Journal of Combinatorial Optimization (2023) 45:84
https://doi.org/10.1007/s10878-023-01012-x

Fast searching on cactus graphs

Yuan Xue[1] · Boting Yang[1] · Sandra Zilles[1] · Lusheng Wang[2,3]

Accepted: 22 February 2023 / Published online: 19 March 2023

Abstract
The problem of finding the fast search number of a graph is NP-complete. It is challenging even when the graph has very small treewidth. However, it can be much easier to find an optimal fast search strategy for smaller subgraphs with special properties. This observation motivates us to establish relationships between optimal fast search strategies for a graph and its subgraphs although fast searching does not have the subgraph-closed property. In this paper, we introduce the notion of k-combinable graphs and study their properties. We propose a new method for computing the fast search number of k-combinable graphs. As an application of this method, we examine the fast searching for cactus graphs. We investigate the properties of optimal fast search strategies and give a linear time algorithm for computing the fast search number of cactus graphs.

1 Introduction

The first graph searching model was introduced by Parsons in 1976 (Parsons 1976). In this model we are given a graph that contains an invisible fugitive who hides on vertices or edges. Both searchers and the fugitive move along the edges of a graph in a

The preliminary result of this work has been published in the proceedings of the 16th International Conference on Algorithmic Aspects in Information and Management, 2022.

✉ Boting Yang
boting.yang@uregina.ca

Yuan Xue
xue228@uregina.ca

Sandra Zilles
zilles@cs.uregina.ca

Lusheng Wang
cswangl@cityu.edu.hk

[1] Department of Computer Science, University of Regina, Regina, SK S4S0A2, Canada

[2] Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong, China

[3] City University of Hong Kong Shenzhen Research Institution, Shenzhen, China

continuous way. Megiddo et al. (1988) introduced the edge searching model in which the searchers have three actions: placing, removing and sliding. They first proved that the edge searching problem is NP-hard, and then gave a $O(n)$-time algorithm for computing the edge search number of trees, where n is the number of vertices. Kirousis and Papadimitriou (1986) introduced the node searching model in which the searchers have two actions: placing and removing. They related the search number of an undirected graph G to the minimum and maximum of the progressive pebble demands of the directed acyclic graphs obtained by orienting G. Bienstock and Seymour (1991) introduced the mixed searching problem which is related the edge searching and node searching problems. They showed that both the edge searching problem and the node searching problem are monotonic (Bienstock and Seymour 1991; LaPaugh 1993). More models and results can be found in Alspach (2006), Bienstock (1991), Bonato and Nowakowski (2011), Bonato and Yang (2013), Fomin and Petrov (1996), Fomin and Thilikos (2008), and Hahn (2007).

Dyer et al. (2008) introduced the fast searching model. There are two actions for searchers in this model: placing and sliding. They proposed a $O(n)$-time algorithm to compute the fast search number of trees. In Yang (2011), Yang proved that the fast searching problem is NP-complete. Dereniowski et al. (2013) gave a characterization for the graphs that are 2-searchable or 3-searchable in the fast searching model. Stanley and Yang (2011) presented a $O(n^2)$-time algorithm for computing the fast search number of cubic graphs. Note that the problem of finding the node search number of cubic graphs is NP-complete (Makedon et al. 1985). Xue et al. (2018) provided lower bounds and upper bounds on the fast search number of complete k-partite graphs. They also solved an open problem about the fast search number of complete bipartite graphs. Xue and Yang (2017) investigated the fast search number of the Cartesian product of an Eulerian graph and a path. They also presented upper and lower bounds on the fast search number of hypercubes.

This paper is organized as follows. In Sect. 2, we give some definitions and notation. In Sect. 3, we introduce the notion of k-combinable graphs, and develop a new method for computing their fast search number. In Sect. 4, we investigate the optimal fast search strategies of cactus graphs. In Sect. 5, we use the new method to design a linear time algorithm for computing the fast search number of cactus graphs. In Sect. 6, we prove the correctness of our algorithm and analyze its running time. Finally, we conclude this paper in Sect. 7.

2 Preliminaries

In this paper, we only consider finite simple graphs. Let $G = (V, E)$, where V is a set of vertices and E is a set of edges. Let v be a vertex of G. The *degree* of v, written $\deg_G(v)$, is the number of edges incident to v. If $\deg_G(v) = 1$, then v is called a *leaf* of G. The edge that is incident with a leaf is called a *pendent edge*. Let V' be a subset of vertices of G. The subgraph of G that consists of all the vertices of V' and all the edges of G between vertices in V' is referred to as the *subgraph of G induced by V'*, denoted by $G[V']$. The graph that is obtained from G by deleting all the vertices of

V' is denoted by $G - V'$. For simplicity, the graph that is obtained from G by deleting a vertex v is denoted by $G - v$.

Let E' be a subset of edges of G. The graph that is obtained from G by deleting all the edges of E' is denoted by $G - E'$. If G is connected and E' contains only pendent edges, we abuse the notation by using $G - E'$ to denote the non-empty subgraph of G after the edges of E' are deleted from G. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be subgraphs of G. Define $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$.

In the fast searching model introduced by Dyer et al. (2008), we consider a graph that contains a fugitive. The fugitive can stay on any vertex or along any edge. Note that the fugitive is invisible to searchers. The fugitive can move at a great speed at any time from a vertex u to another vertex v if there is a path from u to v that contains no searchers. The searchers have two actions: *placing* and *sliding*. In each step of a fast searching process, we can either place a searcher on a vertex or slide a searcher along an edge from one endpoint to the other. Notice that the major difference between the fast searching model and the edge searching model is that the searchers cannot be removed from the graph and every edge can be traversed by searchers at most once in the fast searching model. If an edge may contain the fugitive, then this edge is called *contaminated*. If we are certain that an edge does not contain the fugitive, then we say that this edge is *cleared*. For a contaminated edge, if there is a searcher sliding along the edge from one endpoint to the other, then this edge becomes cleared. In order to capture the fugitive, the searchers clear the graph edge by edge until all edges are cleared. Since every edge can be traversed by searchers at most once, the searchers have to protect the already cleared edges from recontamination.

For a graph G, a *fast search strategy* of G is a sequence of searchers' actions that clear all contaminated edges of G. The *fast search number* of G, denoted by $\mathrm{fs}(G)$, is the smallest number of searchers required to capture the fugitive in G. A fast search strategy is *optimal* if the number of searchers used by this strategy is equal to $\mathrm{fs}(G)$.

For any positive integer k, we use $[k]$ to denote the set $\{1, 2, \ldots, k\}$.

3 Aligning operation and k-combinable graphs

In this section, we will introduce a class of graphs named k-combinable graphs. Then we describe our new method for finding an optimal fast search strategy for k-combinable graphs. Let G be a connected graph and let E'_G be the set of all pendent edges of G. A *component* of a graph G is a connected subgraph that is not part of any larger connected subgraph of G. The *profile* of G is an ordered tuple $\pi = (\pi_1, \ldots, \pi_z)$ of positive integers, which is defined as follows:

1. If $E'_G = \emptyset$, then $z = 1$ and $\pi_1 = \mathrm{fs}(G)$.
2. If $|E'_G| = k \geq 1$, then $z = k!2^k$ and each component π_i of π is associated with a specific permutation σ and a specific orientation of each edge in E'_G. In particular, π_i is the smallest number of searchers placed on non-leaf vertices of G satisfying that these searchers and the searchers placed on leaves (where searchers have to start from these leaves depending on orientations) can clear G such that they traverse the edges in E'_G in the order of σ and in the directions as given by the chosen orientations.

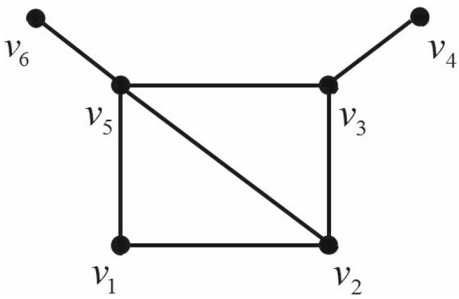

Fig. 1 A graph with profile (1, 2, 3, 3, 1, 2, 3, 3) shown in Table 1

Table 1 Each π_i, the smallest number of searchers placed on non-leaf vertices of the graph in Fig. 1, depends on the permutation and orientation of the pendent edges

Permutation	Orientation		π_i
	$v_3 \leftrightarrow v_4$	$v_5 \leftrightarrow v_6$	
(v_3v_4, v_5v_6)	$\leftarrow$	$\leftarrow$	1
(v_3v_4, v_5v_6)	$\leftarrow$	$\rightarrow$	2
(v_3v_4, v_5v_6)	$\rightarrow$	$\leftarrow$	3
(v_3v_4, v_5v_6)	$\rightarrow$	$\rightarrow$	3
(v_5v_6, v_3v_4)	$\leftarrow$	$\leftarrow$	1
(v_5v_6, v_3v_4)	$\leftarrow$	$\rightarrow$	2
(v_5v_6, v_3v_4)	$\rightarrow$	$\leftarrow$	3
(v_5v_6, v_3v_4)	$\rightarrow$	$\rightarrow$	3

We use the graph in Fig. 1 to explain the profile. This graph has two pendent edges v_3v_4 and v_5v_6. Each pendent edge is cleared by sliding a searcher from one endpoint to the other. The profile of the graph is (1, 2, 3, 3, 1, 2, 3, 3), and Table 1 lists each search number that is associated with a specific permutation and orientation of $\{v_3v_4, v_5v_6\}$. For the first component, by the permutation and orientation of $\{v_3v_4, v_5v_6\}$, we know that v_3v_4 is cleared before v_5v_6 by sliding a searcher from v_4 to v_3 ($v_3 \leftarrow v_4$) and sliding another searcher from v_6 to v_5 ($v_5 \leftarrow v_6$). So the graph can be cleared by placing a searcher on v_3 and clear the remaining edges by sliding $v_3 \rightarrow v_2$, $v_3 \rightarrow v_5 \rightarrow v_2 \rightarrow v_1 \rightarrow v_5$. Notice that π_i is the smallest number of searchers placed on non-leaf vertices. Thus $\pi_1 = 1$. For π_2, by the permutation and orientation of $\{v_3v_4, v_5v_6\}$, we know that sliding $v_3 \leftarrow v_4$ to clear v_3v_4 is before sliding $v_5 \rightarrow v_6$ to clear v_5v_6. After v_3v_4 is cleared by sliding $v_4 \rightarrow v_3$, the remaining graph can be cleared by placing a searcher on v_3 and v_5, respectively, and sliding $v_3 \rightarrow v_2$, $v_3 \rightarrow v_5 \rightarrow v_2 \rightarrow v_1 \rightarrow v_5 \rightarrow v_6$. Thus $\pi_2 = 2$. For π_3, by the permutation and orientation, sliding $v_3 \leftarrow v_4$ to clear v_3v_4 is before sliding $v_6 \rightarrow v_5$ to clear v_5v_6. We have to place three searchers on v_3 and clear all incident edges of v_3 by sliding $v_3 \rightarrow v_4$, $v_3 \rightarrow v_5$ and $v_3 \rightarrow v_2$. The remaining graph can be cleared by sliding $v_6 \rightarrow v_5 \rightarrow v_2 \rightarrow v_1 \rightarrow v_5$. So $\pi_3 = 3$. For π_4, by the permutation and orientation, sliding $v_3 \rightarrow v_4$ is before sliding $v_5 \rightarrow v_6$. We first place two searchers on v_5 and one searcher on v_2. Then clear all edges by sliding $v_5 \rightarrow v_2 \rightarrow v_1 \rightarrow v_5 \rightarrow v_3$, $v_2 \rightarrow v_3 \rightarrow v_4$ and $v_5 \rightarrow v_6$. Hence $\pi_4 = 3$. The remaining π_i, $5 \leq i \leq 8$, can be determined similarly.

Let G_1 be a connected graph that has $k_1 \geq 1$ pendent edges, and let G_2 be a connected graph having $k_2 \geq 1$ pendent edges. Let k be an integer satisfying $1 \leq k \leq \min\{k_1, k_2\}$. Let $\overrightarrow{e_1} = (u_1u'_1, \ldots, u_ku'_k)$, where $u_iu'_i \in E(G_1)$ and u'_i is a leaf. Let $\overrightarrow{e_2} = (v_1v'_1, \ldots, v_kv'_k)$, where $v_iv'_i \in E(G_2)$ and v'_i is a leaf. Let H be the graph obtained from G_1 and G_2 by performing the following operations on G_1 and G_2 with respect to $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$: For each $i \in [k]$, remove leaves u'_i and v'_i, and add edge u_iv_i.

Note that the above operations depend on the choice of the sequences $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ which we will henceforth call *edge pairing sequences*. If we permute either of the edge pairing sequences, this would create a different graph H. Hence, we define the *aligning operation* on G_1 and G_2 with respect to $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$, denoted as $(G_1, \overrightarrow{e_1})\triangle(G_2, \overrightarrow{e_2})$, to be the above operations to obtain H.

Definition 3.1 Let $G_1, \ldots, G_m$, $m \geq 2$, be a sequence of connected graphs. We say that the sequence $(G_1, \ldots, G_m)$ is *k-combinable* if there are edge pairing sequences $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}, \overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$ satisfying that:

1. for each $i \in [m]$, $\overrightarrow{e_i}$ is a sequence of pendent edges of G_i;
2. for each $i \in \{2, \ldots, m-1\}$, $\overrightarrow{e_{1,i}}$ is a sequence of pendent edges of H_i, where $H_2 = (G_1, \overrightarrow{e_1})\triangle(G_2, \overrightarrow{e_2})$, and $H_{i+1} = (H_i, \overrightarrow{e_{1,i}})\triangle(G_{i+1}, \overrightarrow{e_{i+1}})$;
3. for each $i \in [m]$, the set of all edges of G_i, which occur in $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$ and $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$, has size at most k; and
4. for each $i \in \{2, \ldots, m-1\}$, the set of all edges of H_i, which occur in $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$ and $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$, has size at most k.

We call H_m a *k-combination* of $(G_1, \ldots, G_m)$, in particular, this is the k-combination of $(G_1, \ldots, G_m)$ with respect to $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}, \overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$.

Note that there may exist more than one graph that is a k-combination of $(G_1, \ldots, G_m)$. Further, for each k-combination G of $(G_1, \ldots, G_m)$, there exist specific edge pairing sequences $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}, \overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$ to obtain G. In the remainder of this section, we always assume that every time an algorithm handles profiles of graphs, it implicitly associates the profiles with corresponding edge pairing sequences.

Theorem 3.2 *Given the profiles and edge pairing sequences of two connected graphs G_1 and G_2, let G be a k-combination of (G_1, G_2) with respect to the edge pairing sequences. Then there exists an algorithm that runs in $O(k!(k_1+k_2-2k)!2^{k_1+k_2-k})$ time to compute the profile of G, where k_1 and k_2 refer to the number of pendent edges of G_1 and G_2 respectively.*

Proof Let $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ denote the edge pairing sequences of G_1 and G_2, respectively, such that G is the k-combination of (G_1, G_2) with respect to $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$. Consider all the edges in $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$. If we are given a set of rules instructing how these edges are cleared in a fast search strategy, then in accordance with the rules, we can figure out the number of searchers that need to be placed on the non-leaf vertices in $V(G)$. Thus, for each component in the profile of G, it takes $O(k!2^k)$ time to compute its value, where $k!2^k$ is the number of the permutations and orientations of the k pairing edges. Note that the number of components in the profile of G is $(k_1+k_2-2k)!2^{k_1+k_2-2k}$. Hence, the time complexity for computing the profile of G is $O(k!(k_1+k_2-2k)!2^{k_1+k_2-k})$. □

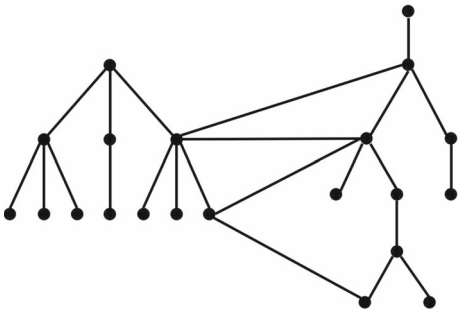

Fig. 2 A graph that can be split into two vertex-disjoint trees by deleting four edges

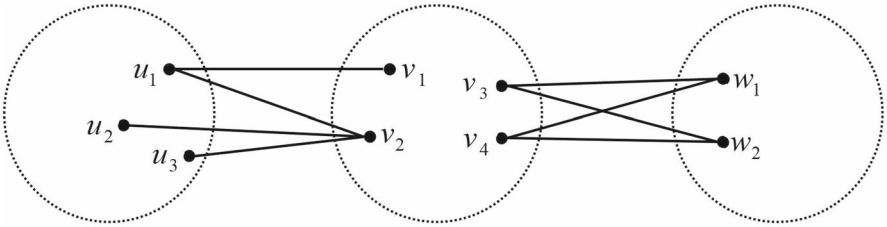

Fig. 3 A graph consists of three components after the solid edges are deleted

It follws from Theorem 3.2 that our method can be applied to find an optimal fast search strategy for quite complicated graphs, if the graph can be split into two smaller graphs for which fast search strategies are easy to find. For example, consider the graph in Fig. 2. This graph G can be decomposed into two vertex-disjoint trees T_1 and T_2, and the two trees are connected by four edges. So G is a 4-combination of (T_1', T_2') with respect to the edge pairing sequences, where T_i', $1 \leq i \leq 2$, is the tree obtained from T_i by attaching four pendent edges in the edge pairing sequence. Note that the fast search number of a tree can be computed in linear time (Dyer et al. 2008). Upon knowing the profiles of T_1' and T_2', we then can apply the new method presented in Theorem 3.2 to compute the fast search number of the graph G.

Moreover, if we are given G that is a k-combination of $(G_1, \ldots, G_m)$ where $m \geq 3$, by repeatedly applying the procedure presented in the proof of Theorem 3.2, we can find an optimal fast search strategy for G. This novel method reveals an interesting property of fast searching that has not been exploited systematically in the literature to date. Moreover, as we will show in the remainder of this paper, our method can be applied to a wide class of graphs.

Corollary 3.3 *Let G be a k-combination of $(G_1, \ldots, G_m)$ with respect to edge pairing sequences $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}, \overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$, where $G_1, \ldots, G_m$ are connected graphs and k is a constant. Given the profiles of G_1, ..., G_m as input, there exists an algorithm which runs in polynomial time to compute the fast search number of G.*

Note that in Definition 3.1, G_i may not always be a subgraph of G. Consider the graph in Fig. 3. Suppose that after removing edges $u_1v_1, u_1v_2, u_2v_2, u_3v_2, v_3w_1, v_3w_2, v_4w_1$ and v_4w_2, the remaining graph consists of three components that are represented by three dotted circles. Let H_1, H_2 and H_3 denote the three components from left to

right. Let H_1' be obtained from H_1 by adding the edges u_1v_1, u_1v_2, u_2v_2 and u_3v_2. Let G_1 be obtained from H_1 by adding four pendent edges u_1u_1', u_1u_2', u_2u_3', u_3u_4'. Consider H_1'. Note that u_1, u_2 and u_3 have a common neighbor v_2. Hence, if v_2 has both cleared and contaminated incident edges at some moment in a fast search strategy, then at least one searcher must reside on v_2. In G_1, however, we know that u_1, u_2 and u_3 share no common neighbor that is outside H_1. Hence, no searcher needs to reside on the leaves u_1', u_2', u_3', u_4' whenever their incident pendent edges are cleared.

In the next section, we will use Theorem 3.2 to compute the fast search number of cactus graphs.

4 Cactus graphs

A *cactus* is a connected graph in which each edge is contained in at most one cycle. A *cut-vertex* of a graph is a vertex whose deletion increases the number of components. Throughout this section, we use G to denote a cactus.

Definition 4.1 Let G be a cactus that contains a cut-vertex v. Let H be a component in $G - v$, where a component of a graph G is a maximal connected subgraph of G. The subgraph of G induced by $V(H) \cup \{v\}$, denoted by G_v, is called a *subcactus* of G attached to v. For a subcactus G_v, if the degree of v in G_v is one, let G_v' denote the graph obtained from G_v by adding one pendent edge to v; if the degree of v in G_v is two, let G_v' denote the graph obtained from G_v by adding two pendent edges to v. G_v' is called an *extension* of G_v with respect to v.

For a subcactus G_v defined in Definition 4.1, it is easy to see that v has degree at most two in G_v. So we have two cases.

CASE 1. If v has degree one in G_v, let $u \in V(G_v)$ be the neighbor of v. An *I-strategy* for G_v is a fast search strategy such that vu is cleared by sliding a searcher from v to u, and the number of searchers placed on $V(G_v) \setminus \{v\}$ is minimum. This minimum number is denoted by $\pi_I(G_v)$. Note that if vu is cleared by sliding a searcher from v to u in a fast search strategy, then a searcher must be placed on v at the beginning of the strategy and this searcher is not counted in $\pi_I(G_v)$. Similarly, an *O-strategy* for G_v is a fast search strategy such that vu is cleared by sliding a searcher from u to v, and the number of searchers placed on $V(G_v)\setminus\{v\}$ is minimum. This minimum number is denoted by $\pi_O(G_v)$.

For the extension G_v' of G_v with respect to v, let wv be the added pendent edge such that w is not in G_v and has degree one in G_v'. An *I-strategy* for G_v' is a fast search strategy such that wv is cleared by sliding a searcher from w to v, and the number of searchers placed on $V(G_v)$ is minimum. This minimum number is denoted by $\pi_I(G_v')$. Similarly, we can define *O-strategy* for G_v' and $\pi_O(G_v')$.

CASE 2. If v has degree two in G_v, let $u_1, u_2 \in V(G_v)$ be the two neighbors of v. For $i \in \{1, 2\}$, we say vu_i is cleared by a *slide-in* action if a searcher slides from v to u_i along vu_i, and we say vu_i is cleared by a *slide-out* action if a searcher slides from u_i to v along vu_i. We use $\pi_{II}(G_v)$ (resp. $\pi_{OO}(G_v)$) to denote the minimum number of searchers placed on $V(G_v) \setminus \{v\}$ in a fast search strategy for G_v, in which

vu_1 and vu_2 are both cleared by slide-in (resp. slide-out) actions. We use $\pi_{IO}(G_v)$ (resp. $\pi_{OI}(G_v)$) to denote the minimum number of searchers placed on $V(G_v) \setminus \{v\}$ in a fast search strategy for G_v, in which a slide-in (resp. slide-out) action clears one of vu_1 and vu_2 first, and a slide-out (resp. slide-in) action clears the other edge later. An *II-strategy* for G_v is a fast search strategy such that vu_1 and vu_2 are both cleared by slide-in actions and the number of searchers placed on $V(G_v) \setminus \{v\}$ is $\pi_{II}(G_v)$. Similarly we can define *IO-strategy*, *OI-strategy* and *OO-strategy* for G_v.

For the extension G'_v of G_v, let w_1v and w_2v be two added pendent edges where $w_1, w_2 \notin V(G_v)$ and both have degree one in G'_v. We use $\pi_{II}(G'_v)$ to denote the minimum number of searchers placed on $V(G_v)$ in a fast search strategy for G'_v, in which w_1v and w_2v are both cleared by slide-in actions (i.e., sliding from w_i to v along w_iv). An *II-strategy* for G'_v is a fast search strategy such that w_1v and w_2v are both cleared by slide-in actions and the number of searchers placed on $V(G_v)$ is $\pi_{II}(G'_v)$. Similarly we can define $\pi_{IO}(G'_v)$, $\pi_{OI}(G'_v)$, $\pi_{OO}(G'_v)$, *IO-strategy*, *OI-strategy* and *OO-strategy* for G'_v.

Definition 4.2 Let G_v and G'_v be defined in Definition 4.1. If v is a leaf in G_v, then the profile of G_v (resp. G'_v) is defined as the pair $(\pi_I(G_v), \pi_O(G_v))$ (resp. $(\pi_I(G'_v), \pi_O(G'_v))$). If v has degree two in G_v, the profile of G_v (resp. G'_v) is defined as the 4-tuple $(\pi_{II}(G_v), \pi_{IO}(G_v), \pi_{OI}(G_v), \pi_{OO}(G_v))$ (resp. $(\pi_{II}(G'_v), \pi_{IO}(G'_v), \pi_{OI}(G'_v), \pi_{OO}(G'_v))$).

Note that in Definition 4.2, when v has degree two in G_v, the profile of the extension G'_v is defined as the 4-tuple $(\pi_{II}(G'_v), \pi_{IO}(G'_v), \pi_{OI}(G'_v), \pi_{OO}(G'_v))$, instead of an 8-tuple in the general definition of profile π at the beginning of this section. The reason is that the two pendent edges in G'_v are incident with the same vertex, i.e., v. So the ordering of clearing these two pendent edges, that is, the permutation of the two edges, does not affect the fast search number of G'_v. Thus we simplify the notation of profile by only considering the orientation of the two pendent edges.

Definition 4.3 Let $\mathcal{S}$ be a fast search strategy for G. The *reverse* of $\mathcal{S}$ is obtained from $\mathcal{S}$ by making the following modifications:

1. Remove all placing actions from $\mathcal{S}$. For each vertex $v \in V(G)$ that contains searchers at the end of $\mathcal{S}$, insert placing actions at the beginning that place the same number of searchers on v.
2. For each edge $e \in E(G)$, reverse the sliding action on e by letting searcher move in the opposite direction to clear it.
3. Reverse the ordering of all sliding actions.

It is not hard to prove that a fast search strategy $\mathcal{S}$ and its reverse use the same number of searchers to clear G.

We now show that the profile of a subcactus G_v must have one of the properties in the following theorem.

Theorem 4.4 *Let G_v be a subcactus defined in Definition 4.1. If v has degree two in G_v, then the profile of G_v has one of the properties (R_1) - (R_6). If v is a leaf in G_v, then the profile of G_v has property (R_7). Moreover, for each of these relations, there exist subcacti whose profiles have the relation.*

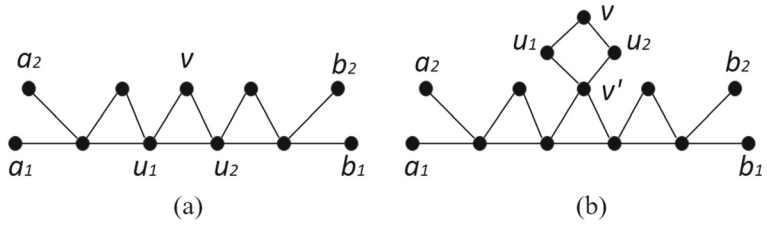

Fig. 4 **(a)** $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v) = 2$, and $\pi_{OO}(G_v) = 4$. **(b)** $\pi_{II}(G_v) = \pi_{IO}(G_v) = 2$, $\pi_{OI}(G_v) = 3$, and $\pi_{OO}(G_v) = 4$

(R_1) $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v) = \pi_{OO}(G_v) - 2$;
(R_2) $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v) - 1 = \pi_{OO}(G_v) - 2$;
(R_3) $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v) - 2 = \pi_{OO}(G_v) - 2$;
(R_4) $\pi_{II}(G_v) = \pi_{IO}(G_v) - 1 = \pi_{OI}(G_v) - 1 = \pi_{OO}(G_v) - 2$;
(R_5) $\pi_{II}(G_v) = \pi_{IO}(G_v) - 1 = \pi_{OI}(G_v) - 2 = \pi_{OO}(G_v) - 2$;
(R_6) $\pi_{II}(G_v) = \pi_{IO}(G_v) - 2 = \pi_{OI}(G_v) - 2 = \pi_{OO}(G_v) - 2$;
(R_7) $\pi_I(G_v) = \pi_O(G_v) - 1$.

Proof If v has exactly one incident edge in G_v, let u be the neighbor of v. Note that an *I-strategy* for G_v is a fast search strategy such that vu is cleared by sliding a searcher from v to u, and the searcher placed on v is not counted in $\pi_I(G_v)$. Notice that a fast search strategy for G_v and its reverse use the same number of searchers to clear G_v. Thus $\pi_I(G_v) = \pi_O(G_v) - 1$, that is, the profile of G_v has property (R_7).

Suppose that v has two incident edges in G_v. Let $u_1, u_2 \in V(G_v)$ be the two neighbors of v. Note that an *II-strategy* for G_v is a fast search strategy such that vu_1 and vu_2 are both cleared by slide-in actions and the number of searchers placed on $V(G_v) \setminus \{v\}$ is $\pi_{II}(G_v)$. So the two searchers placed on v are not counted in $\pi_{II}(G_v)$. Since a fast search strategy and its reverse use the same number of searchers, we have $\pi_{II}(G_v) = \pi_{OO}(G_v) - 2$. Furthermore, from the definitions of $\pi_{IO}(G_v)$ and $\pi_{OI}(G_v)$, we have $\pi_{II}(G_v) \leq \pi_{IO}(G_v) \leq \pi_{OI}(G_v) \leq \pi_{OO}(G_v)$. Hence, there are six possible relations, i.e., (R_1) - (R_6), among the four components of the profile of G_v. We will show that each of these relations is held for some subcacti G_v.

(i) We first show that the subcactus in Fig. 4a satisfies property (R_1). For an II-strategy, we place searcher λ_1 on a_1 and place λ_2 on a_2. Let λ_3 and λ_4 be two slide-in searchers on v, and move them to u_1 and u_2 respectively. Note that λ_3 and λ_4 are not counted in $\pi_{II}(G_v)$. Then move λ_1 and λ_2 to u_1 along two edge-disjoint paths respectively, move λ_1 to u_2, and move λ_1 and λ_4 to b_1 and b_2 along two edge-disjoint paths respectively. Thus $\pi_{II}(G_v) \leq 2$. For an IO-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Let λ_3 be the slide-in searcher that moves from v to u_2 (λ_3 is not counted in $\pi_{IO}(G_v)$). Then move λ_1 and λ_2 to u_1, move λ_1 to u_2, move λ_2 to v, and move λ_1 and λ_3 to b_1 and b_2 respectively. So $\pi_{IO}(G_v) \leq 2$. For an OI-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Then move λ_1 and λ_2 to b_1 and b_2 along two edge-disjoint paths respectively. Hence $\pi_{OI}(G_v) \leq 2$. For an OO-strategy, we place λ_1 on a_1 and place λ_2 on a_2. We place λ_3 and λ_4 on u_2. Then move λ_1 and λ_2 to u_1, move λ_2 to v, move λ_1 to u_2, move λ_4 to v, and move λ_1 and λ_3 to b_1 and b_2 respectively. Thus $\pi_{OO}(G_v) \leq 4$. Note that one searcher cannot clear G_v in an II-strategy, IO-strategy,

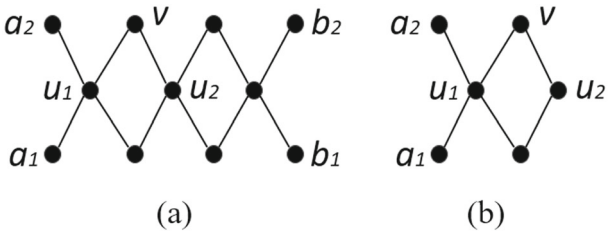

Fig. 5 **(a)** $\pi_{II}(G_v) = 1, \pi_{IO}(G_v) = \pi_{OI}(G_v) = 2, \pi_{OO}(G_v) = 3$. **(b)** $\pi_{II}(G_v) = 0, \pi_{IO}(G_v) = 1, \pi_{OI}(G_v) = \pi_{OO}(G_v) = 2$

or OI-strategy. Therefore $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v) = 2$. It is easy to see that 3 searchers cannot clear G_v in an OO-strategy. Hence $\pi_{OO}(G_v) = 4$.

(ii) We next show that the subcactus in Fig. 4b satisfies property (R_2). For an II-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Let λ_3 and λ_4 be two slide-in searchers moving from v to u_1 and u_2 respectively. Then move λ_3 and λ_4 to v', move λ_1 and λ_2 to b_1 and b_2 along two edge-disjoint paths respectively. So $\pi_{II}(G_v) \leq 2$. For an IO-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Let λ_3 be the slide-in searcher that moves from v to u_1. Then move λ_1 and λ_2 to b_1 and b_2 along two edge-disjoint paths respectively; when a searcher, say λ_2, is on v', move λ_3 to v', then to u_2 and back to v. Thus $\pi_{IO}(G_v) \leq 2$. For an OI-strategy, we place λ_1 on a_1, λ_2 on a_2, and λ_3 on v'. Then move λ_1 and λ_2 to b_1 and b_2 along two edge-disjoint paths respectively; when λ_2 (or λ_1) is on v', move λ_3 along the top 4-cycle to clear it. Hence $\pi_{OI}(G_v) \leq 3$. For an OO-strategy, we place λ_1 on a_1, λ_2 on a_2, and place λ_3 and λ_4 on v'. Then move λ_1 and λ_2 to b_1 and b_2 along two edge-disjoint paths respectively; when λ_2 (or λ_1) is on v', move λ_3 and λ_4 to v along the two edge-disjoint paths respectively. Thus $\pi_{OO}(G_v) \leq 4$. Note that one searcher cannot clear G_v in an II-strategy or IO-strategy, two searchers cannot clear G_v in an OI-strategy, and three searchers cannot clear G_v in an OO-strategy. Therefore $\pi_{II}(G_v) = \pi_{IO}(G_v) = 2$, $\pi_{OI}(G_v) = 3$ and $\pi_{OO}(G_v) = 4$.

(iii) If G_v is a cycle of length at least 4, then we have $\pi_{II}(G_v) = \pi_{IO}(G_v) = 0$. For an OI-strategy or OO-strategy, since one searcher cannot clear a cycle of length at least 4 but two searchers can, we know that $\pi_{OI}(G_v) = \pi_{OO}(G_v) = 2$.

(iv) Consider the subcactus G_v in Fig. 5 a. For an II-strategy, we place λ_1 on a_1. Let λ_2 and λ_3 be two slide-in searchers moving from v to u_1 and u_2 respectively. Then move λ_1 to u_1 and then to a_2, move λ_2 to u_2, and move λ_2 and λ_3 to b_1 and b_2 respectively. Note that $\pi_{II}(G_v) \geq 1$, and thus $\pi_{II}(G_v) = 1$. For an IO-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Let λ_3 be the slide-in searcher moving from v to u_2. Then move λ_1 and λ_2 to u_1, move λ_1 to u_2, move λ_2 to v, and move λ_1 and λ_3 to b_1 and b_2 respectively. It is easy to see that one searcher cannot clear G_v in an IO-strategy. So $\pi_{IO}(G_v) = 2$. For an OI-strategy, we place λ_1 on a_1 and place λ_2 on a_2. Then move λ_1 and λ_2 to u_1, move λ_1 to u_2, move λ_2 to v and then to u_2, and move λ_1 and λ_2 to b_1 and b_2 respectively. Thus $\pi_{OI}(G_v) \leq 2$. For an OO-strategy, we place searcher λ_1 on b_1 and place λ_2 on b_2. We place λ_3 on a_1. Then move λ_1 and λ_2 to u_2, move λ_2 to v, move λ_1 to u_1, move λ_3 to u_1, and move λ_1 and λ_3 to v and

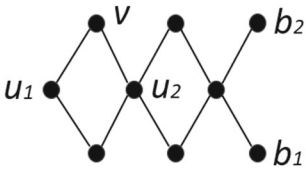

Fig. 6 $\pi_{II}(G_v) = 0$ and $\pi_{IO}(G_v) = \pi_{OI}(G_v) = \pi_{OO}(G_v) = 2$

a_2 respectively. Hence $\pi_{OO}(G_v) \leq 3$. Since one searcher cannot clear G_v in an OI-strategy, and two searchers cannot clear G_v in an OO-strategy, we have $\pi_{OI}(G_v) = 2$ and $\pi_{OO}(G_v) = 3$.

(v) Consider the subcactus G_v in Fig. 5b. Similarly to the above cases, we can show that $\pi_{II}(G_v) = 0$, $\pi_{IO}(G_v) = 1$ and $\pi_{OI}(G_v) = \pi_{OO}(G_v) = 2$.

(vi) We next show that the subcactus in Fig. 6 satisfies property (R_6). For an II-strategy, let λ_1 and λ_2 be two slide-in searchers on v, and move them to u_1 and u_2 respectively. Then move λ_1 to u_2 and then move λ_1 and λ_2 to b_1 and b_2 respectively. Thus $\pi_{II}(G_v) = 0$. For an IO-strategy, we place λ_1 and λ_2 on u_2. Let λ_3 be the slide-in searcher on v, and move this searcher to u_1, and then to u_2 and move back to v. Then move λ_1 and λ_2 to b_1 and b_2 respectively. So $\pi_{IO}(G_v) \leq 2$. For an OI-strategy, we place λ_1 on b_1 and place λ_2 on b_2. Then move λ_1 and λ_2 to u_2, move λ_2 to v and then to u_1, and move λ_1 to u_1. Hence $\pi_{OI}(G_v) \leq 2$. Similarly, for an OO-strategy, we place λ_1 on b_1 and place λ_2 on b_2. Then move λ_1 and λ_2 to u_2, move λ_2 to v, and move λ_1 to u_1 and then to v. So $\pi_{OO}(G_v) \leq 2$. Note that one searcher cannot clear G_v in an IO-strategy, OI-strategy, or OO-strategy. Therefore $\pi_{IO}(G_v) = \pi_{OI}(G_v) = \pi_{OO}(G_v) = 2$. □

5 Algorithm

In this section, we give an algorithm that can compute the fast search number of any cactus G. FastSearchCactus(G) (see Algorithm 1) is the main function, which invokes functions Profile1 and Profile3 to compute the profiles of subcacti.

Algorithm 1 FastSearchCactus(G)

Input: A cactus G.
Output: The fast search number of G.

1: If G is a tree, then call FS(G) in Dyer et al. (2008) and *output* fs(G). If G is a cycle, then fs(G) $= 2$ and *output* this number.
2: Arbitrarily select a cut vertex v in $V(G)$. Let $H_1, \ldots, H_k$ be all components in $G - v$. Let $G_i, 1 \leq i \leq k$, denote the subgraph of G induced by $V(H_i) \cup \{v\}$. Let E_v denote the edge set consisting of the edges connecting v and vertices in $V(H_1)$. Let H be the subgraph of G induced by $V(H_2) \cup \cdots \cup V(H_k) \cup V(E_v)$.
3: For each $i \in [k]$, call Profile1(G_i, v) and let $\mathcal{P}_{G_i}$ be its output.
4: Call Profile3(H, E_v, v, $\{\mathcal{P}_{G_2}, \ldots, \mathcal{P}_{G_k}\}$) and let $\mathcal{P}_H$ be its output.
5: For all possible combinations of the profiles from $\mathcal{P}_{G_1}$ and $\mathcal{P}_H$ with respect to sliding actions on the edges in E_v, let α be the minimum number among the number of searchers used in each of these combinations of profiles.
6: *Output* α.

PROFILE1 (see Algorithm 2) computes the profile of G_i defined in Algorithm 1. Note that G_i is a subcactus attached to v that is a cut vertex of G. The output of PROFILE1 is the profile of G_i.

Algorithm 2 PROFILE1$(G_i,\ v)$

1: If G_i is a tree, then compute $\pi_O(G_i)$ by calling FS(G) in Dyer et al. (2008). Let $\pi_I(G_i) = \pi_O(G_i) - 1$. *Output* $(\pi_I(G_i), \pi_O(G_i))$.
2: If G_i is a simple cycle, let $\pi_{II}(G_i) = \pi_{IO}(G_i) = 0$ and $\pi_{OI}(G_i) = \pi_{OO}(G_i) = 2$. *Output* $(\pi_{II}(G_i), \pi_{IO}(G_i), \pi_{OI}(G_i), \pi_{OO}(G_i))$.
3: If v is contained in a cycle of G_i, then *output* the profile of G_i, which is computed by PROFILE2$(G_i,\ v)$.
4: If v is a leaf of G_i, let $u \in V(G_i)$ be the vertex adjacent to v. *Output* the profile of G_i, which is computed by PROFILE1$(G_i - v,\ u)$.
5: Otherwise, for all subcacti $X_1, \ldots, X_m$ attached to v, compute their profiles $\mathcal{P}_{X_j}$ by calling PROFILE1$(X_j,\ v)$, $1 \le j \le m$; merge these profiles to obtain the profile of G_i by calling PROFILE3$(Y,\ E_v,\ v,\ \{\mathcal{P}_{X_1}, \ldots, \mathcal{P}_{X_m}\})$, where Y is an extension of the subgraph induced by the vertices of all X_j by adding two pendent edges to v which form the edge set E_v. *Output* the profile of G_i.

PROFILE2 (see Algorithm 3) is used to compute the profile of G_i in Step 3 of PROFILE1 when the cut-vertex v is contained in a cycle of G_i. The output of PROFILE2 is the profile of G_i.

Algorithm 3 PROFILE2$(G_i,\ v)$

1: Let $C = vu_1 \ldots u_{k'}v$ be the simple cycle in G_i that contains v. Let $X_{u_1}, \ldots, X_{u_{k'}}$ denote the components after deleting all edges of C from G_i. Let $E_{u_j} \subset E(C)$, $1 \le j \le k'$, be the set containing the two incident edges upon u_j. Let Y_{u_j} denote the graph obtained from X_{u_j} by attaching the two edges of E_{u_j} to u_j.
2: For each $j \in [k']$ compute the profile of Y_{u_j} as follows: Let $Z_1, \ldots, Z_{m_j}$ denote all components in $X_{u_j} - u_j$ and let Z'_ℓ, $1 \le \ell \le m_j$, be the subgraph of X_{u_j} induced by $V(Z_\ell) \cup \{u_j\}$; for each $\ell \in [m_j]$, call PROFILE1$(Z'_\ell,\ u_j)$ and let $\mathcal{P}_{Z'_\ell}$ be its output; call PROFILE3$(Y_{u_j},\ E_{u_j},\ u_j,\ \{\mathcal{P}_{Z'_1}, \ldots, \mathcal{P}_{Z'_{m_j}}\})$ and its output is the profile of Y_{u_j}.
3: Initially let $W = Y_{u_1}$ and $j = 2$.
4: Compute the profile of $W \cup Y_{u_j}$ with respect to the pendent edges on u_1 and u_j,
5: Update $W \leftarrow W \cup Y_{u_j}$ and $j \leftarrow j + 1$. If $j = k'$, similarly to Step 4, compute the profile of W and *output* this profile; otherwise, go to Step 4.

PROFILE3 (see Algorithm 4) is used for computing the profile of G'_v defined in Definition 4.1. Recall that G_v satisfies (R_i) if it has the i-th property in Theorem 4.4, where $1 \le i \le 7$. The input of PROFILE3 includes $G'(= G'_v)$, E_v (the pendent edge(s) added to v when G'_v is constructed from G_v), the vertex v (a cut vertex of G) and the set of profiles $\mathcal{P}$ (refer to Step 4 in Algorithm 1 or Step 2 in PROFILE2). The output of the algorithm is the profile of G'.

Algorithm 4 PROFILE3(G', E_v, v, $\mathcal{P}$)

1: If $|E_v| = 1$, then call FASTSEARCH($G' - E_v$, 1, 1, $\mathcal{P}$) and assign its output to $\pi_I(G')$. Let $\pi_O(G') = \pi_I(G') + 1$. *Output* $(\pi_I(G'), \pi_O(G'))$ that is the profile of G'.
2: If $|E_v| = 2$, then compute each component of the profile of G' as follows.

(i) Call FASTSEARCH($G' - E_v$, 2, 2, $\mathcal{P}$) and assign its output to $\pi_{II}(G')$.
(ii) Let $\pi_{OO}(G') = \pi_{II}(G') + 2$.
(iii) Call FASTSEARCH($G' - E_v$, 0, 1, $\mathcal{P}$) and assign its output to $\pi_{IO}(G')$.
(iv) If there are at least five edges in $E(G')$ incident with v, then call FASTSEARCH($G' - E_v$, 0, 0, $\mathcal{P}$) and assign its output to $\pi_{OI}(G')$; otherwise, for the subcacti whose profiles are $\mathcal{P}$, let r_i, $1 \le i \le 7$, be the number of those subcacti that satisfy (R_i).
  (a) If $1 \le r_7 \le 2$, let t be the minimum number of searchers required for clearing $G' - E_v$, such that all the subcacti of $G' - E_v$ attached to v and satisfying (R_7) are cleared by O-strategies. Let $\pi_{OI}(G') = t + 2 - r_7$, and go to Step v.
  (b) If there is some i, where $2 \le i \le 6$, such that $r_i = 1$, let t be the minimum number of searchers required for clearing $G' - E_v$, such that the subcacti of $G' - E_v$ satisfying (R_i) is cleared by an OO-strategy. Let $\pi_{OI}(G') = t$, and go to Step v.
  (c) If $r_1 = 1$, let t be the minimum number of searchers required for clearing $G' - E_v$, such that the subcacti of $G' - E_v$ satisfying (R_1) is cleared by an OI-strategy. Let $\pi_{OI}(G') = t + 1$.
(v) *Output* $(\pi_{II}(G'), \pi_{IO}(G'), \pi_{OI}(G'), \pi_{OO}(G'))$, the profile of G'.

FASTSEARCH (see Algorithm 5) is called by PROFILE3 as a subroutine, which computes the total number of searchers for clearing $G' - E_v$ under some specific setting determined by σ_1 and σ_2. Let $\mathcal{P}$ be the set containing the profiles of all the subcacti of $G' - E_v$ attached to v (refer to Step 4 in Algorithm 1 or Step 2 in PROFILE2). σ_1 and σ_2 are defined in the following two paragraphs.

We first explain σ_1. For the case when $|E_v| = 1$, let x_1 be the minimum number of searchers required for clearing G' where the edge in E_v is cleared by a slide-in action, and let $\mathcal{S}_1$ be the corresponding strategy of G'. Let $\mathcal{S}'_1$ be the strategy obtained from $\mathcal{S}_1$ by replacing the slide-in action on the edge in E_v with a placing action on v. Clearly, $\mathcal{S}'_1$ clears $G' - E_v$. Hence, x_1 is equal to the minimum number of searchers required for clearing $G' - E_v$ where v is initially placed at least one searcher. For the case when $|E_v| = 2$, let x_2 be the minimum number of searchers required for clearing G' where the two edges in E_v are cleared both by slide-in actions. Obviously, x_2 is equal to the minimum number of searchers required for clearing $G' - E_v$ where at least two searchers are initially placed on v. Hence, we can always convert a slide-in action on an incident edge of v to a placing action on v. In a similar way, a slide-out action on an incident edge of v can be converted to the removal of a placing action on v. For convenience, we use σ_1 to denote the number of searchers placed on v, which is equal to the sum of (1) the number of searchers placed on v, and (2) the number of incident edges cleared by slide-in actions minus the number of incident edges cleared by slide-out actions. Note that in the process of assigning strategies to subcacti of $G' - E_v$, the difference between the numbers of incident edges that are cleared by slide-in actions and slide-out actions may change dynamically. Hence, σ_1 can change dynamically throughout the process of assigning strategies to subcacti of $G' - E_v$.

We now explain σ_2. Let σ_2 be the sum of (1) the number of searchers placed on v, and (2) the number of slide-out actions in OO-strategies and OI-strategies that are already assigned to subcacti of $G' - E_v$. Similarly, σ_2 can change dynamically while

we are assigning strategies to subcacti of $G' - E_v$. We use σ_2 to help determine if IO-strategies can be assigned to subcacti of $G' - E_v$ without placing additional searchers on v. If $\sigma_2 \geq 2$, then this is sufficient for us to assign IO-strategies to any number of subcacti of $G' - E_v$. For simplicity, σ_2 is always set to 2 if the actual value of σ_2 is larger than 2.

Algorithm 5 FASTSEARCH$(G, \sigma_1, \sigma_2, \mathcal{P})$

1: Let $\mathcal{G}_v$ be the set of subcacti corresponding to the profiles in $\mathcal{P}$. For each $1 \leq i \leq 7$, find r_i, the number of subcacti in $\mathcal{G}_v$ which satisfy (R_i).
2: Find non-negative integers s_6^1, s_6^2, s_7^1 and s_7^2 satisfying that (1) $s_6^1 + s_6^2 = r_6$; (2) $s_7^1 + s_7^2 = r_7$; (3) $\sigma_1 + s_7^2 - s_7^1 + 2(s_6^2 - s_6^1)$ has the minimum non-negative value.
3: Let s_6^1 subcacti satisfying (R_6) be cleared by II-strategies; let s_6^2 subcacti satisfying (R_6) be cleared by OO-strategies. Let s_7^1 subcacti satisfying (R_7) be cleared by I-strategies; let s_7^2 subcacti satisfying (R_7) be cleared by O-strategies.
4: Update $\sigma_1 \leftarrow \sigma_1 + s_7^2 - s_7^1 + 2(s_6^2 - s_6^1)$, and $\sigma_2 \leftarrow \min\{\sigma_2 + s_7^2 + 2s_6^2, 2\}$.
5: List all feasible strategy assignments to the subcacti of G which have not been assigned strategies, where the following conditions are satisfied.

(a) If $\sigma_2 \leq 1$, then at most one searcher is placed on v or at most one subcactus is assigned an OO-strategy; if $\sigma_2 = 2$, then no subcactus is assigned an OO-strategy and no searcher is placed on v.
(b) If a subcactus is assigned an OO-strategy in (a), then update $\sigma_1 \leftarrow \sigma_1 + 2$ and $\sigma_2 \leftarrow 2$; if a searcher is placed on v in (a), then update $\sigma_1 \leftarrow \sigma_1 + 1$ and $\sigma_2 \leftarrow \min\{\sigma_2 + 1, 2\}$.
(c) If $\sigma_1 \geq 2$, then at most one subcactus is assigned an II-strategy; if there is one subcactus being assigned an II-strategy, then update $\sigma_1 \leftarrow \sigma_1 - 2$.
(d) For each of the remaining subcacti that has not been assigned a strategy: if the subcactus satisfies (R_1) or (R_4), then assign an OI-strategy to the subcactus; if the subcactus satisfies (R_2), (R_3) or (R_5), then assign an IO-strategy to the subcactus.

6: Find the strategy from the feasible strategy assignments in Step 5 which uses the minimum number of searchers. *Output* this minimum number.

6 Correctness and running time

We say that two fast search strategies for a graph are *equivalent* if both of them use the same number of searchers to clear the graph. From this definition, we know that a fast search strategy and its reverse are equivalent.

Lemma 6.1 *Let G be a cactus that contains a cut-vertex v, and let $\mathcal{G}_v$ be the set of all subcacti attached to v. For any optimal fast search strategy for G, there exists an equivalent strategy such that the subcacti in $\mathcal{G}_v$ are cleared in the following order:*

1. *All subcacti that are cleared by an O-strategy or OO-strategy;*
2. *All subcacti that are cleared by an OI-strategy (for each of these subcacti, perform all actions in the strategy for this cactus until exactly one of the two edges incident on v is cleared);*
3. *All subcacti that are cleared by an IO-strategy;*
4. *All subcacti that are cleared by an OI-strategy (for each of these subcacti, perform all remaining actions in the strategy for this cactus);*
5. *All subcacti that are cleared by an I-strategy or II-strategy.*

Proof Let $\mathcal{S}$ be an optimal fast search strategy that clears G using $\mathrm{fs}(G)$ searchers. Since G is a cactus and v is a cut vertex, every subcactus attached to v has one or two edges incident with v. We will transform $\mathcal{S}$ into an equivalent strategy $\mathcal{S}'$ satisfying the ordering in the lemma. For a $G_v \in \mathcal{G}_v$, we have the following cases for clearing it by $\mathcal{S}$.

CASE 1. If only one edge of G_v is incident on v and this edge is cleared by a slide-out action in $\mathcal{S}$, then the searchers to clear G_v by $\mathcal{S}$ will stay in G_v except the searcher who slides out of G_v. Thus we can group the actions of $\mathcal{S}$ for clearing G_v into an O-strategy. Similarly, if two edges of G_v are incident on v and they are cleared by two slide-out actions in $\mathcal{S}$, we can group the actions of $\mathcal{S}$ for clearing G_v into an OO-strategy.

CASE 2. If two edges of G_v are incident on v, where one of them, say uv, is cleared by a slide-out action in $\mathcal{S}$ before the other is cleared by a slide-in action, then we can group the actions of $\mathcal{S}$ for clearing G_v into two parts of an OI-strategy: the first part consists of actions of $\mathcal{S}$ on G_v that are performed before and including the slide-out action on uv, and the second part consists of actions on G_v that are performed after the slide-out action on uv.

CASE 3. If two edges of G_v are incident on v, where one of them is cleared by a slide-in action in $\mathcal{S}$ before the other is cleared by a slide-out action, then we can group the actions of $\mathcal{S}$ for clearing G_v into an IO-strategy.

CASE 4. If only one edge of G_v is incident on v and this edge is cleared by a slide-in action in $\mathcal{S}$, then all searchers to clear G_v by $\mathcal{S}$ will stay in G_v. Thus we can group the actions of $\mathcal{S}$ for clearing G_v into an I-strategy. Similarly, if two edges of G_v are incident on v and they are cleared by two slide-in actions in $\mathcal{S}$, we can group the actions of $\mathcal{S}$ for clearing G_v into an II-strategy.

For the groups of actions in the above cases, we can arrange these groups in the following ordering:

1. Each group that can be considered as an O-strategy or OO-strategy for clearing a subcactus of $\mathcal{G}_v$;
2. The first part of each group that can be considered as an OI-strategy for clearing a subcactus of $\mathcal{G}_v$;
3. Each group that can be considered as an IO-strategy for clearing a subcactus of $\mathcal{G}_v$;
4. The second part of each group that can be considered as an OI-strategy for clearing a subcactus of $\mathcal{G}_v$;
5. Each group that can be considered as an I-strategy or II-strategy for clearing a subcactus of $\mathcal{G}_v$.

Since in $\mathcal{S}$, the searchers to clear a subcactus $G_v \in \mathcal{G}_v$ stay in G_v except those who slide out of G_v, we know that the groups in the above ordering form a fast search strategy $\mathcal{S}'$ that clears G using $\mathrm{fs}(G)$ searchers. Thus $\mathcal{S}'$ and $\mathcal{S}$ are equivalent. □

Lemma 6.2 *Let G be a cactus that contains a cut-vertex v, and let $\mathcal{G}_v$ be the set of all subcacti attached to v. For any optimal fast search strategy for G, there exists an equivalent strategy such that for each $G_v \in \mathcal{G}_v$,*

(i) *If G_v satisfies (R_1), then it is cleared by an OI-strategy or OO-strategy;*
(ii) *If G_v satisfies (R_2), then it is cleared by an IO-strategy, OI-strategy or OO-strategy;*

(iii) *If G_v satisfies (R_3), then it is cleared by an IO-strategy or OO-strategy;*
(iv) *If G_v satisfies (R_4), then it is cleared by an II-strategy, OI-strategy or OO-strategy;*
(v) *If G_v satisfies (R_5), then it is cleared by an II-strategy, IO-strategy or OO-strategy;*
(vi) *If G_v satisfies (R_6), then it is cleared by an II-strategy or OO-strategy.*

Proof (i) Suppose that G_v satisfies (R_1) and $\mathcal{S}_1$ is an optimal fast search strategy for G in which G_v is cleared by an II-strategy or IO-strategy. Let vv_1 and vv_2 be the two edges incident on v in G_v. Without loss of generality, assume that vv_1 is the first edge in G_v that is cleared by $\mathcal{S}_1$. By making the following modifications, $\mathcal{S}_1$ is converted to an equivalent strategy for G in which G_v is cleared by an OI-strategy:

1. Let t denote the moment in $\mathcal{S}_1$ after which the next sliding action clears vv_1.
2. Remove all sliding actions from $\mathcal{S}_1$ that clear edges in G_v; remove all placing actions on vertices in $V(G_v - v)$.
3. Choose an OI-strategy for G_v, insert all its placing actions on $V(G_v - v)$ at the beginning of the strategy, and insert all its sliding actions immediately after t.

Since G_v satisfies (R_1), we have $\pi_{II}(G_v) = \pi_{IO}(G_v) = \pi_{OI}(G_v)$. Thus the modified strategy is equivalent to $\mathcal{S}_1$. Note that if $\mathcal{S}_1$ is a strategy for G in which G_v is cleared by an OO-strategy, then we do not modify $\mathcal{S}_1$. Hence, if G_v satisfies (R_1), then there is an optimal strategy for G such that G_v is cleared by an OI-strategy or OO-strategy.

(ii) If G_v satisfies (R_2), then $\pi_{II}(G_v) = \pi_{IO}(G_v)$. Similar to (i), there is an optimal strategy for G such that G_v is cleared by an IO-strategy, OI-strategy or OO-strategy.

(iii) If G_v satisfies (R_3), then $\pi_{II}(G_v) = \pi_{IO}(G_v)$ and $\pi_{OI}(G_v) = \pi_{OO}(G_v)$. So, if $\mathcal{S}_1$ is an optimal strategy for G in which G_v is cleared by an II-strategy (resp. OI-strategy), then we can modify $\mathcal{S}_1$ to obtain an equivalent strategy such that G_v is cleared by an IO-strategy (resp. OO-strategy).

(iv) If G_v satisfies (R_4), then $\pi_{IO}(G_v) = \pi_{OI}(G_v)$. Similar to (i), there is an optimal strategy for G such that G_v is cleared by an II-strategy, OI-strategy or OO-strategy.

(v) If G_v satisfies (R_5), then $\pi_{OI}(G_v) = \pi_{OO}(G_v)$. Thus there is an optimal strategy for G such that G_v is cleared by an II-strategy, IO-strategy or OO-strategy.

(vi) If G_v satisfies (R_6), then $\pi_{IO}(G_v) = \pi_{OI}(G_v) = \pi_{OO}(G_v)$. Hence, there is an optimal strategy for G such that G_v is cleared by an II-strategy or OO-strategy. □

Lemma 6.3 FASTSEARCH*$(G, \sigma_1, \sigma_2, \mathcal{P})$ computes the minimum number of searchers for clearing G under the given setting in* PROFILE3.

Proof When FASTSEARCH is called, the cactus $G'_v - E_v$, where G'_v is defined in Definition 4.1, is passed to G along with the set $\mathcal{P}$ of profiles of the subcacti attached to the cut vertex v. Note that there are only four different settings for σ_1 and σ_2, that is, $(\sigma_1 = 1, \sigma_2 = 1)$, $(\sigma_1 = 2, \sigma_2 = 2)$, $(\sigma_1 = 0, \sigma_2 = 1)$ and $(\sigma_1 = 0, \sigma_2 = 0)$. We will show that any optimal fast search strategy for G can be converted into an equivalent strategy used in FASTSEARCH$(G, \sigma_1, \sigma_2, \mathcal{P})$, which implies that the fast search strategy used in FASTSEARCH is optimal under the given conditions on σ_1 and σ_2.

Let $\mathcal{G}_v$ be the set of subcacti corresponding to the profiles in $\mathcal{P}$. We first prove the correctness of Steps 2–4. From Lemmas 6.1 and 6.2, there must exist an optimal fast search strategy for G, denoted as $\mathcal{S}$, satisfying that: (1) all the subcacti in $\mathcal{G}_v$ are cleared in the order given in Lemma 6.1, and (2) each subcactus of $\mathcal{G}_v$ is cleared by a strategy compatible with Lemma 6.2. In $\mathcal{S}$, there must exist a moment at which the number of searchers on v is at least $\sigma_2 + s_7^2 + 2s_6^2$; otherwise, searchers are insufficient for clearing all subcacti of $\mathcal{G}_v$ that satisfy (R_6) and (R_7). Hence, we can modify the strategy for $\mathcal{G}_v$ in $\mathcal{S}$, if necessary, such that the modified strategy has the following properties:

1. s_6^1 subcacti satisfying (R_6) are cleared by II-strategies, and s_6^2 subcacti satisfying (R_6) are cleared by OO-strategies.
2. s_7^1 subcacti satisfying (R_7) are cleared by I-strategies, and s_7^2 subcacti satisfying (R_7) are cleared by O-strategies.

From Lemma 6.1, the modified strategy uses the same number of searchers to clear G. For convenience, we still use $\mathcal{S}$ to denote the modified strategy.

We now prove the correctness of Step 5. Note that both σ_1 and σ_2 are updated in Step 4. We first prove that when $\sigma_2 = 2$ and $\sigma_1 \leq 1$, the following strategy assignment is optimal:

(A) Assign OI-strategies to subcacti that satisfy (R_1) and (R_4);
(B) Assign IO-strategies to subcacti that satisfy (R_2), (R_3) and (R_5).

Since $\sigma_2 = 2$, there must exist a moment in $\mathcal{S}$ at which v is occupied by at least two searchers. Hence, we can directly assign IO-strategy to any number of subcacti attached to v. From Lemmas 6.1 and 6.2, the above strategy assignment to subcacti satisfying (R_1), (R_2) and (R_3) is optimal. We then consider the strategy assignment to subcacti satisfying (R_4) and (R_5). If there exists a subcactus $X \in \mathcal{G}_v$ that (1) satisfies (R_4) or (R_5) and (2) is cleared by an II-strategy, then either there must exist a subcactus that is cleared by an OO-strategy in $\mathcal{S}$, or at least one searcher is placed on v in $\mathcal{S}$. Although assigning an II-strategy to X seems to help save one searcher, using OO-strategy or placing an additional searcher would cost at least one more searcher. Hence, the strategy assignment to subcacti satisfying (R_4) and (R_5) is optimal.

In the next, we prove that an optimal strategy assignment, which uses the minimum number of searchers, contains at most one OO-strategy. Assume that A_1 is a strategy assignment that contains at least two OO-strategies. It is easy to show that there also exists a subcactus being cleared by an II-strategy or some other strategy that uses the same number of searchers as the II-strategy. We select this subcactus and a subcatus being cleared by an OO-strategy. For each of the remaining subcacti of G, we assign a strategy to it according to the strategy assignment described in the previous paragraph. Let A_2 be the new strategy assignment after taking the above modifications. Obviously, A_2 is still a feasible strategy assignment, which uses the same or less number of searchers than A_1. Similarly, we can show that when $\sigma_2 \leq 1$, at most one searcher is placed on v since adopting an OO-strategy is always better than placing two searchers on v. Further, if an OO-strategy is assigned to some subcactus, then we know there will exist a moment at which v is occupied by at least two searchers. In this case, we can show that there is no need to place additional searchers on v.

Consider the case when $\sigma_2 = 0$. Note that a feasible strategy assignment must (1) contain at least one OO-strategy, (2) contain at least two OI-strategies, or (3) contain at least one OI-strategy and a searcher is initially placed on v. Consider a feasible strategy assignment. If there exists one subcactus satisfying (R_2) that is assigned an OI-strategy, then we can modify the strategy assignment by (1) letting the subcactus cleared by an IO-strategy, and (2) placing one additional searcher on v. Obviously, the modified strategy assignment is still a feasible strategy assignment and uses the same number of searchers. If there exist two subcacti satisfying (R_2) that are assigned OI-strategies, then we can modify the strategy assignment by (1) letting one of the subcacti cleared by an OO-strategy, and (2) letting the other subcactus cleared by an IO-strategy. It is easy to show that the modified strategy assignment is still a feasible strategy assignment and uses the same number of searchers. In a similar way, we can modify the strategy assignment such that each subcactus satisfying (R_i), where $1 \leq i \leq 5$, is assigned (1) a strategy according to the strategy assignment described in previous paragraph, or (2) an OO-strategy or an II-strategy if allowed by Lemma 6.2. Hence, we can easily show that an optimal feasible strategy assignment must exist in the enumerated feasible strategy assignments. Thus, we can find the one with the minimum number of searchers and output this number in Step 6. □

Lemma 6.4 PROFILE3*(G', E_v, v, $\mathcal{P}$) computes the profile of G'.*

Proof We first prove the correctness of Step 1. Since $|E_v| = 1$, G' satisfies (R_7). Consider the case when the pendent edge incident to v is cleared by a slide-in action. By definition, we have $\sigma_1 = 1$ and $\sigma_2 = 1$. Let $\pi_1(G' - E_v)$ be the minimum number of searchers for clearing $G' - E_v$, in which at least one searcher is initially placed on v. It is easy to see that $\pi_I(G') = \pi_1(G' - E_v)$. It follows from Lemma 6.3 that FASTSEARCH$(G', \sigma_1, \sigma_2, \mathcal{P})$ computes the minimum number of searchers for clearing G' under the given setting described by σ_1 and σ_2. By calling FASTSEARCH$(G', 1, 1, \mathcal{P})$, we can obtain $\pi_I(G')$. From Theorem 4.4, we have $\pi_O(G') = \pi_I(G') + 1$.

In the rest of this proof, we show the correctness of Step 2.

(i) Consider the case where the two pendent edges incident to v are both cleared by slide-in actions. After the two pendent edges are cleared, the two searchers on v can be used to clear subcacti attached to v. Further, we know there must exist a moment in the strategy of G' at which v is occupied by at least two searchers. Hence, $\sigma_1 = 2$ and $\sigma_2 = 2$. Let $\pi_2(G' - E_v)$ be the minimum number of searchers for clearing $G' - E_v$ where at least two searchers are initially placed on v. It is easy to see that $\pi_{II}(G') = \pi_2(G' - E_v)$. By calling FASTSEARCH$(G', 2, 2, \mathcal{P})$, we can obtain $\pi_{II}(G')$.

(ii) It follows from Theorem 4.4 that $\pi_{OO}(G') = \pi_{II}(G') + 2$.

(iii) Consider the case where one of the two pendent edges incident to v is cleared by a slide-in action, and later the other edge is cleared by a slide-out action. We know there must exist a moment at which v is occupied by a searcher, and $\sigma_2 = 1$. Let $\pi_3(G' - E_v)$ be the minimum number of searchers for clearing $G' - E_v$ where one searcher is initially placed on v and cannot move at any moment. So $\pi_{IO}(G') = \pi_3(G' - E_v)$. As the searcher placed on v cannot be used to clear any contaminated edge of subcacti

attached to v, we have $\sigma_1 = 0$. By calling FastSearch(G', 0, 1, $\mathcal{P}$), we can obtain $\pi_{IO}(G')$.

(iv) Consider the case where one of the two pendent edges incident to v is cleared by a slide-out action, and later the other edge is cleared by a slide-in action. Note that for any strategy of G', there must exist a moment at which v is occupied by at least two searchers. If v has at least five incident edges, we know for any strategy of $G' - E_v$, there must exist a moment at which v is occupied by at least two searchers. Let $\pi_4(G' - E_v)$ be the minimum number of searchers for clearing $G' - E_v$. Thus $\pi_{OI}(G') = \pi_4(G' - E_v)$. Hence, by calling FastSearch(G', 0, 0, $\mathcal{P}$), we can obtain $\pi_{OI}(G')$. If v has at most four incident edges, then consider the subcacti attached to v.

(a) Consider the case where v has exactly one subcactus and this subcactus satisfies (R_7), that is $r_7 = 1$. If the subcactus is cleared by an I-strategy, then at least two searchers must be placed on v initially. If the subcactus is cleared by an O-strategy, then at least one searcher must be placed on v initially. Clearly, no matter which strategy is assigned to the subcactus, the minimum number of searchers for clearing G' remain the same. Hence, $\pi_{OI} = t + 2 - r_7 = t + 1$. If v has exactly two subcacti and both of them satisfy (R_7), i.e., $r_7 = 2$, then similarly, we can show that $\pi_{OI} = t + 2 - r_7 = t$.

(b) Consider the case where v has exactly one subcactus that satisfies (R_i), $2 \leq i \leq 6$. Note that t is the minimum number of searchers used by an OO-strategy for clearing the subcactus. It is easy to verify that no matter which strategy is assigned to the subcactus, since v is required to contain two searchers at some moment in any strategy of $G' - E_v$, the number of searchers for clearing $G' - E_v$ is at least t.

(c) Consider the case where v has exactly one subcactus and this subcactus satisfies (R_1). Note that an OO-strategy of the subcactus uses two more searchers than an OI-strategy. Hence, it is easy to verify that any strategy for $G' - E_v$ uses at least $t + 1$ searchers. □

Theorem 6.5 *For any cactus G, Algorithm 1 computes* fs(G).

Proof FastSearchCactus(G) is the main function. If G is a tree, then we can call FS(G) in Dyer et al. (2008) to find fs(G). If G is a cycle, then fs(G) = 2. So in the remainder of this proof, we suppose G is neither a tree nor a cycle. In Algorithm 1, we first pick a cut-vertex v and find all subcacti G_i attached to v. We then invoke function Profile1(G_i, v) to compute the profile $\mathcal{P}_{G_i}$ of each subcactus G_i; we also invoke function Profile3(H, E_v, v, $\{\mathcal{P}_{G_2}, \ldots, \mathcal{P}_{G_k}\}$) to compute the profile $\mathcal{P}_H$ of the union H of subcacti.

In Profile1(G_i, v), if G_i is a tree, since v is a leaf of G_i, we know that there is an optimal fast search strategy such that a searcher will end on v. So fs(G_i) = $\pi_O(G_i)$; and thus $\pi_O(G_i)$ can be computed by calling FS(G) in Dyer et al. (2008). Notice that a fast search strategy and its reverse use the same number of searchers to clear G_i. Since the slide-in searcher in an I-strategy is not counted, we have $\pi_I(G_i) = \pi_O(G_i) - 1$. If G_i is a simple cycle, it is easy to see that $\pi_{II}(G_i) = \pi_{IO}(G_i) = 0$ and $\pi_{OI}(G_i) = \pi_{OO}(G_i) = 2$. If v is contained in a cycle of G_i, we invoke function Profile2 to compute the profile of G_i. If v is a leaf of G_i, we delete v and call Profile1 recursively. Otherwise, we invoke Profile1 for each subcactus attached to v, and then merge them by calling Profile3.

In PROFILE2(G_i, v), v is contained in a simple cycle $C = vu_1 \dots u_{k'}v$ in G_i. Let X_{u_j}, $1 \le j \le k'$, be a subcactus attached on u_j and Y_{u_j} be an extension of X_{u_j}. Let $E_{u_j} \subset E(C)$, $1 \le j \le k'$, be the set containing the two incident edges upon u_j. We then invoke functions PROFILE1 and PROFILE3 to compute the profile of Y_{u_j}. Let $W = Y_{u_1}$ and $j = 2$ initially. In Step 4 of PROFILE2(G_i, v), since W and Y_{u_j} have one edge in common, a strategy for $W \cup Y_{u_j}$ can be obtained from strategies for W and Y_{u_j} by reaching an accord on the sliding action on the common edge of W and Y_{u_j}. Note that in the graph $W \cup Y_{u_j}$, u_1 and u_j have one pendent edge in $E(C)$ respectively. Compute the profile of $W \cup Y_{u_j}$ with respect to the sliding actions on the pendent edges on u_1 and u_j. Then in Step 5, we update $W \leftarrow W \cup Y_{u_j}$ and $j \leftarrow j + 1$. If $j = k'$, we can compute the profile of W in a way similar to Step 4 mentioned above. Note that this W is the same as G_i; so we output the profile of W.

It follows from Lemmas 6.4 and 6.3 that PROFILE3(G', E_v, v, $\mathcal{P}$), together with FASTSEARCH(G', σ_1, σ_2, $\mathcal{P}$), can compute the profile of G' when a set of profiles $\mathcal{P}$ is given.

From the above, we know that Algorithm 1 can compute the fast search number of any cactus. □

We now analyze the running time of our algorithms in this section.

Theorem 6.6 *For any cactus G, the fast search number of G can be computed in linear time.*

Proof In Algorithm 1, Step 1 takes linear time to find fs(G) by calling FS(G) in Dyer et al. (2008). Step 2 takes linear time to construct G_i and H. The running time of Steps 3 and 4 depend on PROFILE1 and PROFILE3. In Step 5, it takes constant time to list all possible combinations of the profiles from $\mathcal{P}_{G_1}$ and $\mathcal{P}_H$ with respect to sliding actions on the edges in E_v because $|E_v| \le 2$. For each combination of these profiles, it takes constant time to find the number of searchers required. Thus, Step 5 takes constant time.

In PROFILE1, if G_i is a tree, Step 1 takes linear time to find $\pi_O(G_i)$ and $\pi_I(G_i)$ due to FS(G) in Dyer et al. (2008). If G_i is a simple cycle, it is easy to see that Step 2 takes constant time. The running time of Steps 3 depends on PROFILE2. Step 4 is a recursive call of PROFILE1. The running time of Step 5 mainly depends on PROFILE3. Although there is a recursion in PROFILE1, its running time is still linear because the structure of the computation is tree-like and has the following properties:

1. the profile of a subcactus attached to any vertex in $V(G)$ has constant size;
2. the profile of any subcactus attached to a vertex in $V(G)$ is computed at most once;
3. the profile of a subcactus attached to a vertex in $V(G)$ is passed as a parameter at most once when computing the profile of another subcactus;
4. the computation of the profile of a subcactus attached to a vertex in $V(G)$ takes constant time.

In PROFILE2, Step 1 takes linear time to construct Y_{u_j}. The running time of Step 2 mainly depends on PROFILE1 and PROFILE3. In Steps 4 and 5, it takes constant time to find the profile of $W \cup Y_{u_j}$.

In PROFILE3, the running time of Step 1 depends on FASTSEARCH($G' - E_v, 1, 1, \mathcal{P}$), and the running time of Step 2 depends on FASTSEARCH($G' - E_v, 2, 2, \mathcal{P}$), FASTSEARCH($G' - E_v, 0, 1, \mathcal{P}$) and FASTSEARCH($G' - E_v, 0, 0, \mathcal{P}$).

In FASTSEARCH($G, \sigma_1, \sigma_2, \mathcal{P}$), Step 1 takes linear time to find r_i. Step 2 also takes linear time to find s_6^1, s_6^2, s_7^1 and s_7^2. It takes constant time to update σ_1 and σ_2 in Step 4. Steps 5–6 takes linear time to list all the feasible strategy assignments and to find the strategy from these assignments which uses the minimum number of searchers.

Overall, Algorithm 1 computes fs(G) in linear time. □

7 Conclusion

In this paper, we introduced the notion of k-combinable graphs and proposed a method for computing the fast search number of these graphs. Using this method, a linear-time algorithm for computing the fast search number for cacti graphs, along with rigorous analysis, is presented.

Funding Boting Yang: Research supported in part by an NSERC Discovery Research Grant, Application No.: RGPIN-2018-06800. Sandra Zilles: Research supported in part by an NSERC Discovery Research Grant, Application No.: RGPIN-2017-05336. Lusheng Wang: Research supported by National Science Foundation of China (NSFC: 61972329) and GRF grants for Hong Kong Special Administrative Region, P. R. China (CityU 11210119 and CityU 11206120).

Data availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors declare that they have no competing interests.

References

Alspach B (2006) Sweeping and searching in graphs: a brief survey. Matematiche 59:5–37
Bienstock D (1991) Graph searching, path-width, tree-width and related problems (a survey). DIMACS Ser Discr Math Theoret Comput Sci 5:33–49
Bienstock D, Seymour P (1991) Monotonicity in graph searching. J Algorithms 12:239–245
Bonato A, Nowakowski R (2011) The Game of Cops and Robbers on Graphs. American Mathematical Society, Providence, Rhode Island
Bonato A, Yang B (2013) Graph searching and related problems. In: Pardalos P, Du D-Z and Graham R (eds) Handbook of Combinatorial Optimization, pp 1511–1558. Springer, 2nd edition
Dereniowski D, Diner Ö, Dyer D (2013) Three-fast-searchable graphs. Discr Appl Math 161:1950–1958
Dyer D, Yang B, Yaşar Ö (2008) On the fast searching problem. In: Algorithmic Aspects in Information and Management, pp 143–154. Springer
Fomin FV, Petrov NN (1996) Pursuit-evasion and search problems on graphs. Congressus Numerantium, pp 47–58
Fomin FV, Thilikos DM (2008) An annotated bibliography on guaranteed graph searching. Theor Comput Sci 399:236–245
Hahn G (2007) Cops, robbers and graphs. Tatra Mt Math Publ 36:163–176
Kirousis LM, Papadimitriou CH (1986) Searching and pebbling. Theor Comput Sci 47:205–218
LaPaugh A (1993) Recontamination does not help to search a graph. J ACM 40:224–245
Makedon FS, Papadimitriou CH, Sudborough IH (1985) Topological bandwidth. SIAM J Algebr Discr Methods 6:418–444

Megiddo N, Hakimi SL, Garey MR, Johnson DS, Papadimitriou CH (1988) The complexity of searching a graph. J ACM 35:18–44
Parsons TD (1976) Pursuit-evasion in a graph. In: Proceedings of the International Conference on the Theory and Applications of Graphs, Lecture Notes in Mathematics, pp 426–441. Springer-Verlag
Stanley D, Yang B (2011) Fast searching games on graphs. J Comb Optim 22:763–777
Xue Y, Yang B (2017) The fast search number of a cartesian product of graphs. Discr Appl Math 224:106–119
Xue Y, Yang B, Zhong F, Zilles S (2018) The fast search number of a complete k-partite graph. Algorithmica 80:3959–3981
Yang B (2011) Fast edge searching and fast searching on graphs. Theoret Comput Sci 412:1208–1219