 2 (*Loop l*). We know that $s(l) = 2$. For any edge weight we see that $ws(l) = 2$.

Example 3 (*Cycle of Length n:*). For every n cycle of length n, denoted as $\mathcal{C}_n$, observe that $s(\mathcal{C}_n) = 2$.

$$ws(\mathcal{C}_n) = \begin{cases} 2, & \text{if } w(e) = 1 \quad \forall e \in E; \\ 4, & \text{if } \exists e_1, e_2, e_3 \in E, \text{ each of weight at least 3;} \\ 3, & \text{otherwise.} \end{cases}$$

Remark 1. Note that edge searching a weighted graph is not the same as edge searching an unweighted multigraph where each edge e of weight $w(e)$ is replaced with $w(e)$ parallel edges. One example is the path of length two where both edges have weight 3. Then the corresponding unweighted multigraph, with 3 vertices and 6 edges has search number 3 whereas the weighted graph has search number 2.

3. Bounds on weighted search number

In this section we will give results that relate the weighted search number with other parameters. First we will consider the weighted search number of complete graphs. For $n \geq 4$ we know that $s(K_n) = n$, [14].

Proposition 3. *For $n \geq 4$, we have*

$$ws(K_n) = \begin{cases} n+1, & \textit{when all edges have weight at least 3,} \\ n, & \textit{otherwise.} \end{cases}$$

To show the result above it is a simple exercise to construct weighted search strategies with the corresponding weighted search numbers for the complete weighted graphs and to show that n searchers will not suffice for the first case by a contradictory example.

It is known that if H is a minor of G, then

$$s(H) \leq s(G).$$

However, this result does not hold for monotone search [6]. The following theorem implies that weighted edge searching is also minor closed.

Theorem 4. *If $H = (V', E', w')$ is a lighter minor of $G = (V, E, w)$, where G and H are weighted reflexive multigraphs, then*

$$ws(H) \leq ws(G).$$

Proof. Assume that G is cleaned according to a strategy S. We construct a strategy S' for H from S. Let $f : V \to V'$ be the function that is associated with the edge contractions and deletions which transform G to H. Assume that S uses m searchers. We order the vertices that the searchers are placed on G during S as $v_1, v_2, \ldots, v_m$, where v_i's are not necessarily distinct. In S', we place the searchers on vertices $f(v_i)$, $\forall i = 1, \ldots, m$, at the same step as they appeared in the strategy S. If $f(v_i) = f(v_j)$, for $i \neq j$, we place both searchers on the same vertex. To clean H, when a searcher moves from u to v, we will move the searcher on $f(u)$ to $f(v)$. Since we do not necessarily construct S' as an internal or a monotone search, these modifications will give us a search strategy for H. ■

Next, we give two bounds comparing $ws(G)$ and $s(G)$.

Theorem 5. *For a weighted reflexive multigraph G,*

$$ws(G) \leq s(G) + 2.$$

In particular, if $w(e) \leq 2\ \forall e \in E$, then

$$ws(G) \leq s(G) + 1.$$

Proof. Assume that S is a search strategy that uses $s(G)$ searchers to clean G. We will give a search strategy S' that cleans the weighted G using $s(G) + 2$ searchers. To construct S' we start with S and modify it. Assume that $e = uv$ is cleaned by sliding a searcher s_0 from u to v according to S. When cleaning the weighted graph we place two extra searchers, one on each of u and v. Holding these searchers on their places we clean e by s_0, which slides along e back and forth. Next we remove the extra searchers from the end points of e and put them on the end points of the next edge to be cleaned according to S. Note that no extra recontamination will occur, since $s(G)$ searchers were assumed to be sufficient to clean G. Since e was arbitrary we clean the weighted graph in this way. The second inequality is shown similarly. ■

Together with Eq. (1) the theorem above implies that $s(G) \leq ws(G) \leq s(G) + 2$ for any reflexive multigraph G and any weight distribution associated with its edges. In Examples 1 and 3, we saw that for certain distributions of weights equality holds in Theorem 5. In fact, these graphs constitute an infinite family of such graphs.

Let us denote the minimum vertex degree of a graph G by $\delta(G)$. It has been shown in [6] that $s(G) \geq \delta(G) + 1$ for a connected graph G whose minimum degree is at least 3. Below is a similar result for weighted search number.

Theorem 6. *Let $G = (V, E, w)$ be a weighted graph. Assume that $w(e) \geq 3\ \forall e \in E$ and $\delta(G) \geq 3$, then*

$$ws(G) \geq \delta(G) + 2.$$

Proof. We know that $ws(G) \geq s(G) \geq \delta(G) + 1$. Consider a search strategy S on G and let the first vertex cleaned be u. As a first case assume that u is of minimum degree. We claim that S uses at least $\delta(G)+2$ searchers. Let $N(u)$ be the set of vertices adjacent to u. If the graph induced by $N(u)$ forms a clique, then we know from Proposition 3 that we need at least $\delta(G) + 2$ searchers to clean u, and we are done. Hence assume that the graph induced by $N(u)$ does not form a clique. Let the last cleaned edge that is incident to u be $e = uv$. Then u together with all the remaining $\delta(G) - 1$ vertices adjacent to u must each contain a searcher and there must be one more searcher. Hence all $\delta(G) + 1$ searchers are used. Notice that all vertices have minimum degree at least 3, hence none of the $\delta(G) - 1$ searchers located on the $\delta(G) - 1$ adjacent vertices can be moved. This is due to the fact that to be able to remove a searcher from $v \in N(u)$, all neighbors of v other than u must be in $N(u)$ and we need $|N(u)| + 2 = \delta(G) + 2$ searchers, where $|N(u)|$ denotes the cardinality of the set $N(u)$. Hence the validity of our claim is shown. Since all the edges have weight at least 3, the searcher on u cannot be moved either. Therefore, u cannot be cleaned, since $e = uv$ cannot be cleaned by a single free searcher and a searcher on u, because all vertices have degree at least 3.

Otherwise, if u is not of minimum degree, then there are at least $\delta(G) + 1$ vertices adjacent to u. Since the weights of the edges are at least 3, when u is cleaned u together with all its neighbors must contain a searcher, and there must be one more searcher to clean the edges incident to u. This makes in total at least $\delta(G) + 3$ searchers. Hence the theorem is proved. ■

3.1. Weighted edge search number and pathwidth

A *path decomposition* for a graph $G = (V, E)$ is a sequence $X_1, X_2, \ldots, X_r$ of subsets of V such that the following conditions hold:

(1) $\bigcup_{i=1}^{r} X_i = V$,
(2) $\forall e \in E$, $\exists i$ such that X_i contains every end point of e,
(3) For all $1 \leq i \leq j \leq k \leq r$, $X_i \cap X_k \subseteq X_j$.

The *pathwidth* of a graph G, denoted by $pw(G)$, is the minimum $h \geq 0$ such that G has a path decomposition $X_1, X_2, \ldots, X_r$ where $|X_i| \leq h + 1$ for each $i = 1, \ldots, r$.

A *vertex separator* of G is a set of vertices removal of which makes the graph disconnected. A *layout* of a graph $G = (V, E)$ where $|V| = n$ is a one to one mapping L from V to $\{1, 2, \ldots, n\}$. A *partial layout* of G is a one to one mapping L' from a subset V' of V to $\{1, 2, \ldots, n'\}$ where $n' = |V'|$. Given a partial layout L', we define $V_{L'}(i) := \{v \in V : \exists u \in V$ such that $uv \in E$ and $L'(v) \leq i$ and either $L'(u) > i$ or $L'(u)$ is undefined$\}$. For a given partial layout L' where $|\text{domain}(L')| = n'$, the vertex separation of G with respect to L' is defined as $vs_{L'}(G) := \max\{|V_{L'}(i)| : 1 \leq i \leq n'\}$. The *vertex separation* of G is $vs(G) = \min\{vs_L(G) : L$ is a layout of $G\}$.

Kinnersley has shown that pathwidth and vertex separation have the same value for any graph [9]. For edge searching it is known that $s(G)$ is bounded below by $pw(G)$ and above by $pw(G) + 2$ [7]. In Theorem 7 we will prove that the same bounds also hold for weighted edge searching.

For the algorithm in Theorem 7, given a partial layout L' where $\text{domain}(L') = V'$ and $1 \leq i \leq |V'|$, we define the partial layout L_i as the one that assumes the same values for the vertices in $\{L'^{-1}(1), L'^{-1}(2), \ldots, L'^{-1}(i)\}$ and undefined elsewhere. An edge $e = uv$ is *dangling* in L' when $u \in V'$ and $v \notin V'$. A vertex u is *active* in a partial layout L' if $u \in V'$ and u is incident to a dangling edge.

Theorem 7. *For any weighted reflexive multigraph G,*

$$pw(G) \leq s(G) \leq ws(G) \leq pw(G) + 2.$$

Proof. The lower bound is trivial since $pw(G) \leq s(G)$ and $s(G) \leq ws(G)$ [7,9].

To show the upper bound, we will give an algorithm that is derived from Lemma 2.2 in [7]. The algorithm will take as input a weighted graph G, a layout L of G and it will result in all edges of G being simultaneously clean. It will use at most $vs_L(G) + 2$ searchers. Since $pw(G) = vs(G)$ the result will follow.

```
Algorithm weightedsearch(G, L)
for i := 1 to |V|
do
   let v := L^{-1}(i);
   place a searcher s_1 at v;
   for u ∈ V such that L(u) < i and e = uv ∈ E
   do
      place a searcher s_2 at u;
      clean e by sliding s_2 along e back and forth w(e) times;
      remove s_2;
   end
   for every loop e = vv ∈ E
   do
     place a searcher s_2 at v;
     slide s_2 along e back and forth w(e) times;
     remove s_2;
   end
   remove searchers from the vertices that are not active in L_i;
end
```

First observe that at the end of ith iteration the set of active vertices has size no more than $|vs_{L_i}(G)|$. Hence the number of searchers that remains on the graph at the end of every iteration is no more than the search number of the graph.

Notice that before the beginning of the ith iteration of the outer do loop, the subgraph induced by the domain of L_{i-1} is cleaned. Furthermore, at each vertex in the domain of L_{i-1} there is exactly one searcher and there are no other searchers on G. This is why no recontamination occurs during the second do loop. By induction we see that G is cleaned as a result of the algorithm. Note also that at each iteration of the algorithm there are at most $vs_L(G) + 2$ searchers on G since at each iteration the algorithm calls for at most two searchers other than the ones on at most $vs_L(G)$ vertices. Hence for an optimal layout, the algorithm will use at most $vs(G) + 2$ searchers. ■

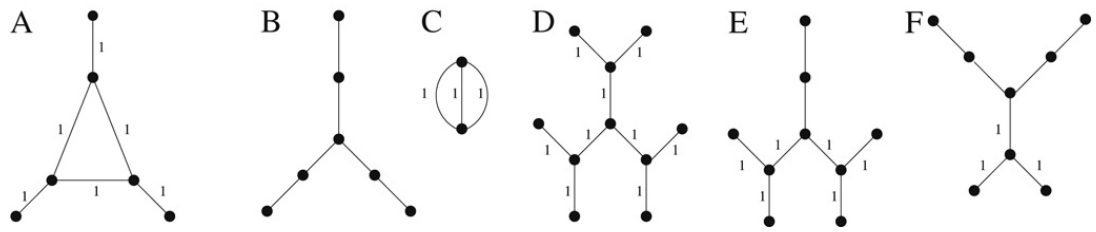

Fig. 1. Forbidden configurations *A, B, C, D, E* and *F*.

4. Restricted weighted edge searching

We now consider graphs that can be cleaned by small numbers of searchers. In the rest of the text we will consider weighted reflexive multigraphs. First, let us give the conditions for a graph to have weighted edge search number equal to one.

Theorem 8. *For a weighted graph G,* $ws(G) = 1$ *if and only if G is either a path with n edges where all edges have weight one, or G is a single edge of an arbitrary weight.*

For the proof note that for $ws(G)$ to be 1, G cannot have a vertex v such that $deg(v) > 2$, in which case we need at least two searchers to clean v.

4.1. Two-searchable graphs

Here we are going to characterize graphs for which $ws(G) \leq 2$. We will introduce the notion of containment relation between two graphs for which we will define a set of rules called *reduction rules*;

(A) Any suspended path with edge weights 1 is reduced to a single edge of weight 1. Hence, in the reduced graph, there are no degree 2 vertices whose incident edges both have weight 1.

(B) Consecutive internal edges of a suspended path that have weight 2 are reduced to a single edge of weight 2.

We say that *G reduces to* G' if G' is obtained from G by the reduction rules. For instance, a path $\mathcal{P}_n$ where all edges have weight 1, will reduce to a single edge of weight 1. A cycle $\mathcal{C}_n$ where all edges have weight 1, will reduce to a loop. If G and H reduce to the same graph, then we say that G and H have the same reduction.

Any search strategy for a graph G can be transformed into a search strategy that uses the same number of searchers for the reduced G and vice versa. Hence, we have the following result.

Lemma 9. *If G and H have the same reduction, then* $ws(G) = ws(H)$.

We say that *G contains F* if there exists a graph H such that

(1) G and H have the same reduction, and
(2) F is a lighter minor of H.

Theorem 10. *For any reduced graph G, the following are equivalent:*

(1) $ws(G) \leq 2$
(2) *G either does not contain any of the configurations* A, B, C, D, E, F *given in Fig. 1 or the following conditions hold simultaneously:*
  (a) *G does not contain any edge e that is not a loop or a pendant edge and* $w(e) > 2$,
  (b) *G does not contain any 2-cycle having an edge of weight greater than* 1,
  (c) *G does not contain the graphs D, E and F where any two pendant edges with a common end are replaced with a loop of weight* 1,
  (d) *Every vertex of degree two in graphs B, E and F has an edge incident to it with weight 1 and the other edge with weight* 2.
(3) *G consists of a path with vertex set* $\{v_1, v_2, \ldots, v_n\}$ *together with the following conditions:*
  (a) *The only edges between* v_i*'s are the ones between each consecutive pair and they can either be a single edge of weight at most* 2 *or a pair of edges of weight* 1.
  (b) *There may be pendant edges or loops of arbitrary weight attached to each* v_i.

Proof. We will prove the equivalence by showing that (1) $\Rightarrow$ (2) $\Rightarrow$ (3) $\Rightarrow$ (1).

(1) $\Rightarrow$ (2) None of the graphs in *(2)* have weighted search number less than 3. The result follows since weighted edge searching is minor closed due to Theorem 4.

(2) $\Rightarrow$ (3) G does not contain C, hence there are no chords in G. By Theorem 1 edges or cycles are the only possible biconnected components of G. Because of condition (2b), edges of the cycles can only have weight 1. On the other hand, G does not contain A, hence at most two vertices of a cycle can have degree at least 3. This implies that the only biconnected

components of G are paired edges with weight 1, loops and edges with arbitrary weight. Then by Theorem 2, G must be a tree together with paired edges of weight 1 and loops of arbitrary weight. Due to condition (2a), the internal edges of the tree can have weight at most 2. The result follows if we show that when all of the vertices of degree one are removed, the resulting graph is a path, with possible loops or paired edges that have weights as described in (3). Assume that it is not true. Then, there must be a vertex of G that has three different neighbors none of which is a leaf. When none of these neighbors have degree less than three, G would contain D together with condition (2c). Similarly, when all of these neighbors have degree two, G would contain B together with condition (2d). When one of these neighbors has degree two, G would contain E together with condition (2c) or (2d). Finally, when two of these neighbors both have degree two, G would contain F together with condition (2c) or (2d). Since all of these configurations are forbidden, we arrive at a contradiction. Therefore, G has the form given in (3).

(3) $\Rightarrow$ (1) The first vertex v_1 can be cleaned by putting both searchers on v_1, then by keeping one of the searchers on v_1 and cleaning the incident loops or leaves by the other searcher. Then both searchers can either move along the edge that connects v_1 to v_2 or each can move along one of the paired edges of weight 1. The same procedure can be applied to v_2 and in this way one can clean the whole graph. ■

5. Monotonicity of weighted edge searching

In this section, we will show that if there exists a weighted search strategy for a weighted graph G using at most k searchers, then there exists a monotonic weighted search strategy for G using at most k searchers. To prove this, we will modify and use the so called *crusade method*, which is a widely used proof method to show monotonicity in edge searching or its variants. Here the terminology is similar to that used in [3].

Notice that when sliding a searcher along an edge $e = uv$ from u to v, if no recontamination is possible from u, then either e becomes clean or the current weight of e decreases from k to $k-1$, where $k > 1$, in which case we say that *partial cleaning* is done.

At step i, let the set of cleaned edges correspond to A_i, the set of partially cleaned edges correspond to P_i and let Z_i be the set of vertices where at least one searcher is located. In the edge search there may be more than one searcher located at a vertex, hence we consider Z_i to be a multiset. Set difference, namely, $Z_i \setminus \{u\}$ corresponds to removing one copy of u from the multiset Z_i.

A weighted search strategy S that uses n steps for a weighted graph $G = (V, E, w)$ can be recorded as a sequence of a triples of sets

$$S = (A_i, P_i, Z_i)_{i=0}^{n}$$

such that $A_i \subseteq E, P_i \subseteq E \setminus A_i, Z_i \subseteq V$ for $0 \leq i \leq n$ and $A_0 = P_0 = P_n = \emptyset, A_n = E$. If $v \in V$ is incident with at least one edge in $A_i \cup P_i$ and at least one edge in $E \setminus A_i$, then $v \in Z_i$.

The following are the only possible situations we may encounter during a weighted search:

(1) Putting new searchers: $A_i = A_{i-1}, Z_i \supseteq Z_{i-1}, P_i = P_{i-1}$.
(2) Recontamination:
   - by removing searchers: $A_i \subseteq A_{i-1}, Z_i \subseteq Z_{i-1}, P_i \subseteq P_{i-1}$, or,
   - by sliding a searcher along an edge: $A_i \subseteq A_{i-1}, Z_i = (Z_{i-1} \setminus \{u\}) \cup \{v\}, P_i \subseteq P_{i-1}$
(3) Partial Cleaning: For $uv = e \in E$ such that $w(e) \geq 2, A_i = A_{i-1}$,

$$Z_i = (Z_{i-1} \setminus \{u\}) \cup \{v\} \quad \text{and} \quad P_i = \begin{cases} P_{i-1}, & 2 \leq w_{i-1}(e) < w(e) \text{ or;} \\ P_{i-1} \cup \{e\}, & w_{i-1}(e) = w(e). \end{cases}$$

(4) Cleaning: For $uv = e \in E, A_i = A_{i-1} \cup \{e\}, Z_i = (Z_{i-1} \setminus \{u\}) \cup \{v\}$ and

$$P_i = \begin{cases} P_{i-1} \setminus \{e\}, & 2 \leq w(e) \text{ or;} \\ P_{i-1}, & w(e) = 1. \end{cases}$$

We break up the steps of the strategy so that at most one action is done at each move. In this way, in each move we force that at most one edge gets cleaned, partially cleaned or contaminated.

For a given edge set E and $A \subseteq E, P \subseteq E \setminus A$, $\delta(A, P)$ denotes the set of vertices that have at least two edges, e_1 and e_2, incident to it such that $e_1 \in A \cup P$ and $e_2 \in E \setminus A$.

In edge searching, if A is the set of clean edges and P is the set of partially clean edges at some instant, then $\delta(A, P)$ would correspond to the set of exposed vertices. Every exposed vertex must contain a searcher.

In this section we will need the following lemma. For a proof, refer to [18].

Lemma 11 (*Submodularity*). *For given pairs of subsets of E, (A, P), (B, R) where $P \subseteq E \setminus A$ and $R \subseteq E \setminus B$, the following holds:*

$$|\delta((A \cap B), (P \cap R))| + |\delta((A \cup B), (P \cup R))| \leq |\delta(A, P)| + |\delta(B, R)|. \tag{2}$$

5.1. Pairs of Crusades

Consider a weighted search strategy for a given graph $G = (V, E, w)$. For a sequence of pairs of subsets of the edge set E, $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$, where $Y_i \subseteq E \setminus X_i$, for $0 \leq i \leq n$ and $X_0 = Y_0 = Y_n = \emptyset, X_n = E$, consider the sequence $(X_0, X_1, \ldots, X_n)$, where $\forall i = 0, 1, \ldots, n$ and $\forall e \in X_i$, there exists a step $j \leq i$ such that $w_j(e) = 0$. Then $(X_0, X_1, \ldots, X_n)$ is called a **crusade** associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$ if for all $1 \leq i \leq n$

$$|X_i \setminus X_{i-1}| + |Y_i \setminus Y_{i-1}| \leq 1. \tag{3}$$

We say that a crusade uses at most k searchers if $|\delta(X_i, Y_i)| \leq k$ for all $0 \leq i \leq n$.
A crusade is **progressive** if X_i's form a nested sequence, i.e., $X_0 \subseteq X_1 \subseteq \cdots \subseteq X_n$ and for all $1 \leq i \leq n$,

$$|X_i \setminus X_{i-1}| + |Y_i \setminus Y_{i-1}| = 1. \tag{4}$$

Proposition 12. *If $ws(G) \leq k$, then there exists a crusade using at most k searchers.*

Proof. Let $ws(G) \leq k$ and let $(A_0, P_0, Z_0), (A_1, P_1, Z_1), \ldots, (A_n, P_n, Z_n)$ be a weighted search strategy for G. Then $|Z_i| \leq k$ for $0 \leq i \leq n$. From the definition of $\delta(\cdot, \cdot)$, we know that if $v \in \delta(A_i, P_i)$ then v is an exposed vertex. Since on every exposed vertex there must be a searcher, $\delta(A_i, P_i) \subseteq Z_i$ and hence $|\delta(A_i, P_i)| \leq |Z_i| \leq k$. Each A_i corresponds to a set of clean edges at step i, hence $\forall e \in A_i, \exists j \leq i$ such that $w_j(e) = 0$. Eq. (3) holds because of the definition of a weighted search since each step corresponds to only one action. Therefore associated with $(A_0, P_0), (A_1, P_1), \ldots, (A_n, P_n)$, the sequence $A_0, A_1, \ldots, A_n$ is a crusade that uses at most k searchers. ■

Proposition 13. *If there exists a crusade using at most k searchers, then there exists a progressive crusade using at most k searchers.*

Proof. To each sequence of pairs $(X_0, Y_0), (X_1, Y_1), \ldots, (X_N, Y_N)$ one can associate two numbers $a(N) = \sum_{i=0}^{N}(|\delta(X_i, Y_i)| + 1)$ and $b(N) = \sum_{i=0}^{N} |X_i|$. Among all crusades $(X_0, X_1, \ldots, X_N)$ using at most k searchers and associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_N, Y_N)$ we will pick the one for which

(1) $a(N)$ is minimum and
(2) $b(N)$ is minimum subject to condition (1).

We denote such a crusade by $C = (X_0, X_1, \ldots, X_n)$.

If $|Y_i \setminus Y_{i-1}| = 1$, then $|X_i \setminus X_{i-1}| = 0$ due to Eq. (3). Instead, assume that $|Y_i \setminus Y_{i-1}| = 0$ and $|X_i \setminus X_{i-1}| = 0$. Then $|Y_{i+1} \setminus Y_{i-1}| \leq 1$ and $|X_{i+1} \setminus X_{i-1}| \leq 1$. Therefore $(X_0, X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$ is a crusade with respect to $(X_0, Y_0), (X_1, Y_1), \ldots, (X_{i-1}, Y_{i-1}), (X_{i+1}, Y_{i+1}), \ldots, (X_n, Y_n)$. For this sequence $a(N)$ takes a smaller value than for C, which contradicts our assumption. Therefore $|X_i \setminus X_{i-1}| = 1$.

We only need to show that X_i's form a nested sequence. Observe that if

$$|\delta(X_{i-1} \cup X_i, Y_{i-1} \cup Y_i)| < |\delta(X_i, Y_i)|,$$

then $(X_0, X_1, \ldots, X_{i-1}, X_{i-1} \cup X_i, X_{i+1}, \ldots, X_n)$ is a crusade with respect to $(X_0, Y_0), (X_1, Y_1), \ldots, (X_{i-1}, Y_{i-1}), (X_{i-1} \cup X_i, Y_{i-1} \cup Y_i), \ldots, (X_n, Y_n)$. For this sequence $a(N)$ takes a smaller value than for C, hence

$$|\delta(X_{i-1} \cup X_i, Y_{i-1} \cup Y_i)| \geq |\delta(X_i, Y_i)|. \tag{5}$$

Combining Eqs. (2) and (5), we have

$$|\delta(X_{i-1} \cap X_i, Y_{i-1} \cap Y_i)| \leq |\delta(X_{i-1}, Y_{i-1})|.$$

From the result above we observe that $(X_0, X_1, \ldots, X_{i-2}, X_{i-1} \cap X_i, X_i, \ldots, X_n)$ is a crusade with respect to $(X_0, Y_0), (X_1, Y_1), \ldots, (X_{i-2}, Y_{i-2}), (X_{i-1} \cap X_i, Y_{i-1} \cap Y_i), (X_i, Y_i), \ldots, (X_n, Y_n)$. From the minimality of $b(N)$ for C we must have

$$|X_{i-1} \cap X_i| \geq |X_{i-1}|.$$

Hence $X_{i-1} \subseteq X_i$. ■

Proposition 14. *If there exists a weighted search strategy $S = (A_i, P_i, Z_i)_{i=0}^{n}$ for a weighted graph $G = (V, E, w)$ that uses k searchers, then there exists a weighted search strategy $S' = (A'_j, P'_j, Z'_j)_{j=0}^{m}$ for G such that $|P'_j| \leq 1$ $\forall j = 0, 1, \ldots, m$ and S' uses k searchers as well. Furthermore, for S' the following hold:*

(1) *If $e \in P'_j$ and $w_j(e) = 1$, then $w_{j+1}(e) = 0$, $P'_{j+1} = \emptyset$ and $A'_{j+1} = A'_j \cup \{e\}$.*
(2) *If $P'_j = P'_{j-1} = \{e\}$, then $A'_j = A'_{j-1}$.*

Proof. From S, we construct the required strategy S' that uses the same number of searchers to clean G. First, if an edge $e = uv$, $w(e) \geq 2$ is partially cleaned at step i in S by sliding a searcher s_1 from u to v, in S' we remove s_1 from u and place it on v. We modify S' step by step so that according to its final version G will be cleaned. While modifying S' we only need to consider edges that have capacity at least two, since edges of unit capacity are never in any P'_j.

In S during the steps that reduce the weight of an edge $e = uv$, there is a step that results in enough searchers on the ends of e to clean e. If this happens at the kth step in S, then during S' the cleaning can be done successively instead of the kth step. So in S' we will have $w(e_{k_1}) = w(e) - 1$, $w(e_{k_2}) = w(e) - 2, \ldots, w(e_{k_{w(e)-1}}) = 1$, and $w(e_{k_{w(e)}}) = 0$. We have to show that we can clean every edge in this way.

If a searcher s_1 ends up on u in S after the kth step but s_1 ends up on $v \neq u$ in S' after the $k_{w(e)}$th step, then we remove s_1 from v and place it on u at steps $k_{w(e)+1}$ and $k_{w(e)+2}$.

Let e_0 be the first edge with weight at least 2 that gets cleaned in S. If $e_0 = uv$ is a pendant edge, where $deg(v) = 1$, then we need at least two searchers to clean it, since $w(e_0) \geq 2$. If these two searchers are put on one or both ends of e_0 for the first time at step k in strategy S, in S' we clean e_0 in steps $k_1, k_2, \ldots, k_{w(e_0)}$ by letting one of them guard u and the other slide on e until it becomes clean.

If e_0 is not a pendant edge, then we need at least 3 searchers, two to guard the ends of e_0 and one to slide along e_0, when $w(e_0) \geq 2$. If these three searchers are put on ends of e_0 for the first time at step k in strategy S, in S' we clean e_0 in steps $k_1, k_2, \ldots, k_{w(e_0)}$.

Assume that we continue cleaning edges according to S and construct S' in this way. Let $e = uv$ be the next edge that is cleaned according to S at step i.

Note that e might have been cleaned and contaminated during S before step i. But we know that there exists a step $j < i$ in S such that $w_{j-1}(e) = w(e)$, $w_j(e) = w(e) - 1$, $w_{i-1}(e) = 1$, $w_i(e) = 0$ and there exists no k such that $j < k < i$ and $w_k(e) = w(e)$. In other words, e does not become recontaminated between step j and step i.

If e is not a pendant edge, then just before the ith step, there must be at least one searcher located on each of u and v.

Case 1: *e is not a pendant edge and $w(e) = 2$*

At some step, say j in S, $w_{j-1}(e) = 2$ and $w_j(e) = 1$. Here j is a step between the last time e was cleaned and the ith step.

(1) If at the jth step two searchers were located on u and v, one on each vertex, and a third searcher was sliding along e, either from u to v or from v to u, in S' we can clean e in two steps, j_1 and j_2, using the same three searchers.

(2) If at the $(j-1)$th step two searchers, s_1 and s_2, were located on u and at the jth step one searcher, s_2, slid along e from u to v, there are two possibilities to reduce the weight from 1 to 0. If a third searcher slides along e at step k, then we clean e in S' in two steps, k_1 and k_2, with these 3 searchers. Otherwise s_2 may slide along e from v to u at step k (or s_1 may slide along e from u to v at step k, which can be transferred to S' similarly). Notice that all of the edges incident to v are contaminated at step j (since they are on an unguarded path to a contaminated edge e). Furthermore at step $k-1$ all edges incident to v, except for e, must be clean. Since otherwise when s_2 slides along e from v to u, it would not be partially cleaning e. Hence, all edges incident to v, except for e, must be cleaned between the jth step and the kth step and they all have weight 1. At some step l, such that $j < l < k$, one of those edges, say e_1, gets clean by a searcher s_3 sliding along e_1 either starting from v or ending at v. Therefore in S', we clean e in steps l_1, l_2 where s_3 slides two times along e.

Case 2: *e is not a pendant edge and $w(e) \geq 3$*

Since the number of searchers needed for reducing the weight from 3 to 2 and 2 to 1 is the same as reducing the weight from n to $n-1$ and from $n-1$ to $n-2$ for $n \geq 3$, it is enough to consider the case $w(e) = 3$.

At some point, say j during S, $w_{j-1}(e) = 3$ and $w_j(e) = 2$. If this is done by using three searchers, then e can be cleaned in S' in three steps j_1, j_2, j_3 which would replace the jth step of S. If in the jth step two searchers, s_1 and s_2, are used by placing both of them on u and sliding one of them to v, after this step, u or v cannot be left unguarded. Hence, to reduce the weight from 2 to 1 we need one more searcher, say s_3, which is going to slide along e at step k in S. Accordingly, in S', we replace step k with steps k_1, k_2, k_3 where s_3 slides back and forth along e.

If $e = uv$ is a pendant edge where $\deg(v) = 1$, we consider two cases.

Case 3: *$e = uv$ is a pendant edge and $w(e) = 2$*

The weight of e should go from 2 to 1 in S at some step, say at k. If this is done by a searcher s_1 sliding from u to v, then there must be another searcher on u. In S', we replace the kth step with steps k_1, k_2 in which s_1 slides back and forth along e twice.

If the weight of e is reduced from 2 to 1 by a searcher s_1 sliding from v to u, then after this step u must always be guarded by a searcher. There are two ways to reduce the weight from 1 to 0.

Another searcher, say s_2, may slide along e at the kth step of S. In S' we replace the kth step with steps k_1, k_2 in which s_2 slides back and forth along e twice.

In S, all the edges incident to u may get clean and s_1 may slide back from u to v. Then during cleaning of an edge $e_1 \neq e$ incident to u, there must be a searcher s_2 either sliding from u or ending at u, say at the lth step. Accordingly, in S', just after the lth step, we insert steps l_1 and l_2 in which s_2 guards u and s_1 slides twice back and forth along e.

Case 4: *$e = uv$ is a pendant edge and $w(e) \geq 3$*

Again, we only need to consider the case $w(e) = 3$. At some step the weight of e should go from 3 to 2 during S, say at step j.

If this is done by a searcher s_1 sliding from u to v, then there must be another searcher on u. In S', we replace the jth step with steps j_1, j_2, j_3 in which s_1 slides back and forth along e three times.

If a searcher s_1 slid from v to u to reduce the weight of e from 3 to 2, then, after this step u cannot be left unguarded, otherwise the edge would be recontaminated. Now, to reduce the weight from 2 to 1, another searcher, say s_2, has to slide along e. If this happens at the kth step of S, in S' we replace the kth step with steps k_1, k_2, k_3 in which s_2 slides back and forth along e. ■

Proposition 15. *Assume that S is a weighted search strategy for $G = (E, V, w)$ that uses k searchers. Then there exists a progressive crusade $(X_0, X_1, \ldots, X_n)$ associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$, where $Y_i \subseteq E \setminus X_i$, for $0 \leq i \leq n$, that uses at most k searchers such that both the following conditions hold:*

(1) *If $|Y_i \setminus Y_{i-1}| = 0$, then either*
  (a) $Y_i = \emptyset, Y_{i-1} = \emptyset, X_i \setminus X_{i-1} = \{e\}$, *and* $w(e) = 1$, *or;*
  (b) $Y_i = \emptyset, Y_{i-1} = \{e\}, X_i \setminus X_{i-1} = \{e\}$, *and* $w(e) \geq 2$.
(2) *If $|Y_i \setminus Y_{i-1}| = 1$, then $Y_{i-1} = \emptyset, Y_i = \{e\}, X_i \setminus X_{i-1} = \emptyset, X_{i+1} \setminus X_i = \{e\}$, and $w(e) \geq 2$.*

Proof. Using the procedure given in the proof of Proposition 14, we construct a weighted search (A'_i, P'_i, Z'_i) such that $|P'_i| \leq 1, \forall i$. Next we delete the (A'_i, P'_i, Z'_i)'s for which $|P'_i| = 1$ and $\exists j \neq i$ such that $(A'_j, P'_j) = (A'_i, P'_i)$ except for the (A'_i, P'_i, Z'_i)'s such that $w_i(e) = 1$ where $\{e\} = P'_i$. We apply Propositions 12 and 13 to this reduced sequence and obtain a progressive crusade. The two conditions of the theorem follow from the construction of S' and the implications of Proposition 14 if we let $X_i = A_i$ and $Y_i = P_i$. If two consecutive partially cleaned sets are empty, then S' is cleaning an edge of weight 1, which corresponds to part 1(a). Part 1(b) is the same as part (1) of Proposition 14. In both of them an edge e which is partially clean at step $i-1$ becomes clean at step i. Finally, during the consecutive steps where an edge is partially cleaned no other edge is cleaned. This is part (2). ■

In the proof of Theorem 16 we will consider the indices of the sets X_i and Y_i as levels. Hence the levels consists of steps. The ith level may not correspond to the ith step in the strategy due to the construction of the progressive crusade in the proof of Proposition 15.

Theorem 16. *If there exists a weighted search S using at most k searchers for a weighted graph G, then there exists a monotone edge search S' using at most k searchers for G.*

Proof. Proposition 14 implies that from S we can construct a weighted search S' that uses at most k searchers and $|P'_i| \leq 1$ for all i. Proposition 15 ensures that there exists a progressive crusade $X_0, X_1, \ldots, X_n$ associated with $(X_0, Y_0), (X_1, Y_1), \ldots, (X_n, Y_n)$ which can be obtained from S' and it uses at most k searchers. We construct a monotone weighted search strategy inductively that cleans the edges in the order $e_1, e_2, \ldots, e_m$, where for each e_j there exists i such that $X_i \setminus X_{i-1} = \{e_j\}$. Assume that we cleaned the edges $e_1, e_2, \ldots e_{l-1}$ in this order and no edge other than these is cleaned. Assume that we finished cleaning e_{l-1} at the end of $(i-1)$th level. We show that in the next steps we will clean e_l. We will use the implications of Proposition 15. We consider three cases.

Case 1. $|Y_i \setminus Y_{i-1}| = 0$, $Y_i = \emptyset$, and $Y_{i-1} = \emptyset$. Here $X_i \setminus X_{i-1} = \{e_l\}$, and $w(e_l) = 1$,. By using the argument in [3], we see that e_l can be cleaned with at most k searchers. Refer to [18] for details.

Case 2. $Y_i = \emptyset$, $Y_{i-1} = \{e_l\}$. We have $w(e_l) \geq 2$. We consider the cases whether e_l is a pendant edge or not.

(1) If e_l is not a pendant edge, then at $(i-1)$th level, there should be at least one searcher on each end of e_l, say s_1 on u and s_2 on v. If $|\delta(X_{i-1}, Y_{i-1})| + 1 \leq k$, then we can finish the cleaning of e_l by the searcher that is free, i.e., the one that is not on any exposed vertex. Otherwise $|\delta(X_{i-1}, Y_{i-1})| + 1 > k$ and since $|\delta(X_{i-1}, Y_{i-1})| \leq k$, we must have $|\delta(X_{i-1}, Y_{i-1})| = k$. There are four subcases to consider when e_l is not a pendant edge.

In the first subcase let each of u and v have at least one edge incident to them other than e_l that is already clean, hence in X_{i-1}. Then before $(i-1)$th level there must be two searchers located on each of u and v. In this case, the only way that e_l became partially clean for the first time is that a third searcher, other than the ones on u and v, slid along e_l. Hence, e_l can be cleaned by this third searcher in the next steps.

Consider together the subcase where there are no clean edges incident to u or v and the subcase where only one of the end points of e_l, say u, has an edge incident to it that is clean and an edge that is contaminated. In both of these cases the exposed vertices in level $(i-1)$ are the same as in level i, hence $\delta(X_{i-1}, Y_{i-1}) = \delta(X_i, Y_i)$. Hence none of the searchers can move from their places during the steps between these levels. On the other hand, we know that $|X_i \setminus X_{i-1}| = \{e_l\}$. But there is no possible way to clean e_l when none of the searchers is moving. Hence we arrive at a contradiction.

The last subcase is where all edges other than e_l incident to one of the end points of e_l, say u, are already clean and all edges incident to v are contaminated. We know that $u \in \delta(X_{i-1}, Y_{i-1})$ and $u \notin \delta(X_i, Y_i)$ since at level i all edges incident to u are in X_i (since X_i's are nested). The only way this can happen is that either $w_{i-1}(e_l) = 1$, and we are done, or $w_{i-1}(e_l) > 1$ and a third searcher slides along e_l to clean it totally during the steps between these levels, which is impossible if $\delta(X_{i-1}, Y_{i-1}) = k$ since none of the searchers can move.

(2) If e_l is a pendant edge, let $\deg(v) = 1$ and $\deg(u) > 1$. We only need to consider two subcases. When there is at least one edge incident to u that is clean and hence in X_{i-1}, then before the $(i-1)$th level there must have been a searcher, say

s_1, located on u. As in the previous subcase, the only way for e_l to become partially clean is that a searcher other than s_1, say s_2, slides along e_l at the step that corresponds to level $i-1$. Hence, e_l can be cleaned with s_2 together with s_1 which will be kept on u as a guard.

Finally, if all edges incident to u other e_l are contaminated, observe that $u \in \delta(X_{i-1}, Y_{i-1}) = \delta(X_i, Y_i)$. This implies that none of the searchers could move during the steps between the levels $i-1$ and i, which contradicts $|X_i \setminus X_{i-1}| = \{e_l\}$.

Case 3. $|Y_i \setminus Y_{i-1}| = 1$. We have $Y_{i-1} = \emptyset$, $Y_i = \{e_l\}$. We know that $X_i \setminus X_{i-1} = \emptyset$, $X_{i+1} \setminus X_i = \{e_l\}$, and $w(e) \geq 2$. This case reduces to the previous case by shifting all the observations to the levels i and $i+1$. Applying the same procedures we can finish the cleaning of e_l at the successive steps after level i. ■

Theorem 16 implies that $ws(G) = mws(G)$. From this we deduce that the WEIGHTED EDGE SEARCHING problem belongs to NP, since we only need to guess in which order the edges are cleaned and then to check whether the edges can be cleaned according to this sequence using at most k searchers. In Section 1 we noted that the problem is NP-hard. It follows from these two observations that the problem is NP-complete. Hence we have the following.

Corollary 17. *WEIGHTED EDGE SEARCHING is NP-complete.*

6. Conclusion

In this paper, we introduced a new weighted version of the edge searching problem. Our motivation was that there may be networks where links have different capacities or importance factors. In this setting decontamination is not the same for all edges.

For any graph G, Theorem 8 implies that the pathwidth and the weighted edge search number may differ by at most 2. Furthermore, since $pw(G) \leq s(G) \leq ws(G) \leq pw(G)+2$, if $s(G) = pw(G)+2$, we have $s(G) = ws(G)$. From this result a characterization of graphs for which any weight distribution would give the same search number and weighted search number can be obtained.

In Section 4 we identified the forbidden graphs containment of which prevents the weighted search number to be at most 2. The characterization of graphs G for which the search number is at most 3 is done in [13] and it is more complicated than the characterization of graphs such that $s(G) \leq 2$. The characterization for $s(G) \leq 4$ is not known. For fixed k the set of forbidden graphs is called the *obstruction set*. The theory on graph minors built by Robertson and Seymour [15] implies that since weighted edge searching is minor closed, due to Theorem 4, the obstruction set is finite. However, a construction of the obstruction set is not known for any fixed $k \geq 4$.

The main result of the paper is Theorem 16, namely, for any weighted graph G, we have $ws(G) = mws(G)$. One implication of this important result is the NP-completeness of the weighted edge searching problem. Secondly, monotonicity implies that one can always use a monotonic strategy without any increase on the number of searchers. This is important since when considering edge searching problems most of the time it is simpler to work with a monotonic strategy. Monotonicity is also advantageous when there are costs related to moving along an edge and number of moves. If sliding along an edge is very costly, a monotonic strategy would be desired.

Acknowledgments

We are grateful to Dr. D. Morgan and Mr. N. McKay for fruitful discussions. We would like to thank the referees for comments and criticism. The authors were supported in part by NSERC.

References

[1] L. Barrière, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an intruder by mobile agents, in: 14th ACM Symp. on Parallel Algorithms and Architectures, SPAA'02, Winnipeg, August 10–13, 2002, pp. 200–209.
[2] L. Barrière, P. Fraigniaud, N. Santoro, D. Thilikos, Searching is not jumping, in: Proc. 29th Workshop on Graph Theoretic Concepts in Computer Science, WG 2003, in: Lecture notes in Computer Science, vol. 2880, 2003, pp. 34–45.
[3] D. Bienstock, P. Seymour, Monotonicity in graph searching, Journal of Algorithms 12 (1991) 239–245.
[4] R.L. Breisch, An intuitive approach to speleotopology, Southwestern Cavers 6 (1967) 72–78.
[5] F. Chung, On the cutwidth and the topological bandwidth of a tree, SIAM Journal of Algebraic Discrete Methods 6 (2) (1985) 268–277.
[6] D. Dyer, Sweeping graphs and digraphs, Ph.D. Thesis, Simon Fraser University, Canada, 2004.
[7] J.A. Ellis, I.H. Sudborough, J.S. Turner, The vertex separation and search number of a graph, Information and Computation 113 (1994) 50–79.
[8] F.V. Fomin, P. Heggernes, J.A. Telle, Graph searching, elimination trees, and a generalization of bandwidth, Algorithmica 41 (2) (2004) 73–87.
[9] N.G. Kinnersley, The Vertex Separation Number of a graph equals its path-width, Information Processing Letters 42 (1992) 345–350.
[10] L.M. Kirousis, C.H. Papadimitriou, Searching and pebbling, Theoretical Computer Science 47 (2) (1986) 205–218.
[11] A.S. LaPaugh, Recontamination does not help to search a graph, Journal of ACM 40 (1993) 224–245.
[12] F.S. Makedon, C.H. Papadimitriou, I.H. Sudborough, Topological bandwidth, SIAM Journal Algebraic Discrete Methods 6 (1985) 418–444.
[13] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, C.H. Papadimitriou, The complexity of searching a graph, ACM 35 (1) (1988) 18–44.
[14] T. Parsons, Pursuit-evasion in a graph, in: Theory and Applications of Graphs, in: Lecture Notes in Mathematics, Springer-Verlag, 1976, pp. 426–441.
[15] N. Robertson, P.D. Seymour, Graph minors—a survey, in: I. Anderson (Ed.), Surveys in Combinatorics, Cambridge University Press, Cambridge, 1985, pp. 153–171.
[16] D.B. West, Introduction to Graph Theory, Prentice Hall, 1996.
[17] H. Whitney, Congruent graphs and the connectivity of graphs, American Journal of Mathematics 54 (1932) 150–168.
[18] Ö. Yaşar, Algorithmic complexity and extremality characterizations for edge searching and its variations, Ph.D. Thesis, Memorial University of Newfoundland, Canada, 2008.