Theoretical Computer Science 412 (2011) 1208–1219

Contents lists available at ScienceDirect

Theoretical Computer Science

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

Fast edge searching and fast searching on graphs

Boting Yang

Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada

ARTICLE INFO

Article history:
Received 8 September 2010
Accepted 27 December 2010
Communicated by D.-Z. Du

Keywords:
Graphs
Fast searching
Edge searching
Pursuit-and-evasion problems
Cops-and-robber games

ABSTRACT

Given a graph $G = (V, E)$ in which a fugitive hides on vertices or along edges, graph searching problems are usually to find the minimum number of searchers required to capture the fugitive. In this paper, we consider the problem of finding the minimum number of steps to capture the fugitive. We introduce the fast edge searching problem in the edge search model, which is the problem of finding the minimum number of steps (called the fast edge-search time) to capture the fugitive. We establish relations between the fast edge searching and the fast searching that is the problem of finding the minimum number of searchers to capture the fugitive in the fast search model. While the family of graphs whose fast search number is at most k is not minor-closed for any positive integer $k \geq 2$, we show that the family of graphs whose fast edge-search time is at most k is minor-closed. We establish relations between the fast (fast edge) searching and the node searching. These relations allow us to transform the problem of computing node search numbers to the problem of computing fast edge-search numbers or fast search numbers. Using these relations, we prove that the problem of deciding, given a graph G and an integer k, whether the fast (edge-)search number of G is less than or equal to k is NP-complete; and it remains NP-complete for Eulerian graphs. We also prove that the problem of determining whether the fast (edge-)search number of G is half of the number of odd vertices in G is NP-complete; and it remains NP-complete for planar graphs with maximum degree 4. We present a linear time approximation algorithm for the fast edge-search time that always delivers solutions of at most $(1 + \frac{|V|-1}{|E|+1})$ times the optimal value. This algorithm also gives us a tight upper bound on the fast search number of graphs. We also show a lower bound on the fast search number using the minimum degree and the number of odd vertices.

1. Introduction

Given a graph in which a fugitive hides on vertices or along edges, graph searching problems are usually to find the minimum number of searchers required to capture the fugitive. The edge searching problem and the node searching problem are two major graph searching problems. The edge searching problem was introduced by Megiddo et al. [13]. They showed that determining the edge search number of a graph is NP-hard. They also gave a linear time algorithm to compute the edge search number of a tree. The node searching problem was introduced by Kirousis and Papadimitriou [9]. They showed that the node search number is equal to the pathwidth plus one and that the edge search number and node search number differ by at most one. Both searching problems are monotonic [4,10].

Let $G = (V, E)$ be a graph with vertex set V and edge set E. In the edge search model, initially, G contains no searchers but G contains one fugitive who hides on vertices or along edges. The fugitive is invisible to searchers, and he can move at a great speed at any time from one vertex to another vertex along a searcher-free path between the two vertices. There are three types of actions for searchers in each step, i.e., *placing* a searcher on a vertex, *removing* a searcher from a vertex, and *sliding* a

E-mail address: boting@cs.uregina.ca.

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.12.065

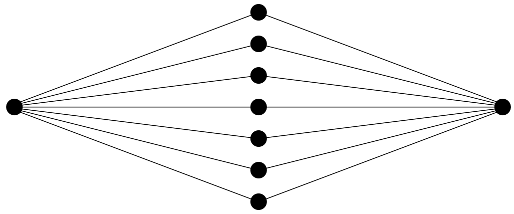

Fig. 1. The fast search number is 3 while the brush number is 7.

searcher along an edge from one endpoint to the other. An edge is cleared only by a sliding action. An edge the fugitive could be on is said to be *contaminated*, and an edge the fugitive cannot be on is said to be *cleared*. A contaminated edge uv can be cleared in one of two ways by one sliding action: (1) sliding a searcher from u to v along uv while at least one searcher is located on u, and (2) sliding a searcher from u to v along uv while all edges incident on u except uv are already cleared. An *edge search strategy* in a k-step search is a sequence of k actions such that the final action leaves all edges of G cleared. The graph G is *cleared* if all edges are cleared. The minimum number of searchers required to clear G in the edge search model is the *edge search number* of G, denoted by $\mathrm{es}(G)$. In this paper, we introduce a new searching problem in the edge search model, called *fast edge searching*, which is the problem of finding the minimum number of steps (or equivalently, actions) to clear G in the edge search model. In the fast edge searching problem, the minimum number of steps required to clear G is the *fast edge-search time* of G, denoted by $\mathrm{fet}(G)$, and the minimum number of searchers required so that G can be cleared in $\mathrm{fet}(G)$ steps is the *fast edge-search number* of G, denoted by $\mathrm{fen}(G)$. A fast edge-search strategy that uses $\mathrm{fet}(G)$ steps to clear G is called an *optimal fast edge-search strategy*.

The motivation to consider the fast edge searching problem is that, in some real-life scenarios, the cost of a searcher may be relatively low in comparison to the cost of allowing a fugitive to be free for a long period of time. For example, if a dangerous fugitive hiding along streets in an area, policemen always want to capture the fugitive as soon as possible.

The fast edge searching problem has a strong connection with the fast searching problem, which was first introduced by Dyer et al. [7]. The fast search model has the same setting as the edge search model except that every edge is traversed exactly once by a searcher and searchers cannot be removed. The minimum number of searchers required to clear G in the fast search model is the *fast search number* of G, denoted by $\mathrm{fsn}(G)$. A *fast search strategy* in a k-step fast search is a sequence of k actions such that the final action leaves all edges of G cleared. Notice that this definition is slightly different from the one used in [7].[1] A fast search strategy that uses $\mathrm{fsn}(G)$ searchers to clear G is called an *optimal fast search strategy*.

Note that the goal of the fast edge searching problem is to find the minimum number of steps to capture the fugitive in the edge search model, while the goal of the fast searching problem is to find the minimum number of searchers to capture the fugitive in the fast search model.

The fast searching problem has a close relation with the graph brushing problem [1,12] and the balanced vertex-ordering problem [3]. For any graph, the brush number is equal to the total imbalance of an optimal vertex-ordering. For some graphs, such as trees, the fast search number is equal to the brush number. But for some other graphs, the gap between the fast search number and the brush number can be arbitrarily large. For example, for a complete graph K_n with n ($n \geq 4$) vertices, the fast search number is n, and the brush number is $n^2/4$ if n is even, and $(n^2 - 1)/4$ otherwise. The difference is caused mainly by the different behavior of searchers and brushes. In the fast search problem, a searcher can go through an occupied vertex to clear two incident edges, but this is not allowed for brushes in the brushing problem. This difference can be illustrated by the graph H with k ($k \geq 4$) parallel paths sharing the same ends (see Fig. 1). The fast search number of H is 3 and the brush number is k.

Bonato et al. [5] introduced the capture time on cop-win graphs in the Cops and Robber game. While the capture time of a cop-win graph on n vertices is bounded above by $n - 3$, half the number of vertices is sufficient for a large class of graphs including chordal graphs.

In Section 2, we give definitions and notation. In Section 3, we establish relations between the fast edge searching and the fast searching. We also show that the family of graphs whose fast edge-search time is at most k is minor-closed for any positive integer k. In Section 4, we first establish relations between the fast (fast edge) searching and the node searching. By these relations, the problem of computing the node search number of a graph is equivalent to that of computing the fast (edge-)search number of a related graph. We then prove that the problem of deciding, given a graph G and an integer k, whether the fast (edge-)search number of G is less than or equal to k is NP-complete; and it remains NP-complete for Eulerian graphs. Since the family of graphs $\{G : \mathrm{fsn}(G) \leq k\}$ is not minor-closed for any positive integer $k \geq 2$ [7], we cannot obtain an upper bound on the fast search number using the fast search number of complete graphs. In Section 5, we present a linear time approximation algorithm. Using this algorithm, we show that the number of vertices in a graph is an upper bound on the fast search number of the graph. The lower bounds given in [14] are basically based on the number of odd vertices in the graph. In Section 6, we show a new lower bound based on the minimum degree and the number of odd vertices. In Section 7, we first show the fast search number of the graph of a family of functions is equal to the number of

[1] In [7], a fast search strategy for graph G is a sequence of $|E(G)|$ sliding actions that clear G.

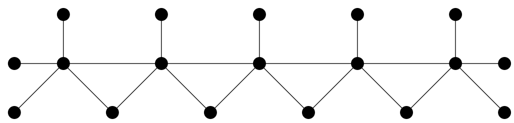

Fig. 2. A graph H with $3n+6$ vertices and $4n+5$ edges, where $n=4$.

functions in the family. We then prove that the problem of deciding, given a graph H, whether the fast search number of H is a half of the number of odd vertices in H is NP-complete; and it remains NP-complete for planar graphs with maximum degree 4. Finally, we conclude this paper in Section 8.

2. Preliminaries

Throughout this paper, all graphs and multigraphs have no loops. We use $G=(V,E)$ to denote a graph with vertex set V and edge set E, and we also use $V(G)$ and $E(G)$ to denote the vertex set and edge set of G respectively. We use uv to denote an edge with endpoints u and v. Definitions omitted here can be found in [15].

For a graph $G=(V,E)$, the *degree* of a vertex $v \in V$, denoted by $\deg_G(v)$, is the number of edges incident on v. A vertex is *odd* when its degree is odd. Similarly, a vertex is *even* when its degree is even. Let $V_{\text{odd}}(G)$ be the set of all odd vertices in G, and $V_{\text{even}}(G) = V \setminus V_{\text{odd}}(G)$. For a vertex $v \in V$, the set $\{u : uv \in E\}$ is the *neighborhood* of v, denoted as $N_G(v)$. In the case with no ambiguity, we use $\deg(v)$ and $N(v)$ without subscripts. Let $\delta(G) = \min\{|N(v)| : v \in V(G)\}$. For a subset $V' \subseteq V$, $G[V']$ denotes the subgraph induced by V', and for a subset $E' \subseteq E$, $G[E']$ denotes the subgraph formed by E'.

A *component* of a graph G is a maximal connected subgraph of G. A *cut-edge* or *cut-vertex* of a graph is an edge or vertex whose deletion increases the number of components. A *block* of a graph G is a maximal connected subgraph of G that has no cut-vertex. If G itself is connected and has no cut-vertex, then G is a block. It is easy to see that an edge of G is a block if and only if it is a cut-edge. If a block has at least 3 vertices, then it is 2-connected. Thus, the blocks of a graph are its isolated vertices, its cut-edges, and its maximal 2-connected subgraphs. The *block graph* of G is a graph T in which each vertex represents a block of G and two vertices are connected by an edge of T if the two corresponding blocks share a vertex of G. The block graph must be a tree if G is connected. A block of G that corresponds to a degree-one vertex of T is called a *leaf block*. Note that every leaf block has exactly one cut-vertex.

A *path* is a list $v_0, e_1, v_1, \ldots, e_k, v_k$ of vertices and edges such that each edge e_i, $1 \le i \le k$, has endpoints v_{i-1} and v_i and each vertex appears exactly once (except that its first vertex might be the same as its last). Thus we can denote a path by a list of vertices $v_0v_1 \ldots v_k$. A *cycle* is a path that begins and ends on the same vertex.

We say that a vertex in G is *occupied* at some moment if at least one searcher is located on this vertex at this moment.

3. Fast edge searching vs. fast searching

In this section, we consider the relationship between the fast edge searching in the edge search model and the fast searching in the fast search model.

Theorem 3.1. *For any graph* $G=(V,E)$, $\text{fet}(G) = \text{fsn}(G) + |E|$.

Proof. Note that a fast search strategy can be considered as an edge-search strategy. Since a fast search strategy consists of $\text{fsn}(G)$ placing actions and $|E|$ sliding actions, we have $\text{fet}(G) \le \text{fsn}(G) + |E|$. Recall that the fast edge-search time is the minimum number of actions needed to clear G in the edge search model. If an edge-search strategy of G containing removing actions, we can delete all removing actions and the remaining actions still form a valid edge-search strategy with fewer actions (and possibly more searchers). If an edge-search strategy of G containing sliding actions that slide a searcher from u to v along a cleared path between them, we can replace these actions by placing a searcher on vertex v. The resulted strategy is an edge-search strategy that may contain fewer actions (and again possibly more searchers). So, in an optimal fast edge-search strategy, removing actions are not contained, and traversing along a cleared path is also not necessary. Thus, we can always convert a fast edge-search strategy to a fast search strategy. Hence, $\text{fsn}(G) \le \text{fet}(G) - |E|$, which completes the proof. □

Corollary 3.2. *For any graph* G, $\text{fen}(G) \le \text{fsn}(G)$.

The difference between $\text{fen}(G)$ and $\text{fsn}(G)$ can be large. As illustrated in Fig. 2, let H be a graph with $3n+6$ $(n>1)$ vertices and $4n+5$ edges. Note that the fast search number of a graph is at least half of the number of odd vertices in the graph [7]. Since H has $2n+6$ odd vertices, we know that $\text{fsn}(H) \ge n+3$. In fact, we can clear H using $n+3$ searchers by a fast search strategy. But we can clear H using 2 searchers by a fast edge-search strategy. Thus the ratio $\text{fsn}(H)/\text{fen}(H) = (n+3)/2$ can be arbitrarily large. We have the following relation between the fast edge-search number and the fast search number.

Theorem 3.3. *For any graph G, let $\widehat{G}$ be a graph obtained from G by replacing each edge of G by a path of length 2. Then,* $\text{fsn}(G) = \text{fen}(\widehat{G})$.

Proof. Because every optimal fast search strategy of G can be converted to a fast edge-search strategy of $\widehat{G}$, we have $\mathrm{fen}(\widehat{G}) \leq \mathrm{fsn}(G)$. We now show that $\mathrm{fsn}(G) \leq \mathrm{fen}(\widehat{G})$. Let S be an optimal fast edge-search strategy of $\widehat{G}$. For any edge $uv \in E(G)$ and its corresponding path $uu'v$ in $\widehat{G}$, the following two cases cannot happen in S: (1) if a searcher slides from u to v along $uu'v$ and then back from v to u along $vu'u$, then these 4 sliding actions can be replaced by 3 actions, that is, placing a searcher on v and sliding the searcher from v to u along $vu'u$; and (2) if a searcher slides from u to u' and another searcher slides from v to u' and then one searcher slides from u' to v and the other slides from u' to v, then these 4 sliding actions can be replaced by 3 actions as in case (1). Because the above two cases cannot happen in S, we can easily convert S to a fast search strategy of G. Thus $\mathrm{fsn}(G) \leq \mathrm{fen}(\widehat{G})$, which completes the proof. □

Corollary 3.4. *Let G be a graph such that for every vertex v with $\deg(v) \neq 2$, all neighbors of v have degree 2. Then, $\mathrm{fsn}(G) = \mathrm{fen}(G)$.*

From [7], we know that the family of graphs $\{G : \mathrm{fsn}(G) \leq k\}$ is not minor-closed for any positive integer $k \geq 2$. For any integer ℓ, there exist graphs G and its subgraph H such that the ratio $\mathrm{fsn}(H)/\mathrm{fsn}(G) > \ell$. But for the fast edge searching, we can show that the family of graphs $\{G : \mathrm{fet}(G) \leq k\}$ is minor-closed.

Theorem 3.5. *Given a graph G, if H is a minor of G, then $\mathrm{fet}(H) \leq \mathrm{fet}(G)$.*

Proof. For a vertex $v \in V(H)$, let C_v be a subset of vertices from $V(G)$ such that v is obtained from G by identifying the vertices of C_v under contraction. Given a fast edge-search strategy of G, we convert it to a search strategy of H by the following rules: whenever a searcher is placed on $v' \in V(G)$, the corresponding searcher in the new search strategy is placed on $v \in V(H)$ if there is a C_v such that $v' \in C_v$, but does nothing otherwise; and whenever a searcher slides along an edge $u'v'$ from $u' \in V(G)$ to $v' \in V(G)$, the new search strategy does one of the following actions: (1) a searcher slides along edge uv from $u \in V(H)$ to $v \in V(H)$ if there are two sets C_u and C_v such that $u' \in C_u$, $v' \in C_v$, and $uv \in E(H)$; (2) a new searcher is placed on $v \in V(H)$ if there are two sets C_u and C_v such that $u' \in C_u$ and $v' \in C_v$, but $uv \notin E(H)$; (3) a new searcher is placed on $v \in V(H)$ if there is a C_v such that $v' \in C_v$, but there is no C_u such that $u' \in C_u$; and (4) does nothing otherwise. Note that all optimal fast edge-search strategies do not contain any removing actions.

It is easy to verify that the new strategy can clear H using at most $\mathrm{fet}(G)$ steps. Thus, $\mathrm{fet}(H) \leq \mathrm{fet}(G)$. □

4. Node searching vs. fast (edge) searching

In this section, we establish relations between the node search number and the fast (edge-)search number. Using these relations, we can prove that both fast edge search problem and fast search problem are NP-hard. In the node search model [9], there are only two types of actions for searchers: placing and removing. An edge is cleared if both endpoints are occupied by searchers. We use $place_X(u)$ to denote the action of placing a searcher on vertex u in the strategy X, and use $remove_X(u)$ to denote the action of removing a searcher from vertex u in the strategy X. For a graph G, the minimum number of searchers needed to clear G in the node search model is the *node search number* of G, denoted by $\mathrm{ns}(G)$. In the fast search model, we use $place_Y(u)$ to denote the action of placing a searcher on vertex u in the strategy Y, and use $slide_Y(u, v)$ to denote the action of sliding a searcher from u to v along edge uv in the strategy Y. In the case with no ambiguity, we use $place(u)$, $remove(u)$ and $slide(u, v)$ without subscripts.

For a path P of length at least 1, we know that $\mathrm{ns}(P) = 2$ and $\mathrm{fsn}(P) = \mathrm{fen}(P) = 1$. For a cycle C of length at least 3, we know that $\mathrm{ns}(C) = 3$ and $\mathrm{fsn}(C) = \mathrm{fen}(C) = 2$. For any graph G, it is easy to see that $\mathrm{ns}(G) \leq \mathrm{fsn}(G) + 1$ and $\mathrm{ns}(G) \leq \mathrm{fen}(G) + 1$. The gap between the node search number and the fast (edge-)search number can be arbitrarily large for some graphs. For example, for a complete bipartite graph $K_{1,n}$ with bipartitions of size 1 and n, we have $\mathrm{fsn}(K_{1,n}) = \mathrm{fen}(K_{1,n}) = \lceil \frac{n}{2} \rceil$ whereas $\mathrm{ns}(K_{1,n}) = 2$.

From [9], we know that node search strategies can be standardized as follows.

Lemma 4.1 ([9]). *For any graph G, there always exists a monotonic node search strategy satisfying the following conditions:*

(i) *it clears G using $\mathrm{ns}(G)$ searchers;*
(ii) *every vertex is visited exactly once by one searcher;*
(iii) *every searcher is removed immediately after all the edges incident on it have been cleared (ties are broken arbitrarily); and*
(iv) *a searcher is removed from a vertex only when all the edges incident on it are cleared.*

An optimal node search strategy satisfying the properties in Lemma 4.1 is called a *standard node search strategy*. For a graph with n vertices, any standard node search strategy is monotonic and has $2n$ actions. It is easy to see the first action is placing and the last action is removing.

For a graph G, let G' be a graph obtained from G by adding a vertex a and connecting it to each vertex of G. Let A'_G be a multigraph obtained from G' by replacing each edge with 4 parallel edges. Let A_G be a graph obtained from A'_G by replacing each edge of A'_G with a path of length 2. In graphs G', A'_G and A_G, the vertex a is called *apex*. It is easy to see that $\mathrm{fsn}(A'_G) = \mathrm{fsn}(A_G)$.

Lemma 4.2. *For a complete graph K_n with $n \geq 2$, $\mathrm{fsn}(A_{K_n}) = \mathrm{fen}(A_{K_n}) = n + 2$.*

Proof. Since $\mathrm{fsn}(A'_{K_n}) = \mathrm{fsn}(A_{K_n})$, we will show that $\mathrm{fsn}(A'_{K_n}) = n+2$. We first show that $\mathrm{fsn}(A'_{K_n}) \le n+2$. We place n searchers on each vertex of K_n and 2 searchers on the apex a. Since A'_{K_n} is an Eulerian graph, we can slide one searcher on a along every edge of A'_{K_n} exactly once. Thus $\mathrm{fsn}(A'_{K_n}) \le n+2$.

We now show that $\mathrm{fsn}(A'_{K_n}) \ge n+2$. Note that A'_{K_n} can be obtained from K_{n+1} by replacing each edge of K_{n+1} with 4 parallel edges. Let S be an optimal fast search strategy of A'_{K_n}, v be the first cleared vertex, and uv be the second last cleared edge incident on v. Just after uv is cleared, there is only one dirty edge incident on v. There are two cases to clear uv.

1. If uv is cleared by sliding a searcher from u to v, then v must be occupied by at least two searchers, and every other vertex of A'_{K_n} is occupied by at least one searcher.
2. If uv is cleared by sliding a searcher from v to u, then u is occupied by at least two searchers, and every other vertex of A'_{K_n} is occupied by at least one searcher.

From the above cases, we have $\mathrm{fsn}(A'_{K_n}) \ge n+2$. Thus, $\mathrm{fsn}(A_{K_n}) = \mathrm{fsn}(A'_{K_n}) = n+2$. It follows from Corollary 3.4 that $\mathrm{fen}(A_{K_n}) = \mathrm{fsn}(A_{K_n}) = n+2$. □

We have the following relation between the node search number of G and the fast search number of A_G.

Lemma 4.3. *For a graph G and its corresponding graph A_G described above,* $\mathrm{fsn}(A_G) \le \mathrm{ns}(G) + 2$.

Proof. Since $\mathrm{fsn}(A'_G) = \mathrm{fsn}(A_G)$, we will show that $\mathrm{fsn}(A'_G) \le \mathrm{ns}(G) + 2$. Let $\mathrm{ns}(G) = k$ and $X = (X_1, \dots, X_{2n})$ be a standard node search strategy, where n is the number of vertices in G. Each X_i is one of the two actions: placing and removing. There is no searcher on G before X_1 and X_1 is a placing-action. Let $E_i(X)$, $1 \le i \le 2n$, be the set of cleared edges just after X_i and $E_0(X)$ be the set of cleared edges just before X_1. We will show that $\mathrm{fsn}(A'_G) \le k+2$ by constructing a fast search strategy Y that uses $k+2$ searchers to clear A'_G. For each action X_i, $1 \le i \le 2n$, we use a sequence of actions, denoted as $y(X_i)$, to simulate the action X_i. So Y is the concatenation of all $y(X_i)$ and can be expressed as $(y_0, y(X_1), \dots, y(X_{2n}))$, where y_0 is a sequence of $k+2$ actions that place $k+2$ searchers on the apex a and each $y(X_i)$, $1 \le i \le 2n$, is a sequence of sliding actions. Let $E_i(Y)$ be the set of all cleared edges by strategy Y just after $y(X_i)$ and $E_0(Y)$ be the set of cleared edges just after y_0. Note that $E_i(Y)$ is not a multiset, that is, a multiple edge pq appears in $E_i(Y)$ only when all parallel edges between p and q are cleared. Let E_a be a set of all edges incident on a.

We now construct Y from X inductively such that $E_i(X) = E_i(Y) \setminus E_a$ for each i satisfying $1 \le i \le 2n$. It is easy to see that $E_0(X) = E_0(Y) = \emptyset$. Initially, if the action X_1 is $\mathit{place}_X(u)$, then let $y(X_1) = (\mathit{slide}_Y(a, u))$. Thus, $E_1(X) = E_1(Y) \setminus E_a = \emptyset$.

Suppose that $E_{j-1}(X) = E_{j-1}(Y) \setminus E_a$ and the set of vertices in G occupied by searchers just after X_{j-1} is equal to the set of vertices in $A'_G - a$ occupied by searchers just after the last action of $y(X_{j-1})$. We now consider $E_j(X)$ and $E_j(Y) \setminus E_a$. There are two cases regarding the action X_j.

Case 1. $X_j = \mathit{place}_X(v)$. If $E_j(X) \setminus E_{j-1}(X) = \emptyset$, then no edge is cleared by X_j, and no recontamination happens. Thus we set $y(X_j) = (\mathit{slide}_Y(a, v))$. It is easy to see that $E_j(X) = E_{j-1}(X) = E_{j-1}(Y) \setminus E_a = E_j(Y) \setminus E_a$. If $E_j(X) \setminus E_{j-1}(X) \ne \emptyset$, the graph G_j formed by the edges of $E_j(X) \setminus E_{j-1}(X)$ is a star with the center v. It is easy to see that each vertex of $G_j - v$ is occupied by a searcher just before X_j. Let $V(G_j - v) = \{u_1, u_2, \dots, u_m\}$. We can construct $y(X_j) = ((\mathit{slide}_Y(a, v))^2, (\mathit{slide}_Y(v, u_1), \mathit{slide}_Y(u_1, v))^2, \dots, (\mathit{slide}_Y(v, u_m), \mathit{slide}_Y(u_m, v))^2, (\mathit{slide}_Y(v, a))^2)$, where $(\mathit{slide}_Y(a, v))^2$ means that the action $\mathit{slide}_Y(a, v)$ contiguously appears two times to clear two parallel edges between a and v, and $(\mathit{slide}_Y(v, u_1), \mathit{slide}_Y(u_1, v))^2$ means that a pair of actions $(\mathit{slide}_Y(v, u_1), \mathit{slide}_Y(u_1, v))$ contiguously appears two times to clear four parallel edges between v and u_1. Since X is a standard node search strategy, the apex a is occupied by at least three searchers just before two of them moves from a to v in the first two actions of $y(X_i)$. Thus $E_j(X) \setminus E_{j-1}(X) = (E_j(Y) \setminus E_a) \setminus (E_{j-1}(Y) \setminus E_a)$. It follows from the inductive hypothesis that $E_j(X) = E_j(Y) \setminus E_a$ and the set of vertices in G occupied by searchers just after X_j is equal to the set of vertices in $A'_G - a$ occupied by searchers just after the last action of $y(X_j)$.

Case 2. $X_j = \mathit{remove}_X(v)$. We set $y(X_j) = (\mathit{slide}_Y(a, v), (\mathit{slide}_Y(v, a))^2)$. Since no edge can be cleared by a removing-action in X, we have $E_j(X) \setminus E_{j-1}(X) = (E_j(Y) \setminus E_a) \setminus (E_{j-1}(Y) \setminus E_a) = \emptyset$. Since X is a standard node search strategy, each edge incident on v is cleared just before X_{j-1}. Thus, just before the first action of $y(X_i)$, each edge in $A'_G - a$ incident on v is cleared and the apex a is occupied by at least two searchers and one of them moves from a to v in the first action of $y(X_i)$. From the inductive hypothesis, we know that $E_j(X) = E_j(Y) \setminus E_a$ and the set of vertices in G occupied by searchers just after X_j is equal to the set of vertices in $A'_G - a$ occupied by searchers just after the last action of $y(X_j)$. □

Lemma 4.4. *For a graph G, let G' be a graph obtained from G by adding a vertex a and connecting it to each vertex of G. Then* $\mathrm{ns}(G') = \mathrm{ns}(G) + 1$.

Proof. For G', we first place one searcher on a, and then use an optimal node search strategy of G to clear G'. Thus, $\mathrm{ns}(G') \le \mathrm{ns}(G) + 1$.

We now show that $\mathrm{ns}(G) \le \mathrm{ns}(G') - 1$. Let S be a standard node search strategy of G'. Since a is adjacent to all other vertices in G', it follows from Lemma 4.1(iv) that no searcher is removed from any vertex before a searcher is placed on a and no searcher is placed on any vertex after the searcher on a is removed. Suppose that there is a moment t at which $G' - a$ contains $\mathrm{ns}(G')$ searchers. Note that the last action before t must be placing a searcher on a vertex in $G' - a$. Since no searcher is placed on any vertex after the searcher on a is removed, it follows from Lemma 4.1(ii) that a has not been occupied before t. Thus, all edges incident on a are dirty. Note that the first action after t must be removing a searcher from a vertex in $G' - a$. From Lemma 4.1(iv), all the edges incident on this vertex are cleared. This is a contradiction. Thus, at any moment when G' contains $\mathrm{ns}(G')$ searchers, there is a searcher on a. Let S' be a strategy obtained from S by deleting the actions *place*(a) and *remove*(a). Then S' is a monotonic node search strategy that can clear the graph $G' - a$ (i.e., G) using $\mathrm{ns}(G') - 1$ searchers. Hence, $\mathrm{ns}(G) \le \mathrm{ns}(G') - 1$. Therefore, $\mathrm{ns}(G') = \mathrm{ns}(G) + 1$. □

Lemma 4.5. *For a graph G and its corresponding graph A_G,* $\mathrm{ns}(G) \le \mathrm{fsn}(A_G) - 2$.

Proof. It follows from Lemma 4.4 that $\mathrm{ns}(G) = \mathrm{ns}(G') - 1$. Since $\mathrm{ns}(G') = \mathrm{ns}(A'_G)$ and $\mathrm{fsn}(A'_G) = \mathrm{fsn}(A_G)$, we only need to show that $\mathrm{ns}(A'_G) \le \mathrm{fsn}(A'_G) - 1$.

Let $S = (S_0, s_1, \ldots, s_m)$ be an optimal fast search strategy of A'_G that clears A'_G using k searchers, where S_0 is a sequence of k placing actions. We can construct a monotonic node search strategy T by modifying S in the following way. For each action s_i ($i \ge 1$) that slides a searcher from u to v, if u is occupied by only one searcher just before sliding, then we delete this action; otherwise, we replace s_i by the actions *remove*(u) and *place*(v).

For any multiple edge between two vertices u and v, when a searcher slides the second time from one endpoint to the other by S, both u and v must be occupied by searchers. Thus, all four parallel edges are cleared by T. Hence, T is a monotonic node search strategy that clears A'_G using k searchers.

We now show that we can modify T to obtain a monotonic node search strategy that clears A'_G using $k - 1$ searchers. Note that some actions in T may be redundant, that is, placing a searcher on an occupied vertex. For any multiple edge uv, when a searcher slides the first time from one endpoint to the other by S, all four parallel edges are cleared by the actions *remove*(u) and *place*(v) in T. Thus, any moment when S requires a searcher to slide along the second parallel edge between u and v, T does not need such a searcher. Therefore, we can delete redundant actions from T to obtain a monotonic node search strategy that clears A'_G using $k - 1$ searchers. □

From Lemmas 4.3, 4.5 and Corollary 3.4, we have the main result of this section.

Theorem 4.6. *For a graph G and its corresponding graph A_G,* $\mathrm{ns}(G) = \mathrm{fsn}(A_G) - 2 = \mathrm{fen}(A_G) - 2$.

For a graph G, let $\mathrm{pw}(G)$ be the pathwidth of G and G'' be the graph obtained from G by replacing each edge of G with a path of length 3. Since $\mathrm{pw}(G) = \mathrm{ns}(G) - 1$, we have $\mathrm{pw}(G) = \mathrm{fsn}(A_G) - 3$. And since $\mathrm{es}(G) = \mathrm{pw}(G'')$, we have $\mathrm{es}(G) = \mathrm{fsn}(A_{G''}) - 3$.

Given a graph G and an integer k, the fast search (fast edge search) problem is to determine whether G can be cleared by k searchers in the fast search (fast edge search) model. Then we have the following result.[2]

Corollary 4.7. *The fast search problem and the fast edge search problem are NP-complete. They remain NP-complete for Eulerian graphs.*

Since the node search problem is NP-complete for cubic graphs [11], we can strength the above theorem as follows.

Corollary 4.8. *The fast search problem and the fast edge search problem are NP-complete for multigraphs that are 12-regular multigraphs after deleting one vertex.*

From Theorem 3.1, we can also show that, given a graph G and an integer k, it is NP-complete to determine whether $\mathrm{fet}(G) \le k$. It remains NP-complete for Eulerian graphs.

5. Approximation algorithm

Since the family of graphs $\{G : \mathrm{fsn}(G) \le k\}$ is not minor-closed for any positive integer $k \ge 2$ [7], we cannot obtain an upper bound on the fast search number using the fast search number of complete graphs. In this section, we present a linear time algorithm that can compute a fast search strategy for any connected graph $G = (V, E)$, which is also a fast edge-search strategy because any fast search strategy is also a fast edge-search strategy. We can use this algorithm to show that the number of vertices in a graph is an upper bound on the fast search number of the graph. Since the fast search number of a complete graph K_n ($n \ge 4$) is n, we know this upper bound is tight. Using this algorithm, we can also compute a fast edge-search strategy of G whose length (i.e., the number of actions) is at most $(1 + \frac{|V|-1}{|E|+1})$ times the fast edge-search time of G.

If G is not connected, the fast search number of G is the sum of fast search numbers of all components. So we only consider connected graphs. The input of the algorithm is a connected graph G with at least 4 vertices. The output of the algorithm is a fast (edge-)search strategy $\langle V_p, A_s \rangle$, where V_p is a multiset of vertices on which we place searchers and A_s is a sequence of arcs corresponding to sliding actions, that is, an arc (u, v) corresponds to sliding along the arc from tail u to head v. Given two vertices u and v in G, the distance between them, denoted by $\mathrm{dist}_G(u, v)$, is the number of edges on the shortest path between them.

Algorithm FastSearch(G)
Input: A connected graph $G = (V, E)$ with at least 4 vertices.
Output: A fast (edge-)search strategy $\langle V_p, A_s \rangle$ of G.

[2] Dereniowski et al. [6] independently proved the fast search problem is NP-complete by a “weak search” approach that is different from our method.

1. Compute a block graph T of G.
2. Arbitrarily pick a leaf t of T. Let B be a block of G corresponding to t and a be a vertex of B which is not a cut-vertex of G. Call FastSearchBlock(B, a).
3. Update T by deleting the leaf t, and update G by deleting all vertices of B except the vertex that is a cut-vertex of G and is incident with a dirty edge of G. If G contains no edges, then stop and output the multiset of vertices V_p on which searchers are placed and output the sequence of arcs A_s in the order when searchers slide along them from tail to head; otherwise, go to step 2.

Algorithm FastSearchBlock(B, a)

1. $B' \leftarrow B - a$, $H \leftarrow B'$, and $\mathcal{P} \leftarrow \emptyset$.
2. If $V_{\text{odd}}(B') = \emptyset$, then place searchers on a if necessary so that a is occupied by at least $\deg_B(a)$ searchers. Slide searchers from a to every vertex in $N_B(a)$ to clear a. If $V(B')$ contains only one vertex, then return to FastSearch; otherwise, place a searcher on each unoccupied vertex in $V(B')$. If there is a vertex occupied by at least two searchers, then slide one of them along all edges of B'; otherwise, place a searcher on an arbitrary vertex of B' and slide it along all edges of B'. Return to FastSearch.
3. Arbitrarily pick a vertex $u \in V_{\text{odd}}(H)$ and find a vertex $v \in V_{\text{odd}}(H)$ such that $\text{dist}_H(u, v) = \min\{\text{dist}_H(u, w) : w \in V_{\text{odd}}(H) \text{ and } w \neq u\}$. Let P_{uv} be the shortest path between u and v. Update $\mathcal{P} \leftarrow \mathcal{P} \cup \{P_{uv}\}$ and $H \leftarrow H - E(P_{uv})$. If $V_{\text{odd}}(H) \neq \emptyset$, repeat Step 3.
4. If H has only one component, then place searchers on a if necessary so that a is occupied by at least $|V(H) \cap N_B(a)|$ searchers. Slide $|V(H) \cap N_B(a)|$ searchers from a to every vertex in $V(H) \cap N_B(a)$. Place a searcher on each unoccupied vertex of H. If a vertex of H is occupied by at least two searchers, then slide one of them along all edges of H; otherwise, place a searcher on a vertex of H and slide it along all edges of H to clear H. For each path in $\mathcal{P}$, we slide the searcher from one end of the path to the other. Return to FastSearch.
5. Let h be the number of components in H. Construct a graph H' such that each vertex v of $V(H')$ represents a component H_v of H and two vertices u and v are connected by an edge of H' if there is a path in $\mathcal{P}$ which contains a vertex of the component H_u corresponding to u and contains a vertex of the component H_v corresponding to v, and the subpath between u and v does not contain any vertex of other components (different from H_u and H_v). Assign a direction to each path in $\mathcal{P}$ such that each path in $\mathcal{P}$ becomes a directed path and H' becomes an acyclic graph. Let $H_1, H_2, \ldots, H_h$ be a sequence of all components in H such that the corresponding sequence of all vertices of H' forms an acyclic ordering. Set $i \leftarrow 1$.
6. If $V(H_i) \cap N_B(a) \neq \emptyset$, then go to Step 9.
7. If H_i contains a single vertex, then slide all searchers on this vertex along untraversed edges to the other endpoints complying with the direction of edges. $i \leftarrow i + 1$ and go to Step 6.
8. Place a searcher on each unoccupied vertex of H_i. If a vertex of H_i is occupied by at least two searchers, then slide one of them along all edges of H_i; otherwise, place a searcher on a vertex of H_i and slide it along all edges of H_i to clear H_i. Go to Step 11.
9. If H_i contains more than one vertex, then go to Step 10. Let x be the unique vertex in H_i. Place searchers on a if it is occupied by less than two searchers so that a is occupied by two searchers. Slide a searcher from a to x. Slide all searchers on x along untraversed edges to the other endpoints complying with the direction of edges. If $i = h$, then return to FastSearch; otherwise, $i \leftarrow i + 1$ and go to Step 6.
10. If $i < h$ and a is occupied by less than $|V(H_i) \cap N_B(a)| + 1$ searchers, then place searchers on a so that a is occupied by $|V(H_i) \cap N_B(a)| + 1$ searchers. If $i = h$ and a is occupied by less than $|V(H_i) \cap N_B(a)|$ searchers, then place searchers on a so that a is occupied by $|V(H_i) \cap N_B(a)|$ searchers. Slide $|V(H_i) \cap N_B(a)|$ searchers from a to every vertex in $V(H_i) \cap N_B(a)$. Place a searcher on each unoccupied vertex of H_i. If a vertex of H_i is occupied by at least two searchers, then slide one of them along all edges of H_i; otherwise, place a searcher on a vertex of H_i and slide it along all edges of H_i to clear H_i.
11. For each pair of vertices $u, v \in V(H_i)$ satisfying that the shortest path P_{uv} between them is a subpath of a path in $\mathcal{P}$, we slide the searcher from one end of P_{uv} to the other complying with the direction of edges. If $i = h$, then return to FastSearch; otherwise, $i \leftarrow i + 1$ and go to Step 6.

Theorem 5.1. *For any connected graph $G = (V, E)$, Algorithm* FastSearch(G) *outputs a fast search strategy that clears G using at most $|V|$ searchers in the fast search model.*

Proof. In FastSearch(G), we first decompose G into blocks. Then we choose a leaf block B, clear B and leave one searcher on the vertex of B which is a cut-vertex of G. Since the block graph of G is a tree, we can repeat this process until G is cleared. If each leaf block B can be cleared using at most $|V(B)|$ searchers, then G can be cleared using $|V|$ searchers.

We now consider how to clear a leaf block B using at most $|V(B)|$ searchers such that the cut-vertex of G in B is occupied by at least one searcher when B is cleared. Note that B has only one cut-vertex of the current G since B is a leaf block in G. If B is an edge uv, where u is a leaf of the current G then uv can be cleared by sliding a searcher from u to v. Thus, B (i.e., uv) can be cleared using one searcher such that the cut-vertex v of G in B is occupied by one searcher when B is cleared.

Suppose that B contains at least three vertices. Pick a vertex a of B which is not a cut-vertex of the current G. Let $B' = B - a$. Then B' is connected since B is a block. We have two cases on $V_{\text{odd}}(B')$.

CASE 1. $V_{\text{odd}}(B') = \emptyset$. Then B' is an Eulerian graph. Clear a by sliding searchers from a to every vertex in $N_B(a)$. Place a searcher on each unoccupied vertex in $V(B')$. If there is a vertex occupied by at least two searchers, then slide one of them along all edges of B', and thus the total number of searchers used to clear B is at most $|V(B)|-1$; otherwise place an additional searcher on an arbitrary vertex of B' and slide it along all edges of B'. Thus, the total number of searchers used to clear B is at most $|V(B)|$.

CASE 2. $V_{\text{odd}}(B') \neq \emptyset$. Note that every graph has even number of odd vertices. Let u and v be two vertices in $V_{\text{odd}}(B')$ and P_{uv} be the shortest path between them. Since v is the closest vertex to u in $V_{\text{odd}}(B')$, we know that $V(P_{uv}) \cap V_{\text{odd}}(B') = \{u, v\}$. Let B'' be the graph obtained from B' by deleting all edge of P_{uv}. Note that both u and v have even degree in B''. Thus, $|V_{\text{odd}}(B'')| = |V_{\text{odd}}(B')| - 2$. We can repeat the above process until we obtain an even graph H and the set of all deleted shortest paths $\mathscr{P}$. If H has only one component, similar to CASE 1, we can clear H using at most $|V(H)|$ searchers. Since all end vertices of paths in $\mathscr{P}$ are different, we can clear each path of $\mathscr{P}$ by sliding a searcher from one endpoint to the other.

Suppose that H contains at least two components, i.e., $h \geq 2$. Since B' is connected, each component H_i, $1 \leq i \leq h$, in H must contain at least one vertex of a path in $\mathscr{P}$. We clear each $H_1, H_2, \ldots, H_h$ in the acyclic ordering of H'. If H_i contains a single vertex v, then v cannot be a leaf of B because B is a block containing at least 3 vertices. Note that v becomes a single vertex in H_i because we delete all edges of $\mathscr{P}$ from H. Thus at least one path in $\mathscr{P}$ contains v as an interior vertex, and furthermore, v cannot be the end vertex of a path in $\mathscr{P}$ because no path in $\mathscr{P}$ contains an odd vertex of H as an interior vertex. If $v \in N_B(a)$, then place searchers on a if necessary so that a is occupied by two searchers, and slide one searcher from a to v. Because the number of in-edges of v is equal to the number of out-edges of v and we clear $H_1, \ldots, H_h$ in the acyclic ordering of H', we can slide searchers from v along all untraversed edges to the other endpoints complying with the edge directions. Suppose that H_i contains at least two vertices. If $V(H_i) \cap N_B(a) \neq \emptyset$, then place searchers on a if necessary so that we can slide $|V(H_i) \cap N_B(a)|$ searchers from a to every vertex in $V(H_i) \cap N_B(a)$. We have two subcases.

CASE 2.1. $i < h$. In this case, we place a searcher on each unoccupied vertex of H_i, and place another searcher on a vertex of H_i and slide it along all edges of H_i to clear H_i. Since $i < h$, we have enough searchers to clear H_i and leave at most $|V(H_i)|$ searchers on vertices $V(H_i)$ when B is cleared.

CASE 2.2. $i = h$. Since $h > 1$, there is a vertex u of H_h that is an end vertex of a path in $\mathscr{P}$ and is occupied before we place searchers on H_h. We place a searcher on each unoccupied vertex of H_h, and place another searcher on the vertex u and slide it along all edges of H_h to clear H_h. Thus, we can use $|V(H_h)| + 1$ searchers to clear H_h and at least one searcher comes from another component.

For each pair of vertices $u, v \in V(H_i)$ satisfying that the shortest path P_{uv} between them is a subpath of a path in $\mathscr{P}$, we slide the searcher from one end of P_{uv} to the other complying with the edge directions. We clear $B'[V(H_i)]$ that is a subgraph of B' induced from $V(H_i)$ using at most $|V(H_i)|$ searchers.

From cases 2.1 and 2.2, B can be cleared using at most $|V(B)|$ searchers. Therefore, it follows from cases 1 and 2 that $\text{fsn}(G) \leq |V|$. □

Theorem 5.2. *Algorithm* FASTSEARCH*(G) can be implemented with linear time.*

Proof. Let n be the number of vertices and m be the number of edges in G. In step 1 of FASTSEARCH(G), it takes $O(n+m)$ time to compute a block graph T of G. For step 2 of FASTSEARCH(G), we first analyze the running time of FASTSEARCHBLOCK(B, a).

In FASTSEARCHBLOCK(B, a), steps 1 and 2 need $O(|V(B)| + |E(B)|)$ time. In step 3, using the breadth first search, it takes $O(|V(H)|+|E(H)|)$ time to compute $\text{dist}_H(u, v)$. Step 4 needs $O(|V(H)|+|E(H)|)$ time. In step 5, it takes $O(|V(H)|+|E(H)|)$ time to construct H' and compute the acyclic orientation of $\mathscr{P}$ and H'. Step 6 needs $O(1)$ time. Steps 7, 9 and 11 need $O(|V(\mathscr{P})| + |E(\mathscr{P})|)$ time. Steps 8 and 10 need $O(|V(H_i)| + |E(H_i)|)$ time. Since every edge is traversed once, the total running time of FASTSEARCHBLOCK(B, a) is $O(|V(B)| + |E(B)|)$.

Thus, step 2 of FASTSEARCH(G) needs $O(|V(B)| + |E(B)|)$ time. Step 3 also needs $O(|V(B)| + |E(B)|)$ time. Therefore, FASTSEARCH(G) can be implemented with $O(n + m)$ time. □

For any connected graph G, since each placing-action places a new searcher in the fast search model, $|V_p|$ is the number of searchers required by FASTSEARCH(G). Then G can be cleared in $|V_p| + |A_s|$ steps by FASTSEARCH(G). Since any fast search strategy is also an edge search strategy, G can be cleared in at most $|V_p| + |A_s|$ steps in the edge search model. Thus, FASTSEARCH(G) is also an approximation algorithm for the fast edge-search time with the following approximation ratio.

Theorem 5.3. *For any connected graph G with n vertices and m edges,*

$$\frac{|V_p| + |A_s|}{\text{fet}(G)} \leq \left(1 + \frac{n-1}{m+1}\right).$$

For odd graphs, the approximation ratio for the fast search number is 2, and for fast edge-search time is $1 + \frac{n}{n+2m}$.

Corollary 5.4. *For any connected odd graph G with n vertices and m edges,*

$$\frac{|V_p|}{\text{fsn}(G)} \leq 2 \quad \textit{and} \quad \frac{|V_p| + |A_s|}{\text{fet}(G)} \leq \left(1 + \frac{n}{n+2m}\right).$$

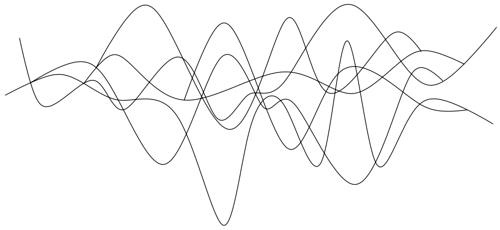

Fig. 3. The graph of a set of 6 functions.

6. Lower bound

In this section, we give a new lower bound that is related to both the number of odd vertices and the minimum degree.

Theorem 6.1. *For a connected graph G with $\delta(G) \geq 3$,*

$$\mathrm{fsn}(G) \geq \max\left\{\delta(G)+1, \left\lceil \frac{\delta(G)+|V_{\mathrm{odd}}(G)|-1}{2} \right\rceil\right\}.$$

Proof. Note that $\mathrm{fsn}(G) \geq \mathrm{es}(G)$ for any graph G. From Theorem 2.4 in [2], we know that $\mathrm{es}(G) \geq \delta(G)+1$ for any connected graph G with $\delta(G) \geq 3$. If $|V_{\mathrm{odd}}(G)| \leq \delta(G)+3$, then $\mathrm{fsn}(G) \geq \delta(G)+1 \geq \lceil \frac{\delta(G)+|V_{\mathrm{odd}}(G)|-1}{2} \rceil$, which completes the proof.

Suppose that $|V_{\mathrm{odd}}(G)| > \delta(G)+3$. Let S be an optimal fast search strategy of G such that searchers are placed on vertices only when it is necessary. Let v be the first vertex cleared by S. When v is cleared, each vertex in $N(v)$ must contain at least one searcher. Let $V' \subseteq V(G)$ be the set of occupied vertices just after v is cleared and k be the total number of searchers on V'. If v is occupied by searchers after it is cleared, then these searchers will stay on v until the end of the search. For each vertex $u \in V' \setminus \{v\}$, if $\deg(u)$ is even, then each searcher on u maybe move to an odd vertex in the rest of the searching process; if $\deg(u)$ is odd, either a searcher was placed on u, or a searcher slid to u and this searcher will stay on u until the end of the search. Thus, just after v is cleared, we need at least $\frac{1}{2}\max\{(|V_{\mathrm{odd}}(G) \setminus \{v\}| - k), 0\}$ additional searchers to clear G. Notices that $N(v) \subseteq V'$ and $|V'| \leq k$. Therefore, $\mathrm{fsn}(G) \geq k + \frac{1}{2}\max\{(|V_{\mathrm{odd}}(G) \setminus \{v\}| - k), 0\}$ $\geq \frac{1}{2}\max\{(|V_{\mathrm{odd}}(G) \setminus \{v\}| + k), 0\} \geq \lceil \frac{\delta(G)+|V_{\mathrm{odd}}(G)|-1}{2} \rceil$. □

From Theorem 6.1, we can improve the approximation ratio of FastSearch(G).

Theorem 6.2. *If G is connected graph with n vertices and m edges, and $\delta(G) \geq 3$, then*

$$\frac{|V_p|+|A_s|}{\mathrm{fet}(G)} \leq \left(1+\frac{n-\delta(G)-1}{m+\delta(G)+1}\right).$$

7. Planar graphs

For a graph G, let $b(G)$ be the brush number of G. Since every brush cleaning strategy gives us a fast search strategy, we know that $b(G) \geq \mathrm{fsn}(G) \geq |V_{\mathrm{odd}}(G)|/2$. Thus, if $b(G) = |V_{\mathrm{odd}}(G)|/2$, then $\mathrm{fsn}(G) = |V_{\mathrm{odd}}(G)|/2$. But when $\mathrm{fsn}(G) = |V_{\mathrm{odd}}(G)|/2$, $b(G)$ can be as large as $\Omega(|V(G)|^2)$. For example, consider a complete graph K_n, where $n \geq 5$ and n is odd. Let K be the graph obtained from K_n by attaching three pendent edges on a vertex of K_n and attaching one pendent edge on all other vertices. We can show that $\mathrm{fsn}(K) = |V_{\mathrm{odd}}(K)|/2 = n+1$ and $b(K) \geq (n^2-1)/4$.

Let $F = \{f_1, f_2, \ldots, f_k\}$ be a family of plane curves satisfying the following conditions (see Fig. 3): (1) each f_i is the graph of a continues function of time with domain $[s_i, t_i]$, $-\infty < s_i < t_i < +\infty$, (2) any pair of curves do not share an endpoint, and (3) each pair of curves have a finite number of intersection points. From condition (2) we know that at each intersection point, at most one curve starts from or ends on this intersection point. From condition (3) we know that no pair of curves overlap over any period of time.

A graph of F, denoted by $G_F = (V_F, E_F)$, is the graph formed from F such that V_F is the set of all endpoints and intersection points of curves in F and $E_F = \{f : f$ is a subcurve of a curve in F whose endpoints belong to V_F and no interior point of f belongs to $V_F\}$. Note that the definition of the edge set E_F can be easily converted to the traditional definition, that is, a set of pairs of vertices.

Theorem 7.1. *Let $F = \{f_1, f_2, \ldots, f_k\}$ be a set of plane curves satisfying the above three conditions, and let $G_F = (V_F, E_F)$ be the graph of F. Then* $\mathrm{fsn}(G_F) = k$ *and* $\mathrm{fet}(G_F) = k + |E_F|$.

Proof. Since each vertex of G_F has coordinates, we can sort all vertices by alphabetical order from left to right in a sequence, say $v_1, v_2, \ldots, v_n$. For simplicity, we consider a continues version of fast searching that is equivalent to our fast search model. For each curve f_i, place a searcher λ_i on the left endpoint of f_i and associate λ_i with f_i. We sweep an imaginary vertical line ℓ from the position passing through v_1 to the position passing through v_n. At any moment when ℓ moves, each intersection point between ℓ and a curve f_i is occupied by the searcher λ_i. It is easy to see that no recontamination can happen because all

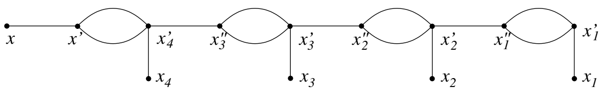

Fig. 4. A variable gadget G_x^k with $k = 4$.

subcurves on the left-hand side of ℓ are cleared and on the right-hand side are dirty. This can be easily converted to the fast searching because every edge is traversed exactly once. Thus, $\text{fsn}(G_F) \le k$. Note that $\text{fsn}(G_F) \ge |V_{\text{odd}}(G_F)|/2 = k$. Therefore, $\text{fsn}(G_F) = k$. It follows from Theorem 3.1 that $\text{fet}(G_F) = k + |E_F|$. $\square$

From [3], we know that the vertex-ordering $v_1, v_2, \ldots, v_n$ in the proof of Theorem 7.1 is perfectly balanced. We can extend Theorem 7.1 to curves in 3D space.

Corollary 7.2. *Let $F = \{f_1, f_2, \ldots, f_k\}$ be a set of curves in 3D space satisfying the following conditions, (1) each curve has at most one intersection point with the plane parallel to xy-plane, (2) any pair of curves do not share an endpoint, and (3) each pair of curves have a finite number of intersection points. Let $G_F = (V_F, E_F)$ be a graph of F. Then $\text{fsn}(G_F) = k$ and $\text{fet}(G_F) = k + |E_F|$.*

The conditions in Theorem 7.1 and Corollary 7.2 are sufficient, but not necessary. Let K_n be a complete graph of order n ($n \ge 5$) and n be an odd number. Let P_m be a path of length m ($m \ge 1$). Let H be the cartesian product of K_n and P_m. Let H' be a graph obtained from H by adding a path $u_0 u_1 \ldots u_{m+1} u_{m+2}$ to H such that u_0 and u_{m+2} are leaves, each u_i, $1 \le i \le m+1$, is a vertex in the ith copy of K_n, and each pair of adjacent vertices u_i and u_{i+1}, $1 \le i \le m$, are not matched in H. It is easy to see that H' is a simple graph with $2(n+1)$ odd vertices. Since each copy of K_n is an Eulerian graph, we can clear H' using $n+1$ searchers. Thus, $\text{fsn}(H') = n+1$. But we cannot draw $n+1$ monotonic curves in 3D as those curves in Corollary 7.2.

From [14], we know that the fast search number of cubic graphs can be found in $O(n^2)$ time. Similar to [8], we now show that the fast search problem is NP-complete for planar graphs with maximum degree 4. We first show a property of variable gadgets as follows.

Lemma 7.3. *Let G_x^k be a multigraph as illustrated in Fig. 4. For any optimal fast search strategy of G_x^k, if a searcher slides from x to its neighbor x', then for each leaf x_i ($1 \le i \le k$) there is a searcher sliding to x_i from its neighbor x'_i; and if a searcher slides to x from its neighbor x', then for each leaf x_i ($1 \le i \le k$) there is a searcher sliding from x_i to its neighbor x'_i.*

Proof. Refer to Fig. 4. If we place $k+1$ searchers on vertices $x'_1, x_1, x_2, \ldots, x_k$, then we can clear G_x^k by sliding searchers from all x_i to their neighbors and then sliding searchers from right to left along the remaining edges. Thus, $\text{fsn}(G_x^k) \le k+1$. On the other hand, $\text{fsn}(G_x^k) \ge |V_{\text{odd}}(G_x^k)|/2 = k+1$. Therefore, $\text{fsn}(G_x^k) = k+1$. Suppose that there is an optimal fast search strategy S, in which xx' is cleared by sliding from x to x' and $x'_i x_i$ is cleared by sliding from x_i to x'_i. Without loss of generality, we can suppose that each $x'_j x_j$ ($i+1 \le j \le k$) is cleared by sliding from x'_j to x_j. Since S is an optimal fast search strategy with $\text{fsn}(G_x^k) = k+1$ and $|V_{\text{odd}}(G_x^k)|/2 = k+1$, every searcher must start from an odd vertex and end at another odd vertex. If $i < k$, since xx' is cleared by sliding from x to x', $x'_k x_k$ is cleared by sliding from x'_k to x_k, and x'_k is an even vertex, we know that the two parallel edges between x' and x'_k must be cleared by sliding two searchers from x' to x'_k. Using the similar argument for each j from k down to $i+1$, we know that the two parallel edges incident on x'_{i+1} must be cleared by sliding two searchers to x'_{i+1}, and then one of them slides from x'_{i+1} to x_{i+1} and the other slides to vertex x''_i that is adjacent to x'_i. Let x'_i be occupied by searcher λ_1 and x''_i be occupied by λ_2. Note that the two parallel edges between x''_i and x'_i are still contaminated. If there is another searcher λ starting from x''_i and sliding to x'_i and then back to x''_i, then there are two searchers ending at x''_i. This is a contradiction. If there is another searcher λ starting from x''_i and sliding to x'_i and λ_2 also slides to x'_i along the other parallel edge, then there are at least two searchers ending at x'_i. This is a contradiction. Similarly, we can find contradictions for all other cases. Therefore, if a searcher slides from x to its neighbor, then for each leaf x_i ($1 \le i \le k$) there is a searcher sliding to x_i from its neighbor.

Note that a fast search strategy is reversible. Thus, if a searcher slides to x from its neighbor in an optimal fast search strategy, then for each leaf x_i ($1 \le i \le k$) there is a searcher sliding from x_i to its neighbor. $\square$

We now use the graph G_x^k in Lemma 7.3 as a variable gadget to show the NP-completeness for planar graphs with maximum degree 4. The reduction is the same as the one used in Theorem 1 of [8]. Because the difference between fast searching and perfect ordering, we need to argue differently.

Theorem 7.4. *Given a planar graph G with maximum degree 4, the problem of determining whether $\text{fsn}(G) = |V_{\text{odd}}(G)|/2$ is NP-complete.*

Proof. It is easy to see that the problem is in NP. We will show it is NP-hard by a reduction from the planar positive 2-in-4SAT problem. Let ϕ be a boolean formula in the conjunctive normal form with m clauses $\{c_1, \ldots, c_m\}$ and n variables $x_1, \ldots, x_n$. That is, $\phi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, where each clause c_i is a disjunctive of four variables. The incident graph of ϕ is the bipartite graph with vertex set $\{c_1, \ldots, c_m, x_1, \ldots, x_n\}$ and edge set $\{c_i x_j : c_i \text{ contains } x_j\}$. The formula ϕ is planar if the incident graph is planar. A truth assignment of ϕ is 2-in-4 satisfying if each clause has exactly two true variables, and ϕ is 2-in-4 satisfiable if there is a 2-in-4 satisfying truth assignment. From [8], we know that the problem of determining whether a planar positive formula ϕ is 2-in-4 satisfiable is NP-complete.

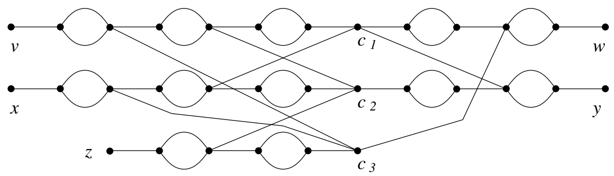

Fig. 5. The graph G_ϕ constructed for $\phi = c_1 \wedge c_2 \wedge c_3$, where $c_1 = (v \vee w \vee x \vee y)$, $c_2 = (v \vee x \vee y \vee z)$ and $c_3 = (v \vee w \vee x \vee z)$.

We now construct an instance of the planar fast searching problem. For each clause c_i, $1 \leq i \leq m$, we construct a vertex c_i as the clause gadget. For each variable x that appears k times in ϕ, we construct the gadget G_x^k (refer to Fig. 4) to correspond to the variable x. Note that G_x^k has $k+1$ leaves $x, x_1, \ldots, x_k$ and x_i $(1 \leq i \leq k)$ corresponds to the i-th occurrence of the variable x. For each variable gadget G_x^k with leaves $x, x_1, \ldots, x_k$, connect vertex x_i to vertex c_j such that the clause c_j contains the i-th occurrence of the variable x. In polynomial time, we can construct a graph with maximum degree 4, denoted by G_ϕ (see Fig. 5). We will show that the planar positive formula ϕ is 2-in-4 satisfiable if and only if $\mathrm{fsn}(G_\phi) = |V_{\mathrm{odd}}(G_\phi)|/2$.

Suppose that the planar positive formula ϕ is 2-in-4 satisfiable. Then G_ϕ is a planar graph. Consider a 2-in-4 satisfying truth assignment of ϕ. For a variable x whose value is true and appearing k times in ϕ, we clear the variable gadget G_x^k by sliding a search from x to x' to clear xx', and sliding a searcher from x_i' to x_i to clear each x_ix_i'. The remaining edges in G_x^k can be cleared correspondingly such that $k+1$ searchers starting from vertices of G_x^k, among them one searcher ends on x_1' and k searchers slide into clause gadgets. For each variable x whose value is false and appearing k times in ϕ, we clear the variable gadget G_x^k by sliding a searcher from x_i to x_i' to clear each x_ix_i' and sliding a search from x' to x to clear xx'. The remaining edges in G_x^k can be cleared correspondingly such that k searchers slide into G_x^k from clause gadgets, one searcher starts from x_1' and ends on x. Since each clause has four variables and two of them have true value, each clause gadget (vertex) has two searchers sliding in and two searchers sliding out. Thus, G_ϕ is cleared by $n+2m$ searchers. Since $|V_{\mathrm{odd}}(G_\phi)| = 2n+4m$, we have $\mathrm{fsn}(G_\phi) = |V_{\mathrm{odd}}(G_\phi)|/2$.

Conversely, suppose that $\mathrm{fsn}(G_\phi) = |V_{\mathrm{odd}}(G_\phi)|/2$. From [14], for each odd vertex, there is at least one searcher is placed on it or occupies it at the end of the game. Since $\mathrm{fsn}(G_\phi) = |V_{\mathrm{odd}}(G_\phi)|/2$, for each even vertex in G_ϕ, there is no searcher is placed on it or occupies it at the end of the game. From Lemma 7.3, for each gadget G_x^k, if edge xx' is cleared by sliding a search from x to x', then we set the corresponding variable true. Note that each $x_i'x_i$ $(1 \leq i \leq k)$ is cleared by sliding a searcher from x_i' to x_i. If edge xx' is cleared by sliding a search from x' to x, then we set the corresponding variable false. Note that each $x_i'x_i$ $(1 \leq i \leq k)$ is cleared by sliding a searcher from x_i to x_i'. Since for each clause gadget c in G_ϕ, it has degree 4 and there is no searcher is placed on it or occupies it at the end of the game, we know that two searchers slide into c and two slide out. Thus, ϕ is 2-in-4 satisfiable.

The multigraph can be easily transformed to a graph by replacing one of each parallel edge by a path of length 2 such that both of them have the same fast search number. □

From Corollary 3.3, we can show that, given a planar graph G with maximum degree 4, it is NP-complete to determine whether $\mathrm{fen}(G) = |V_{\mathrm{odd}}(G)|/2$.

Corollary 7.5. *Given a planar graph G with maximum degree 4, the problem of determining whether* $\mathrm{fet}(G) = \frac{1}{2}|V_{\mathrm{odd}}(G)| + |E(G)|$ *is NP-complete.*

8. Conclusions

Many graph searching problems have been introduced. Most of these problems only consider the minimum number of searchers required to capture the fugitive. In this paper, we consider the minimum number of steps to capture the fugitive. We introduce the fast edge searching problem in the edge search model. We establish relations between the fast edge searching and the fast searching in the fast search model. We also establish relations between the fast (edge) searching and the node searching. By these relations, the problem of computing the fast search number, edge search number, node search number, or pathwidth of a graph is equivalent to that of computing the fast edge-search time of a related graph. This makes the fast (edge) searching is more versatile than others. We can use the fast (edge) searching to investigate either how to draw a graph "evenly" (an extended version of the balanced vertex-ordering), or how to decompose a graph into a "path" (i.e., pathwidth, which is related to many graph parameters). We show that the family of graphs whose fast edge-search time is at most k is minor-closed. This makes arguments for upper bounds and lower bounds of the fast edge-search time less complicated, comparing with the fast search number. We prove NP-completeness results for computing the fast (edge-)search number, and the fast edge-search time, respectively. We also prove that the problem of determining whether $\mathrm{fsn}(G) = \frac{1}{2}|V_{\mathrm{odd}}(G)|$ or $\mathrm{fet}(G) = \frac{1}{2}|V_{\mathrm{odd}}(G)| + |E(G)|$ is NP-complete; and it remains NP-complete for planar graphs with maximum degree 4. For connected graphs with $\delta(G) \geq 3$, we present a linear time approximation algorithm for the fast edge-search time that can give solutions of at most $(1 + \frac{|V|-\delta(G)-1}{|E|+\delta(G)+1})$ times the optimal value. This algorithm also gives us a tight upper bound on the fast search number of graphs.

Acknowledgement

The author would like to thank Donald Stanley for comments and discussions on this paper. This research was supported in part by NSERC.

References

[1] N. Alon, P. Pralat, R. Wormald, Cleaning regular graphs with brushes, SIAM Journal on Discrete Mathematics 23 (2008) 233–250.
[2] B. Alspach, D. Dyer, D. Hanson, B. Yang, Lower bounds on edge searching, in: Proceedings of the 1st International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies, ESCAPE'07, in: Lecture Notes in Computer Science, vol. 4614, Springer, 2007, pp. 516–527.
[3] T. Biedl, T. Chan, Y. Ganjali, M.T. Hajiaghayi, D. wood, Balanced vertex-ordering of graphs, Discrete Applied Mathematics 148 (2005) 27–48.
[4] D. Bienstock, P. Seymour, Monotonicity in graph searching, Journal of Algorithms 12 (1991) 239–245.
[5] A. Bonato, P. Golovach, G. Hahn, J. Kratochvíl, The capture time of a graph, Discrete Mathematics 309 (2009) 5588–5595.
[6] D. Dereniowski, Ö. Diner and D. Dyer, Three-fast-searchable graphs, manuscript.
[7] D. Dyer, B. Yang, and Öznur Yaşar, On the fast searching problem, in: Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management, AAIM'08, in: Lecture notes in Computer Science, vol. 5034, Springer, 2008, pp. 143–154.
[8] J. Kára, J. Kratochvíl, D. Wood, On the complexity of the balanced vertex ordering problem, Discrete Mathematics and Theoretical Computer Science 9 (2007) 193–202.
[9] L. Kirousis, C. Papadimitriou, Searching and pebbling, Theoretical Computer Science 47 (1986) 205–218.
[10] A. LaPaugh, Recontamination does not help to search a graph, Journal of ACM 40 (1993) 224–245.
[11] F. Makedon, C. Papadimitriou, I. Sudborough, Topological bandwidth, SIAM Journal on Algebraic and Discrete Methods 6 (1985) 418–444.
[12] M.E. Messinger, R.J. Nowakowski, P. Pralat, Cleaning a network with brushes, Theoretical Computer Science 399 (2008) 191–205.
[13] N. Megiddo, S. Hakimi, M. Garey, D. Johnson, C. Papadimitriou, The complexity of searching a graph, Journal of ACM 35 (1988) 18–44.
[14] D. Stanley, B. Yang, Lower bounds on fast searching and their applications, in: Proceedings of the 20th International Symposium on Algorithms and Computation, ISAAC'09, in: Lecture notes in Computer Science, vol. 5878, Springer-Verlag, Berlin, 2009, pp. 964–973.
[15] D.B. West, Introduction to Graph Theory, Prentice Hall, 1996.