Fast Searching on k-Combinable Graphs

Yuan Xue, Boting Yang(✉), and Sandra Zilles

Department of Computer Science, University of Regina, Regina, Canada
{xue228,boting,zilles}@cs.uregina.ca

Abstract. Finding an optimal fast search strategy for graphs is challenging, sometimes even when graphs have very small treewidth, like cacti, cartesian product of a tree and an edge, etc. However, it may be easier to find an optimal fast search strategy for some critical subgraphs of the given graph. Although fast searching is not subgraph-closed, this observation still motivates us to establish relationships between optimal fast search strategies for a graph and its subgraphs. In this paper, we introduce the notion of k-combinable graphs and propose a new method for computing their fast search number. Assisted by the new method, we investigate the fast search number of cacti graphs and the cartesian product of a tree and an edge. Algorithms for producing fast search strategies for the above graphs, along with rigorous proofs, are given in this paper.

1 Introduction

Inspired by an article of Breisch [3] who considered the problem of finding a lost explorer in dark complex caves, Parsons [9] first introduced the graph search problem in which both searchers and fugitive move continuously along edges of a graph. Motivated by applied problems in the real world and theoretical issues in computer science and mathematics, graph searching has become a hot topic. It has many models, such as edge searching, node searching, mixed searching, fast searching, etc. These models are basically defined by the class of graphs, the actions of searchers and fugitives, visibility of fugitives, and conditions on what constitutes capture [1,2,6,8].

Given a graph that contains an invisible fugitive, the fast search problem is to find the fast search number, i.e., the minimum number of searchers to capture the fugitive in the fast search model. This model was first introduced by Dyer, Yang and Yaşar [5] in 2008. Let G denote an undirected graph. In the fast search model, a fugitive hides either on vertices or on edges of G. The fugitive can move at a great speed at any time from one vertex to another along a path that contains no searchers. We call an edge *contaminated* if it may contain the fugitive, and we call an edge *cleared* if we are certain that it does not contain the fugitive. In order to capture the fugitive, one launches a set of searchers on some vertices of the graph; these searchers then clear the graph edge by edge while at the same time guarding the already cleared parts of the graph. There are two actions for searchers: placing and sliding. An edge is cleared by a sliding action and every edge must be traversed exactly once. A *fast search strategy* for a graph is a sequence of actions of searchers that clear all contaminated edges of the graph. The *fast search number* of G, denoted by $\mathrm{fs}(G)$, is the smallest number of searchers needed to capture the fugitive in G. For more details about the model setting, please refer to [5].

Q. Ni and W. Wu (Eds.): AAIM 2022, LNCS 13513, pp. 394–405, 2022.
https://doi.org/10.1007/978-3-031-16081-3_34

Dyer et al. [5] proposed a linear time algorithm for computing the fast search number of trees. Stanley and Yang [10] gave a linear time algorithm for computing the fast search number of Halin graphs and their extensions. They also presented a quadratic time algorithm for computing the fast search number of cubic graphs, while the problem of finding the node search number of cubic graphs is NP-complete [7]. Yang [13] proved that the problem of finding the fast search number of a graph is NP-complete; and it remains NP-complete for Eulerian graphs. He also proved that the problem of determining whether the fast search number of G is a half of the number of odd vertices in G is NP-complete; and it remains NP-complete for planar graphs with maximum degree 4. Dereniowski et al. [4] characterized graphs for which 2 or 3 searchers are sufficient in the fast search model. They proved that the fast searching problem is NP-hard for multigraphs. Dyer et al. [5] considered complete bipartite graphs $K_{m,n}$, and computed the fast search number of $K_{m,n}$ when m is even. They also presented lower and upper bounds on the fast search number of $K_{m,n}$ when m is odd. Xue et al. [12] provided lower bounds and upper bounds on the fast search number of complete k-partite graphs. They also solved the open problem of determining the fast search number of complete bipartite graphs. In [11], Xue and Yang provided lower bounds on the fast search number, and gave formulas for the fast search number of the cartesian product of an Eulerian graph and a path, as well as variants of the cartesian product.

In this paper, we introduce the notion of k-combinable graphs, and develop a new method for computing their fast search number. The method can be seen as a general method for finding lower bounds on the fast search number. Using this new method, we examine the fast search number of several classes of graphs including cacti graphs and cartesian product of a tree and an edge.

2 K-Combinable Graphs

We first introduce a class of graphs named *k-combinable graphs*. Then we describe our 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 pendant edges of G. The *profile* of G is an ordered tuple $\pi_G = (\pi_1, \dots, \pi_z)$ of positive integers, which is defined as follows:

1. If $E'_G = \emptyset$, then $z = 1$ and $\pi_1 = \text{fs}(G)$.
2. If $E'_G \neq \emptyset$ and $|E'_G| = k$, then $z = k!2^k$ and each component π_i of π_G 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 with which a fast search strategy can clear G if it traverses the edges in E'_G in the order of σ and in the directions as given by the chosen orientations.

Let G_1 be a connected graph that has $k_1 \geq 1$ pendant edges, and let G_2 be a connected graph having $k_2 \geq 1$ pendant edges. We choose k to be a constant satisfying that $1 \leq k \leq \min\{k_1, k_2\}$. Let $\overrightarrow{e_1} = (u_1u'_1, \dots, u_ku'_k)$, where $u_iu'_i \in E(G_1)$ and u'_i is a leaf node. Let $\overrightarrow{e_2} = (v_1v'_1, \dots, v_kv'_k)$, where $v_iv'_i \in E(G_2)$ and v'_i is a leaf node. 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 $\overrightarrow{e_2}$:

1. remove edges $u_i u_i'$ and $v_i v_i'$, for $1 \leq i \leq k$;
2. remove vertices u_i' and v_i', for $1 \leq i \leq k$;
3. connect u_i and v_i by adding a new edge, for $1 \leq i \leq k$.

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 result. Hence, we define the *align 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 graph obtained by performing the above operations.

Definition 1. Let $m \geq 2$. Let $G_1, \ldots, G_m$ be connected graphs. The sequence $(G_1, \ldots, G_m)$ is k-combinable if there are edge sequences $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$, $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$ such that:

1. For $1 \leq i \leq m$, $\overrightarrow{e_i}$ is a sequence of pendant edges of G_i.
2. For $2 \leq i \leq m-1$, $\overrightarrow{e_{1,i}}$ is a sequence of pendant 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 $1 \leq i \leq m$, 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.
4. For $2 \leq j \leq m-1$, the set of all edges of H_j, 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.

Further, 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 $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$, $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$.

Obviously, there may exist more than one graph that is a k-combination of $(G_1, G_2, \ldots, G_m)$. Further, for each k-combination G of $(G_1, G_2, \ldots, G_m)$, there exist specific $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$ and $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$ for obtaining 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 $\overrightarrow{e_{1,i}}$ and $\overrightarrow{e_j}$, where $2 \leq i \leq m-1$ and $1 \leq j \leq m$.

Theorem 1. *There exists an algorithm that, given the profiles and edge pairing sequences of G_1 and G_2 such that G is the k-combination of (G_1, G_2) with respect to the edge pairing sequences, runs in $O((k_1 + k_2 - k)!2^{k_1+k_2-k})$ time to compute the profile of G. Here k_i refers to the number of pendant edges of G_i, where $1 \leq i \leq 2$.*

Proof. We briefly introduce the idea of how to compute the profile of G. Since G_1 and G_2 have k_1 and k_2 pendant edges respectively, the sizes of profiles of G_1 and G_2 are $k_1!2^{k_1}$ and $k_2!2^{k_2}$. Let $\overrightarrow{e_1}$ and $\overrightarrow{e_2}$ denote the edge pairing sequences of G_1 and G_2 respectively. 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 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 $V(G_1)$ and $V(G_2)$. For each parameter in the profile of G, it takes $O(k!2^k)$ time to compute its value. Further, we know the size of 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 $O((k_1 + k_2 - k)!2^{k_1+k_2-k})$. □

From Theorem 1, it is easy to see 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. 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 1, we can find an optimal fast search strategy for G as stated in Theorem 2. This novel method reveals an interesting property of fast searching that has not been exploited systematically in the literature to date.

Theorem 2. *Let G be a k-combination of $(G_1, \ldots, G_m)$ with respect to $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$, $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$, where $G_1, \ldots, G_m$ are connected graphs and k is a constant. There exists an algorithm which, given (1) the profiles of G_1, G_2, $\ldots$ G_m in sequence, and (2) $\overrightarrow{e_1}, \ldots, \overrightarrow{e_m}$ and $\overrightarrow{e_{1,2}}, \ldots, \overrightarrow{e_{1,m-1}}$, runs in polynomial time to compute the profile of G. Furthermore, the fast search number of G can be found in polynomial time.*

In the next section, we will apply Theorem 2 to the finding of optimal fast search strategy for cacti graphs; further, we also apply the theorem to the finding of optimal fast search strategies for cartesian product of a tree and an edge. We will show that (1) how to split a graph into smaller subgraphs, and (2) how to apply Theorem 2 to obtain an optimal fast search strategy, upon knowing the profiles of all the subgraphs in (1).

3 Cacti Graphs

A connected graph is a *cactus* if and only if each of its edges is contained in at most one cycle. In this section, we use G to denote a cactus graph. Let $v \in V(G)$ and let $\mathcal{G}_1, \ldots, \mathcal{G}_k$ be all the connected components from G by deleting v and all its incident edges. We use $\mathcal{G}_v^i$ to denote the subgraph of G induced by $V(\mathcal{G}_i) \cup \{v\}$, where $1 \leq i \leq k$. $\mathcal{G}_v^1, \ldots, \mathcal{G}_v^k$ are called *sub-cacti* of G with respect to vertex v. Note that $\mathcal{G}_v^1, \ldots, \mathcal{G}_v^k$ must satisfy:

(i) $V(\mathcal{G}_v^1) \cup \cdots \cup V(\mathcal{G}_v^k) = V(G)$,
(ii) $V(\mathcal{G}_v^i) \cap V(\mathcal{G}_v^j) = v$, where $1 \leq i \neq j \leq k$, and
(iii) $u_1, u_2 \in V(G)$ are adjacent, only if there exists i such that $u_1, u_2 \in V(\mathcal{G}_v^i)$.

Consider $\mathcal{G}_v^i$, where $1 \leq i \leq k$. Note that v has degree at most two in $\mathcal{G}_v^i$. If v is a leaf node in $\mathcal{G}_v^i$, then let u be a vertex in $V(\mathcal{G}_v^i)$ satisfying $u \sim v$. We use $\pi_I(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu is cleared by sliding a searcher from v to u. An *I-strategy* for $\mathcal{G}_v^i$ is a strategy in which (1) vu is cleared by sliding a searcher from v to u, and (2) $\pi_I(\mathcal{G}_v^i)$ searchers are placed on $V(\mathcal{G}_v^i) \setminus \{v\}$. Note that if vu is cleared by sliding a searcher from v to u in a strategy, then a searcher must be placed on v at the beginning of the strategy. We use $\pi_O(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu is cleared by sliding a searcher from u to v. An *O-strategy* for $\mathcal{G}_v^i$ is a strategy for $\mathcal{G}_v^i$ in which (1) vu is cleared by sliding a searcher from u to v, and (2) $\pi_O(\mathcal{G}_v^i)$ searchers are placed on $V(\mathcal{G}_v^i) \setminus \{v\}$.

If v has degree two in $\mathcal{G}_v^i$, then let u_1 and u_2 be the two vertices in $V(\mathcal{G}_v^i)$ satisfying that $u_1 \sim v$ and $u_2 \sim v$. 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 $\pi_{I,I}(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu_1 and vu_2 are both cleared by slide-in actions. We use $\pi_{O,O}(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu_1 and vu_2 are both cleared by slide-out actions. We use $\pi_{I,O}(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu_1 or vu_2 is cleared by a slide-in action, and later the other edge is cleared by a slide-out action. We use $\pi_{O,I}(\mathcal{G}_v^i)$ to denote the minimum number of searchers placed on $V(\mathcal{G}_v^i) \setminus \{v\}$ in a strategy for $\mathcal{G}_v^i$, in which vu_1 or vu_2 is cleared by a slide-out action, and later the other edge is cleared by a slide-in action. A strategy for $\mathcal{G}_v^i$ is an *II-strategy*, in which (1) $\pi_{I,I}(\mathcal{G}_v^i)$ searchers are placed on $V(\mathcal{G}_v^i) \setminus \{v\}$, and (2) vu_1 and vu_2 are both cleared by slide-in actions. In a similar way, we define *IO-strategy*, *OI-strategy* and *OO-strategy* for $\mathcal{G}_v^i$ respectively.

Definition 2. Consider a sub-cactus of G with respect to vertex v, i.e., $\mathcal{G}_v^i$.

1. If v has exactly one incident edge in $\mathcal{G}_v^i$, then the profile of $\mathcal{G}_v^i$ is defined as the pair $(\pi_I(\mathcal{G}_v^i), \pi_O(\mathcal{G}_v^i))$.
2. If v has exactly two incident edges in $\mathcal{G}_v^i$, then the profile of $\mathcal{G}_v^i$ is defined as the 4-tuple $(\pi_{I,I}(\mathcal{G}_v^i), \pi_{I,O}(\mathcal{G}_v^i), \pi_{O,I}(\mathcal{G}_v^i), \pi_{O,O}(\mathcal{G}_v^i))$.

For cactus graph G and $v \in V(G)$, we use G'_v to denote the graph obtained by adding either one or two pendant edges to v. There are two possibilities for G'_v:

(1) v has one added pendant edge in G'_v, say vu. Let $\pi_I(G'_v)$ be the minimum number of searchers placed on $V(G'_v) \setminus \{u\}$ in a strategy for G'_v, in which vu is cleared by sliding a searcher from u to v. An *I-strategy* for G'_v is a strategy, in which (a) $\pi_I(G'_v)$ searchers are placed on $V(G'_v) \setminus \{u\}$, and (b) vu is cleared by sliding a searcher from u to v. In a similar way, we define $\pi_O(G'_v)$ and *O-strategy* for G'_v. The *profile of* G'_v is defined as the pair $(\pi_I(G'_v), \pi_O(G'_v))$.

(2) v has two added pendant edges in G'_v. Notice that there are four distinct ways to clear the two added pendant edges of v. In a similar way, we define (1) $\pi_{I,I}(G'_v)$, $\pi_{I,O}(G'_v)$, $\pi_{O,I}(G'_v)$ and $\pi_{O,O}(G'_v)$ for G'_v, and (2) *II-strategy*, *IO-strategy*, *OI-strategy* and *OO-strategy* for G'_v. The *profile of* G'_v is defined as 4-tuple $(\pi_{I,I}(G'_v), \pi_{I,O}(G'_v), \pi_{O,I}(G'_v), \pi_{O,O}(G'_v))$.

Definition 3. For a strategy $\mathcal{S}$ for G, let the *reversed strategy* for $\mathcal{S}$ be obtained from $\mathcal{S}$ by making the following modifications:

1. Remove all placing actions from $\mathcal{S}$.
2. For each vertex $v \in V(G)$ that contains searchers at the end of $\mathcal{S}$, insert a placing action at the beginning that places the same number of searchers on v.

3. For each edge $e \in E(G)$, reverse the sliding action on e by letting searcher move in the opposite way to clear it.
4. Reverse the order of all sliding actions.

Clearly, the reversed strategy for $\mathcal{S}$ uses the same number of searchers to clear G. Hence, we have $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{O,O}(\mathcal{G}_v^i) - 2$, and $\pi_I(\mathcal{G}_v^i) = \pi_O(\mathcal{G}_v^i) - 1$.

Lemma 1. *$\mathcal{G}_v^i$ must have one of the following properties:*

1. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) = \pi_{O,I}(\mathcal{G}_v^i) = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
2. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) = \pi_{O,I}(\mathcal{G}_v^i) - 1 = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
3. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) = \pi_{O,I}(\mathcal{G}_v^i) - 2 = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
4. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) - 1 = \pi_{O,I}(\mathcal{G}_v^i) - 1 = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
5. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) - 1 = \pi_{O,I}(\mathcal{G}_v^i) - 2 = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
6. $\pi_{I,I}(\mathcal{G}_v^i) = \pi_{I,O}(\mathcal{G}_v^i) - 2 = \pi_{O,I}(\mathcal{G}_v^i) - 2 = \pi_{O,O}(\mathcal{G}_v^i) - 2$;
7. $\pi_I(\mathcal{G}_v^i) = \pi_O(\mathcal{G}_v^i) - 1$.

For convenience, we say $\mathcal{G}_v^i$ *satisfies* $(^i)$ if it has the i-th property in Lemma 1, where $1 \le i \le 7$. Consider $\mathcal{G}_v^1, \ldots, \mathcal{G}_v^k$. Let χ_v^i be the number of sub-cacti that satisfy $(^i)$, where $1 \le i \le 7$. Obviously, we have $0 \le \chi_v^i \le k$. Two strategies for G are said to be *equivalent* if they use the same number of searchers to clear G.

For any cactus graph G, algorithm FASTSEARCHCACTUS (See Algorithm 1) computes the minimum number of searchers required for clearing G.

Algorithm 1: FASTSEARCHCACTUS(G)

```
1 Input: A cactus graph $G$.
2 Output: The fast search number of $G$.
   1: Arbitrarily select a cut vertex $v$ in $V(G)$, whose removal results in $k \ge 2$
      connected components $H_1, \ldots, H_k$. Let $G_i$ denote the subgraph of $G$ induced
      by $V(H_i) \cup \{v\}$, where $1 \le i \le k$. Let $E_{\text{cut}}$ denote the edge set consisting of
      all edges connecting $v$ and vertices in $V(H_1)$. Let $G'$ be the subgraph of $G$
      induced by $V(H_2) \cup \cdots \cup V(H_k) \cup V(E_{\text{cut}})$.
   2: Let $\mathcal{P}_{G_i}$ be the output of CLEARCACTI1($G_i$, $v$), where $1 \le i \le k$.
   3: Let $\mathcal{P}_{G'}$ be the output of CLEARCACTI3($G'$, $E_{\text{cut}}$, $v$, $\{\mathcal{P}_{G_2}, \ldots, \mathcal{P}_{G_k}\}$).
   4: List all the possible combinations of the profiles from $\mathcal{P}_{G_1}$ and $\mathcal{P}_{G'}$
      respectively with respect to sliding actions on all the edges in $E_{\text{cut}}$.
   5: return the minimum number of searchers in all the combinations.
```

In algorithm FASTSEARCHCACTUS, we define G_i, where $1 \le i \le k$. Algorithm CLEARCACTI1 (See Algorithm 2) computes the profiles of G_i. The input of the algorithm includes G_i, along with the cut vertex $v \in V(G)$. The output is the profile of G_i.

Algorithm 2: ClearCacti1(G_i, v)

```
1: If $G_i$ is a tree, then let $\pi_I(G_i)$ be the number of searchers that are placed on $V(G_i) \setminus \{v\}$ in the
   I-strategy produced by FS($G_i$) in [5]. Let $\pi_O(G_i) \leftarrow \pi_I(G_i) + 1$. Let $(\pi_I(G_i), \pi_O(G_i))$ be the
   profile of $G_i$.
2: If $G_i$ is a simple cycle, then let $\pi_{I,I}(G_i) \leftarrow 0$, $\pi_{I,O}(G_i) \leftarrow 0$, $\pi_{O,I}(G_i) \leftarrow 2$,
   and $\pi_{O,O}(G_i) \leftarrow 2$. Let $(\pi_{I,I}(G_i), \pi_{I,O}(G_i), \pi_{O,I}(G_i), \pi_{O,O}(G_i))$ be the
   profile of $\mathcal{G}_v^i$.
3: If $G_i$ is neither a tree nor a simple cycle, then there are two subcases:
    (i) if $v$ is contained in a cycle of $G_i$, then let the output of ClearCacti2($G_i$, $v$)
        be the profile of $G_i$;
   (ii) if $v$ is a leaf node of $G_i$, then let $u \in V(G_i)$ be the vertex such that $v \sim u$;
        let the output of ClearCacti1($G_i - \{uv\}$, $u$) be the profile of $G_i$.
4: **return** the profile of $G_i$.
```

Algorithm ClearCacti2 (See Algorithm 3) is used to compute the profile of a sub-cactus in which v is contained in a cycle. The input of the algorithm includes a sub-cactus G_i and the cut vertex v. The output of the algorithm is the profile of G_i.

Algorithm 3: ClearCacti2(G_i, v)

```
1: Let $\mathcal{C} = vu_1 \dots u_{k'}v$ be the shortest cycle in $G_i$ that contains $v$. Let
   $\mathcal{H}_{u_1}, \dots, \mathcal{H}_{u_{k'}}$ denote the $k'$ connected components that contain $u_1, \dots, u_{k'}$
   respectively, which are obtained by deleting all edges in $E(\mathcal{C})$ from $G_i$. Let
   $E_j \subset E(\mathcal{C})$ be the set containing the two incident edges of $u_j$, where $1 \le j \le$
   $k'$. Let $\mathcal{G}_{u_j}$ denote the connected subgraph obtained from $\mathcal{H}_{u_j}$ by adding two
   edges in $E_j$ to $u_j$.
2: For $j \leftarrow 1, \dots, k'$:
   (2.1) Let $H_1, \dots, H_m$ denote all the connected components of $\mathcal{H}_{u_j}$ after
         removing the vertex $u_j$. Let $H'_\ell$ be the subgraph of $\mathcal{H}_{u_j}$ induced by
         $V(\mathcal{H}_\ell) \cup \{u_j\}$, where $1 \le \ell \le m$.
   (2.2) Let $\mathcal{P}_{H'_\ell}$ be the output of ClearCacti1($H'_\ell$, $u_j$), where $1 \le \ell \le m$.
   (2.3) Let the output of ClearCacti3($\mathcal{G}_{u_j}$, $E_j$, $u_j$, $\{\mathcal{P}_{H'_1}, \dots, \mathcal{P}_{H'_m}\}$) be the
         profile of $\mathcal{G}_{u_j}$.
3: Let $W \leftarrow \mathcal{G}_{u_1}$. Let $j \leftarrow 2$.
4: Note that $W$ and $\mathcal{G}_{u_i}$ have one edge in common. A strategy for $W \cup \mathcal{G}_{u_j}$ can
   be obtained from strategies for $W$ and $\mathcal{G}_{u_j}$ by reaching an accord on the
   sliding action on the common edge of $W$ and $\mathcal{G}_{u_j}$. Note that in the graph
   $W \cup \mathcal{G}_{u_j}$, $u_1$ and $u_j$ have one pendent edge in $E(\mathcal{C})$ respectively. Compute the
   profile of $W \cup \mathcal{G}_{u_j}$ with respect to the sliding actions on the pendent edges of
   $u_1$ and $u_j$, which consists of
   $\pi_{I,I}(W \cup \mathcal{G}_{u_j}), \pi_{I,O}(W \cup \mathcal{G}_{u_j}), \pi_{O,I}(W \cup \mathcal{G}_{u_j}), \pi_{O,O}(W \cup \mathcal{G}_{u_j})$.
5: Let $W \leftarrow W \cup \mathcal{G}_{u_j}$. If $j = k'$, then go to Step 6; otherwise, let $j \leftarrow j + 1$
   and go to Step 4.
6: **return** the profile of $W$.
```

Algorithm ClearCacti3 (see Algorithm 4) is used for computing the profile of G'_v, which is obtained by adding either one or two pendant edges to the cut vertex v. The input of the algorithm includes G', E_{cut}, v and $\mathcal{P}$. The output of the algorithm is the profile of G'.

Algorithm 4: ClearCacti3(G', E_{cut}, v, $\mathcal{P}$)

```
1: If $|E_{\text{cut}}| = 1$, then let $\pi_I(G')$ be obtained from the output of
   ClearCacti4($G', 1, 1, \mathcal{P}$). Let $\pi_O(G') \leftarrow \pi_I(G') + 1$.
2: If $|E_{\text{cut}}| = 2$, then:
     (i) let $\pi_{I,I}(G')$ be the minimum number of searchers required for clearing
         $G'$, where edges in $E_{\text{cut}}$ are cleared by slide-in actions.
    (ii) let $\pi_{O,O}(G') \leftarrow \pi_{I,I}(G') + 2$.
   (iii) let $\pi_{I,O}(G')$ be the minimum number of searchers required for clearing
         $G'$, where one edge in $E_{\text{cut}}$ is cleared by a slide-in action, followed by the
         other edge in $E_{\text{cut}}$ being cleared by a slide-out action.
    (iv) let $\pi_{O,I}(G')$ be the minimum number of searchers required for clearing
         $G'$, where one edge in $E_{\text{cut}}$ is cleared by a slide-out action, followed by
         the other edge in $E_{\text{cut}}$ being cleared by a slide-in action.
3: return the profile of $G'$.
```

Algorithm ClearCacti4($G', \sigma_1, \sigma_2, \mathcal{P}$) (which is omitted due to space limit) is called by ClearCacti3 as a subroutine, which computes the total number of searchers for clearing G' under some specific setting. Let $\mathcal{P}$ be the set containing the profiles of all the sub-cacti of $G' - E_{\text{cut}}$ with respect to vertex v. We use σ_1 to record the number of available searchers on v which could be used in an II-strategy or an I-strategy for a sub-cactus. We use σ_2 to denote the maximum number of searchers residing on v at some moment in a strategy for G'. For simplicity, σ_2 is set to 2 if there exists some moment in a strategy for G' at which v contains two or more searchers.

Lemma 2. *Consider all the sub-cacti of G with respect to v. For any strategy for G, there exists an equivalent strategy such that all the sub-cacti are cleared in the following order:*

1. *all the sub-cacti that are cleared by an O-strategy or an OO-strategy;*
2. *all the sub-cacti that are cleared by an OI-strategy (for each sub-cactus, perform all actions of searchers in its strategy until one of v's incident edges is cleared);*
3. *all the sub-cacti that are cleared by an IO-strategy;*
4. *all the sub-cacti that are cleared by an OI-strategy (for each sub-cactus, perform all actions of searchers in its strategy after one of v's incident edges is cleared);*
5. *all the sub-cacti that are cleared by an I-strategy or an II-strategy.*

Lemma 3. *Consider all the sub-cacti of G with respect to v, denoted as $G_1, \ldots, G_k$. For any strategy for G, there exists an equivalent strategy in which:*

1. *if G_i satisfies (^{1}), then it is cleared by an OI-strategy or an OO-strategy;*
2. *if G_i satisfies (^{2}), then it is cleared by an IO-strategy, an OI-strategy or an OO-strategy;*

3. *if G_i satisfies $(^3)$, then it is cleared by an IO-strategy or an OO-strategy;*
4. *if G_i satisfies $(^4)$, then it is cleared by an II-strategy, an OI-strategy or an OO-strategy;*
5. *if G_i satisfies $(^5)$, then it is cleared by an II-strategy, an IO-strategy or an OO-strategy;*
6. *if G_i satisfies $(^6)$, then it is cleared by an II-strategy, or an OO-strategy.*

Definition 4. A strategy is called a standard strategy for G with respect to v, where $v \in V(G)$, if (1) all the sub-cacti with respect to v are cleared in the order given in Lemma 2, and (2) each sub-cactus G_v^i, where $1 \leq i \leq k$, is cleared by a strategy in accordance with Lemma 3.

In the remainder of this section, we assume that every strategy for G'_v is a standard strategy with respect to v without subscripts.

Theorem 3. *For any cactus graph G, the fast search number of G can be computed in linear time by algorithm* FASTSEARCHCACTUS.

Proof. The algorithm FASTSEARCHCACUTS runs in linear time, as we can verify the time complexity as follows:

1. the profile of the sub-cactus with respect to each vertex in $V(G)$ has constant size;
2. the profile of the sub-cactus with respect to each vertex in $V(G)$ is computed at most once;
3. the profile of the sub-cactus with respect to each vertex in $V(G)$ is passed as parameter at most once when computing the profile of other sub-cactus;
4. the computation of the profile of the sub-cactus with respect to a vertex in $V(G)$ takes constant time.

Obviously, the algorithm FASTSEARCHCACUTS computes the fast search number of G in linear time. □

Theorem 4. *For any cactus graph G, we can obtain an optimal fast search strategy in linear time using* FASTSEARCHCACTUS.

Proof. This can be achieved by first using a back-track method to record how every edge of G is cleared after calling FASTSEARCHCACTUS. In addition, we can record the vertices of G on which searchers are placed throughout FASTSEARCHCACTUS. Based on these records, we can easily obtain an optimal fast search strategy for G by letting those searchers move along edges following the prescribed directions. □

4 Cartesian Product of a Tree and an Edge

Given two graphs G and H, the *cartesian product* of G and H, denoted $G\Box H$, is the graph whose vertex set is the cartesian product $V(G) \times V(H)$, and in which two vertices (u, v) and (u', v') are adjacent if and only if $u = u'$ and v is adjacent to v' in H, or $v = v'$ and u is adjacent to u' in G.

In what follows, we apply Theorem 2 to find an optimal fast search strategy for $T\Box P_2$, where T has at least three vertices. Let S_n, where $n \geq 3$, denote a star graph of n vertices. Let H_n denote the graph obtained by connecting the center vertices of two copies of S_n. Without loss of generality, let S_n^1 and S_n^2 denote the two copies of S_n in H_n. For any pair of edges that are from $E(S_n^1)$ and $E(S_n^2)$ respectively, there are four distinct ways to clear the two edges in a fast search strategy for H_n:

1. both edges are cleared by sliding a searcher from leaf to center node;
2. one of the two edges is cleared by sliding a searcher from leaf to center node, followed by the other edge being cleared by sliding a searcher from center node to leaf;
3. one of the two edges is cleared by sliding a searcher from center node to leaf, followed by the other edge being cleared by sliding a searcher from leaf to center node;
4. both edges are cleared by sliding a searcher from center node to leaf.

For convenience, we use *II*, *IO*, *OI* and *OO* to represent the above four ways respectively in the remainder of this section. Note that there are two layers in $T\Box P_2$. Let T_1 and T_2 be the two layers in $T\Box P_2$. Let $v_c^1 \in V(T_1)$ be a vertex of degree $k \geq 3$. Let $v_c^2 \in V(T_2)$ be the vertex where $v_c^2 \sim v_c^1$. Let V_c' be the subset of $V(T\Box P_2)$, which consists of v_c^1, v_c^2 and all their adjacent vertices in $V(T\Box P_2)$. Let E_c' be the subset of $E(T\Box P_2)$, in which v_c^1 or v_c^2 is an end point of each edge. We use $\mathcal{G}_c'$ to denote the connected subgraph of $T\Box P_2$, whose vertex set is V_c' and edge set is E_c'. It is easy to see that $\mathcal{G}_c'$ is the same as H_k. Let $\mathcal{G}_1, \ldots, \mathcal{G}_{k-1}$ be the connected components after deleting all edges in E_c' and all isolated vertices from $T\Box P_2$. We use $\mathcal{G}_i'$, where $1 \leq i \leq k-1$, to denote the subgraph of $T\Box P_2$, which is obtained from $\mathcal{G}_i$ by adding two pendant edges in $E(T\Box P_2)$ that connect vertices from $\{v_c^1, v_c^2\}$ and $V(\mathcal{G}_i)$. Note that there are four ways to clear the two pendant edges of $\mathcal{G}_i'$. We use $s_1(\mathcal{G}_i')$ to denote the minimum numbers of searchers needed to be placed on $V(\mathcal{G}_i)$ in a strategy for $\mathcal{G}_i'$, in which the two pendant edges are cleared by II. In a similar way, we define $s_2(\mathcal{G}_i')$, $s_3(\mathcal{G}_i')$ and $s_4(\mathcal{G}_i')$.

Lemma 4. *$\mathcal{G}_i'$ must have one of the following properties:*

1. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') = s_3(\mathcal{G}_i') = s_4(\mathcal{G}_i') - 2$;
2. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') = s_3(\mathcal{G}_i') - 1 = s_4(\mathcal{G}_i') - 2$;
3. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') = s_3(\mathcal{G}_i') - 2 = s_4(\mathcal{G}_i') - 2$;
4. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') - 1 = s_3(\mathcal{G}_i') - 1 = s_4(\mathcal{G}_i') - 2$;
5. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') - 1 = s_3(\mathcal{G}_i') - 2 = s_4(\mathcal{G}_i') - 2$.
6. $s_1(\mathcal{G}_i') = s_2(\mathcal{G}_i') - 2 = s_3(\mathcal{G}_i') - 2 = s_4(\mathcal{G}_i') - 2$.

Let $\mathcal{G}'$ be the graph with vertex set $V(\mathcal{G}_1) \cup \cdots \cup V(\mathcal{G}_{k-2}) \cup V_c'$ and edge set $E(\mathcal{G}_1) \cup \cdots \cup E(\mathcal{G}_{k-2}) \cup E_c'$. Given the profiles of $\mathcal{G}_i'$, where $1 \leq i \leq k-2$, we can compute the minimum number of searchers required for clearing $\mathcal{G}'$.

Lemma 5. *For each connected component $\mathcal{G}_i'$, where $1 \leq i \leq k-2$, if we know $s_1(\mathcal{G}_i')$, $s_2(\mathcal{G}_i')$, $s_3(\mathcal{G}_i')$ and $s_4(\mathcal{G}_i')$, then we can compute $s_1(\mathcal{G}')$, $s_2(\mathcal{G}')$, $s_3(\mathcal{G}')$ and $s_4(\mathcal{G}')$.*

Note that $E(\mathcal{G}')$ and $E(\mathcal{G}'_j)$ have exactly two common edges. Given the profiles of $\mathcal{G}'$ and $\mathcal{G}'_j$, we can list all the possible combinations of the profiles with respect to the sliding actions on the two edges. The fast search number of $T\Box P_2$ is the minimum number of searchers in all the combinations. From Lemma 5, we have the following result:

Lemma 6. *For each connected component $\mathcal{G}'_i$, where $1 \le i \le k-1$, if we know $s_1(\mathcal{G}'_i)$, $s_2(\mathcal{G}'_i)$, $s_3(\mathcal{G}'_i)$ and $s_4(\mathcal{G}'_i)$, then we can compute the minimum number of searchers for clearing $T\Box P_2$.*

Theorem 5. *An optimal fast search strategy for $T\Box P_2$ can be found in polynomial time.*

Proof. We briefly describe a strategy below for finding an optimal fast search strategy for $T\Box P_2$.

1. Arbitrarily select a pair of vertices v and v', where v' is the corresponding vertex of v in $T\Box P_2$.
2. For each of the connected components of $T\Box P_2$ with respect to v and v', compute its profile.
3. Compute the optimal fast search number of $T\Box P_2$ based on the profiles of all the connected components.
4. Use a back-track method to record how every edge of $T\Box P_2$ is cleared, as well as all vertices that are placed searchers. Based on these records, produce an optimal fast search strategy for $T\Box P_2$ by letting those searchers move along edges following the prescribed directions.

Clearly, the above strategy can find an optimal fast search strategy for $T\Box P_2$ in polynomial time. □

References

1. Bienstock, D.: Graph searching, path-width, tree-width and related problems (a survey). DIMACS Ser. Discrete Math. Theoret. Comput. Sci. **5**, 33–49 (1991)
2. Bonato, A., Yang, B.: Graph searching and related problems. In: Pardalos, P.M., Du, D.-Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, pp. 1511–1558. Springer, New York (2013). https://doi.org/10.1007/978-1-4419-7997-1_76
3. Breisch, R.: An intuitive approach to speleotopology. Southwestern Cavers **6**(5), 72–78 (1967)
4. Dereniowski, D., Diner, Ö., Dyer, D.: Three-fast-searchable graphs. Discret. Appl. Math. **161**(13), 1950–1958 (2013)
5. Dyer, D., Yang, B., Yaşar, Ö.: On the fast searching problem. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 143–154. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68880-8_15
6. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theoret. Comput. Sci. **399**(3), 236–245 (2008)
7. Makedon, F.S., Papadimitriou, C.H., Sudborough, I.H.: Topological bandwidth. SIAM J. Algebraic Discrete Methods **6**(3), 418–444 (1985)

8. Megiddo, N., Hakimi, S.L., Garey, M.R., Johnson, D.S., Papadimitrioum, C.H.: The complexity of searching a graph. J. ACM **35**(1), 18–44 (1988)
9. Parsons, T.: Pursuit-evasion in a graph. In: Proceedings of the International Conference on the Theory and Applications of Graphs, pp. 426–441. Springer-Verlag (1976). https://doi.org/10.1007/BFb0070400
10. Stanley, D., Yang, B.: Fast searching games on graphs. J. Comb. Optim. **22**(4), 763–777 (2011)
11. Xue, Y., Yang, B.: The fast search number of a cartesian product of graphs. Discret. Appl. Math. **224**, 106–119 (2017)
12. Xue, Y., Yang, B., Zhong, F., Zilles, S.: The fast search number of a complete k-partite graph. Algorithmica **80**(12), 3959–3981 (2018)
13. Yang, B.: Fast edge searching and fast searching on graphs. Theoret. Comput. Sci. **412**(12), 1208–1219 (2011)