On Phalanx Graph Search Number

Ondřej Navrátil[1], Sanpawat Kantabutra[2], Sheng-Lung Peng[1]

[1] Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan
[2] Computer Engineering Department, Chiang Mai University, Thailand
navratil@gms.ndhu.edu.tw, sanpawat@alumni.tufts.edu, slpeng@ndhu.edu.tw

Abstract

We introduce an extension of the Connected Graph Search, called Phalanx Graph Search, which inherently emerges from the nature of certain applications. We discuss its key properties, prove NP-hardness of the problem on general graphs and introduce a linear-time algorithm for the class of trees. We exploit our analysis to examine the Minimum Phalanx Graph Search Spanning Tree Problem, again showing its hardness and investigating efficiency of certain approximations. We discuss some of our findings in relation to other search variants.

Keywords: Phalanx graph searching, Connected graph searching, Spanning tree

1 Introduction

In the field of graph theory, graph search is a famous problem with a variety of interpretations in contexts such as search and rescue operations, fugitive capture (or capture evasion on the contrary), or area decontamination. Despite the wide range of applications, the problem definition itself is precise and compact. This provides a fine ground for theoretical analysis of its properties.

Let $G=(V,E)$ be a connected simple graph and $\{u,v\}\in E$ be an edge. We will use $T=(V,E)$ if the graph G is a tree. Furthermore, let C_n, K_n, P_n denote a cycle, a complete graph, and a path graph on n vertices, respectively.

A *search strategy S* on G is a sequence of following operations:

(1) Put a searcher on a vertex, or
(2) Remove a searcher from a vertex, or
(3) Slide a searcher on vertex u along an edge $\{u,v\}$.

Any search strategy is subject to the following restrictions:

- At the beginning, all $e\in E$ are *contaminated*, and
- After the last step of S, all $e\in E$ are *clear*, and
- A clear edge $\{u,v\}$ gets *recontaminated* if, at any step of S, there is a contaminated edge $\{u',v'\}$ and a path $u'\to v'$ without a searcher on any of the path vertices.

Moreover, there are three major variants with regards to the process of clearing an edge $\{u,v\}\in E$ by S:

(1) *Node-search* – both u,v are occupied by searchers,
(2) *Edge-search* – searcher slides along edge $\{u,v\}$,
(3) *Mixed-search* – any of the above.

The metric for comparing strategies is *search number* $s(S)$, defined as the maximum count of searchers in the graph at any step of S. Similarly, for any G, we define a *graph search number* $s(G)$ as:

$$s(S)=\min_{s\ is\ a\ strategy\ in\ G} s(G)$$

Any strategy with $s(S)=s(G)$ is called *optimal*.

The decision version of the GRAPH SEARCHING PROBLEM is defined as follows: Given a graph G and $k\in N$, is $s(G)\le k$?

Further restrictions may be considered in the search strategy to derive related problems. The most popular restrictions include:

- Internal – a searcher, once put, is never removed,
- Monotone – recontamination never occurs,
- Connected – clear edges always induce a connected subgraph.

The respective search numbers are denoted by $is(G)$, $ms(G)$, $cs(G)$. Restrictions can be arbitrarily combined, creating e.g. $mcs(G)$, $mis(G)$ etc. Finally, in applications adopting the fugitive capture interpretation, there are invisible and visible search variants, depending on whether the location of the fugitive is known to the searchers.

The rest of the paper is structured as follows. Section 2 summarizes state-of-the-art results. In Section 3, we introduce our novel search variant, Phalanx Graph Search, show its hardness and investigate key properties. Section 4 aims to describe a lower bound of a search number and finds an optimal algorithm for the case of trees. The last section

*Corresponding Author: Ondřej Navrátil; E-mail: navratil@gms.ndhu.edu.tw

DOI: 10.3966/160792642020072104027

introduces a combination of Phalanx Graph Search and Minimum Spanning TREE problems. We proceed by proving its hardness and discuss various approximation approaches. Further, we conclude the impact of our results on existing search variants.

2 Related Work

GRAPH SEARCH was first mentioned in 1967 in a speleologist journal [1] as an emergency scenario for a case of a lost explorer. It was not until 1978 that the problem was formalized by Parsons [2]. Subsequent applications and interpretations followed. In [3-4], authors addressed the problem as a strategy to clear a set of tunnels contaminated by poisonous gas. Further, the game is often denoted as pursuit evasion, where the aim is to assist police in a search for a fugitive, or vice versa [5]. Since graphs are elementary models for networks, intruder-capture problems following the definition of GRAPH SEARCH are studied in [6]. More recently, applications emerged in the field of robotics [7].

One of the most extensively studied properties of the problem is in the field of complexity theory. Determining whether $s(G) \leq k$ for arbitrary G and k is *NP*-complete according to [8], completed in [9]. Lapaugh [10] proved that for every graph G there is always a monotone search strategy that uses $s(G)$ searchers. The problem of determining minimal search strategies under the connectedness and/or the internality constraint is still *NP*-hard in general (it follows from the reduction in [8], as observed in [11]).

In general, connected strategies deviate further from the original problem. In [12] authors observe that there is a graph for which $mcs(G) > cs(G)$, a major distinction from the non-connected search. This complicates the problem categorization because the limit of the strategy size is not bounded by edge count. Moreover, it is not known whether the decision problem corresponding to connected search is in *NP* or not. The good news though is that the authors in [11] prove that monotony holds for trees, i.e. $cs(T) = mcs(T)$.

Despite the problem hardness, better results can be achieved on certain subclasses. In [8] the authors showed that a tree can be decomposed into a set of paths in different levels. These paths are called avenues. By using avenues, an optimal search strategy can be constructed. The authors in [13-14] extended this idea to other search variants (node and mixed search). In addition, there are trees for which minimal internal search strategies require $\Omega(n \log n)$ moves (i.e. edge traversal) [8].

One may assume that, like monotony or internality, connectedness is achieved without increasing the number of searchers for trees. This assumption is disproved by the case of a 22-node tree T obtained from three copies of the complete binary tree of height 2 connected to a common root. For connected search on trees, authors in [11] introduced a linear time algorithm for both search number and search strategy computation. Notation of caterpillars and spines was introduced in [15], offering better characteristics. Caterpillars represent a connected analogy of an avenue.

Recently, a new variant called Cooperative Graph Search was introduced in [16]. Further, a novel Node-Search Spanning Tree problem was defined in [17], combining the Graph Search and Minimum Spanning Tree problems. In all cases, authors proved *NP*-hardness/completeness of the problems.

3 Phalanx-Search Number

After defining the problem and giving some background, we proceed to compare its formalized properties against some of its applications. On this basis, we attempt to address the discrepancies.

Even though we construct search strategies with complete information about the environment (namely the structure of the underlying graph), performing the strategy relies solely on individual, potentially isolated searchers following their roles. For the search strategy to succeed, it is essential to synchronize the search steps. Manipulating the order of steps performed may render the strategy useless, even if the affected steps concern different searchers. Moreover, this does not contradict searcher's perfect knowledge of the overall strategy. The strategy itself may also be aborted prematurely if the search goal is achieved (e.g. fugitive was captured). Since search steps themselves may be costly and/or hazardous, spreading this information is vital.

From this point of view, we must require some synchronization mechanism between the searchers. Because a search strategy is discrete from the definition, we may therefore assume the existence of some common “clock” and time every action to a certain tick, thus avoiding any potential conflict. However, physical execution of any step (e.g. edge slide) is rarely an atomic operation, and may require a variable, or even beforehand unknown amount of time.

Consider the original problem application with a lost speleologist. In this case, searchers explore narrow corridors surrounded by massively thick rock. Clearly, peer-to-peer wireless communication is out of consideration. They may choose to deploy a set of beacons in the tunnel system to overcome this difficulty, yet this grid of beacons may need to move along with the search strategy.

Furthermore, police/military operations may require radio silence, or another means of undercover limitation. To enable synchronization, a network of short range agents may be used to maintain the connection between searchers.

In a similar way, let us look into an example of fire fighters extinguishing a fire. Not only do the individual searchers need to communicate in an area covered in thick smoke, they also require a water hose supply to support their progress. The hose length limits their reach from a hydrant or fire engine, and in many cases, guards need to secure hose junctions.

All of the scenarios described above suggest some kind of connectedness in search strategies.

Eventually, let us raise questions that follow directly from the current problem formalization. Is the cost of searching an arbitrary large cycle still constant, i.e. $s(C_n) = 2$ (resp. 3) for an arbitrary *n*? Does contracting edges with a degree-2 extremity (or dually - replacing an edge by a degree-2 vertex) introduce no difference to the search number?

To address the abovementioned inaccuracies, we introduce a new variant of graph search.

Definition 1: *We call a search strategy S a Phalanx search strategy if and only if at each point of the search the vertices occupied by searchers induce a connected subgraph.*

Also, denote the minimum number of searchers needed to clear a graph G by a Phalanx strategy as Phalanx-search number of G, or ps(G).

After formalizing our requirements, we utilize its specifics and investigate its key properties.

3.1 Contaminated Edges

Following the reasoning we administered in the previous section, we may require the induced subgraph to be connected even after removing contaminated edges. While this is trivially true for mixed- and edge-search, since any edge in the induced graph is clear from searcher occupation of both of its end vertices, we focus our attention on edge-search. One might suspect that there is always a Phalanx edge-search strategy that satisfies such requirement. This suspicion is easily disproved on C_3. The optimal strategy uses 2 searchers, but requires them to be connected by a contaminated edge at some point of the search, as illustrated in Figure 1. This concept trivially extends to $C_n, n \geq 3$.

Before we investigate further into this matter, we formalize the property. Let $V^S \subseteq V$ be the set of occupied vertices and E^c the set of clean edges at a step of a search strategy *S* in *G*. Denote $E^{c\&s} = \{\{u,v\} \in E^c \mid u,v \in V^s\}$.

Definition 2: *We call a search strategy S a tidy phalanx search strategy if and only if at every step of S, V^S is connected using only clean edges, i.e. the graph $G^{tidy} = (V^S, E^{c\&s})$ is connected.*

Note that we allow contaminated edges between vertices in V^S. However, we also require the induced subgraph to be connected even without them.

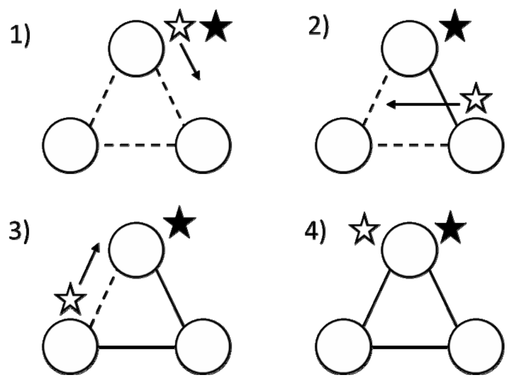

Figure 1. Phalanx strategy on C_3 inevitably uses a contaminated edge

With the formal basis in hand, we consider the upper-bound on the cost of tidiness in Phalanx graph search. Fortunately, the cost is rather minor as shown in the following proposition.

Proposition 1: For any G with $ps(G) = k$, there exists a tidy phalanx strategy S with no more that $k+2$ searchers.

Proof: Assume S^* to be the optimal phalanx strategy for *G*. First, we obtain *S* as a copy of S^*. If *S* is not tidy, then focus on the first step that breaks tidiness:

(a) A searcher is placed on *v*.

Slide an extra searcher from an occupied *u* to *v* and remove it after placing the original searcher on *v*.

(b) A searcher is removed from *v*.

Use an extra searcher to clear all edges between remaining V^S before removing *v*.

(c) A searcher slides along $\{u,v\}$.

Place an extra searcher on *u*, slide original searcher along the edge, then another extra searcher clears all edges between V^S before removing both of the extra searchers.

Observe that in all cases we used upmost two extra searchers. Repeat this procedure until *S* is tidy. **Q.E.D.**

Recall that it was a cycle C_3 that helped us realize that the contaminated edges eventually may be of some use. We attempt to particularize this observation. The utilization of contaminated edges between searchers in Phalanx search turns out to be conditioned by the existence of a cycle in *G*. We show this by a following result on trees.

Theorem 1: *For any tree T with $ps(T) = k$, there is a tidy phalanx strategy $S', s(S') = k$.*

Proof: Assume the optimal general phalanx strategy $S, ps(S) = k$. There are three options how a contaminated edge can occur in the subgraph induced by V^S:

(a) A searcher is placed on *v*.

Since *S* is a phalanx strategy, there must be an adjacent vertex *u* connected by contaminated $e = \{u,v\}$. We can replace this step in S' by putting a searcher on *u* instead and sliding it along *e*, thus keeping it connected by a clear edge. This concept is illustrated in Figure 2.

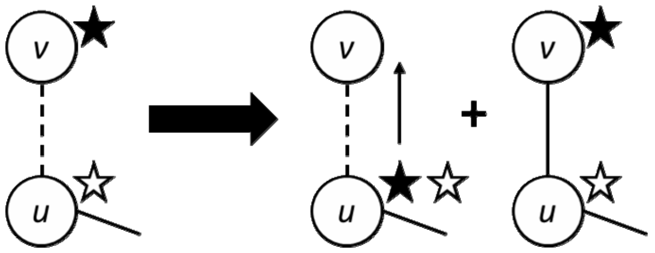

Figure 2. Eliminating non-tidy placement of a searcher

(b) A searcher slides on $e=\{u,v\}$.

Assume a (solitary) searcher on *u* that is initially adjacent to searcher on *w* via clear $\{u,w\}$, that slides along $\{u,w\}$ to eventually become adjacent to another searcher on *x* via $\{v,x\}$. Since *S* is a phalanx strategy, *x* must be also connected with *w* via a clear path $p_1 = x \to w$. There is also another path $p_2 = (w,u,v,x)$ that yields from the strategy step nature. We observe that p_1, p_2 enclose a cycle which contradicts our assumption that *T* is a tree. This situation is depicted in Figure 3.

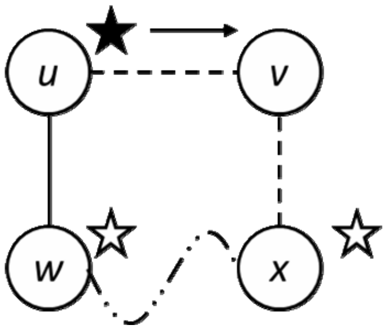

Figure 3. Proof of a cycle in a non-tidy slide

(c) A searcher is removed from *v*.

If a searcher on *u* was originally connected with a clear edge $\{u,v\}$, yet after removing searcher from *v* is only connected via a contaminated $\{u,x\}$, then, again, we have a cycle, since there also must be a clear path $p = u \to x$.

Replace all steps that match the pattern of a) by a tidy analogy. **Q.E.D.**

We cease our exploration of edge search and tidiness cost here. As marginal as the property itself may seem, we deem it useful when comparing Phalanx search to the connected search variant.

3.2 Complexity

According to existing literature, the vast majority of search problems on general graphs is proven to be *NP*-complete or *NP*-hard. Here we will restrict our research on node-search, although we conjecture that similar proofs may be constructed for the remaining search variants. Our main result is described in the following theorem.

Theorem 2: *Let $G=(V,E)$ be an arbitrary connected graph. The decision problem $ps(G)\le k$ is NP-hard for the node-search variant.*

Before providing the proof itself, we examinate an augmented version of *G*. Let $G'=(V',E')$ be an extension of *G* with an extra universal vertex *u*, namely:

$$V' = V \cup \{u\}$$

$$E' = E \cup \{\{u,v \mid v \in V\}$$

An example of augmenting *G* into G' is illustrated in Figure 4. The extra universal vertex *u* will assist us in bridging any strategy into a Phalanx strategy without altering the original problem nature.

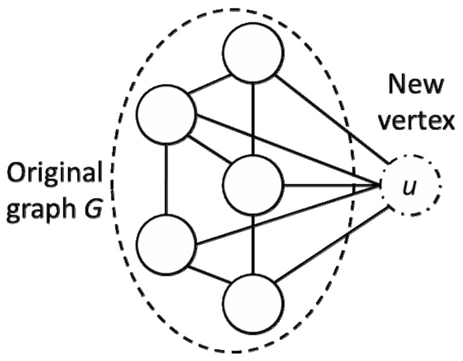

Figure 4. Augmentation of *G* into G'

Lemma 1: $ps(G') < k+1 \Leftrightarrow s(G) < k$

Proof: We will show that both implications hold.

(a) $ps(G') < k+1 \Rightarrow s(G) < k$

Since $ps(G') < k+1$, there is some Phalanx strategy S^* such that $ps(S^*) < k+1$. First, realize that any hypothetical steps performed in S^* before placing a searcher on *u* have no effect, since any unguarded edges will be immediately recontaminated from edges incident to *u*. In a similar way, removing a searcher from *u* before clearing the entire graph recontaminates all of edges without searchers on both end vertices. Therefore, we can assume that placing a searcher on *u* is the first step in S^*, and that this searcher is never removed until the end of the strategy.

Given that $ps(S^*) < k+1$, we can create a search strategy *S* in *G* from S^* by omitting the extra searcher on *u* which immediately yields $s(G) < k$.

(b) $ps(G') < k+1 \Leftarrow s(G) < k$

Since $s(G) < k$, there is some search strategy S^* in *G* such that $s(S^*) < k$. Create a search strategy S' by putting a searcher on *u* in the first step and then following the steps of S^*. This strategy is a Phalanx-strategy since every pair of searchers is always connected through *u*, and uses exactly $s(S^*)+1<k+1$ searchers. **Q.E.D.**

Proof (of Theorem 2): We can prove the problem hardness by reduction from classical Graph Search Problem to Phalanx Graph Search Problem, which directly follows from Lemma 1. **Q.E.D.**

Knowing that the decision version of our problem is *NP*-hard, we may wish to provide an upper bound on its hardness as well. In order to show membership in *NP*, we can either try to find a reduction to another problem from *NP*, or design an *NP* algorithm following the popular guessing and checking scheme. For the latter one, an obvious choice for certificate would be a sequence of edges representing the order in which they are cleared. However, we cannot assume the size of the certificate without studying the effects of recontamination on Phalanx Search. We remind the reader that the cost of monotony posed as an open problem for both edge search and connected search variants for years, surprisingly leading to diverse results for both. Therefore, we leave this problem open for future research.

4 The Case of Trees

In this section, we consider a subclass of graphs, namely trees, and investigate respective complexity of phalanx search. Once again, we will focus on node-search variant and show that a better result can be achieved on trees.

4.1 Lower Bound for $ps(T)$

We start by introducing an infinite class of trees which we deem to be a deciding component for a lower bound of Phalanx search number on trees.

Definition 3: *We denote an Urchin graph U_k as a tripod graph with $3k-2$ vertices where distances between pairs of leaves are equal.*

An example of U_k for certain values of *k* is shown in Figure 5. A crucial property of urchin U_k is described in the following lemma.

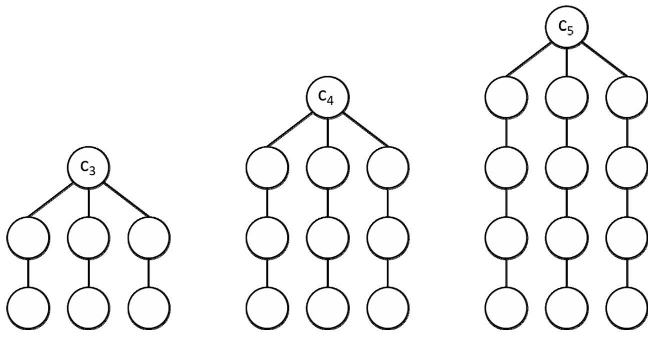

Figure 5. Urchin graphs (from left): U_3, U_4, U_5

Lemma 2: *For $k\geq 3, ps(U_k)=k$.*

Proof: First, we use a strategy that traverses from one leaf towards the degree-3 central vertex, then stretches a chain of *k* searchers towards another leaf, keeping a guard on the center, and eventually traverses from the center towards the last leaf. Therefore, $ps(U_k)\leq k$.

Furthermore, we observe that any phalanx strategy using fewer than *k* searchers cannot keep clear more than one of the edges incident to urchin leaves. Assume that at some point of the search, exactly one of the leaf-incident edges is clear. In order to clear other leaf-incident edge, we need searchers on both of its end vertices. However, since the distance from the leaf to c_k is *k*, the central vertex has to be left unguarded and causes recontamination of the previously cleared leaf-incident edge from the remaining leaf. Hence, $ps(U_k)\geq k$. The Lemma holds. **Q.E.D.**

We exploit the observation described in Lemma 2 to generalize the lower bound that determines $ps(T)$.

Theorem 3: *For a tree T and $k\geq 3$, $ps(T)=k$ if and only if:*

$$k=\max_{U_i \text{ is a subgraph of } G}\{i\} \qquad \textbf{(1)}$$

That is, U_k is the largest urchin subgraph of T.

Proof: $ps(T)\geq k$ directly follows from the proof of Lemma 2. Further, we will construct a phalanx strategy with *k* searchers to clear *T*. Assume that the urchin U_k has degree-3 central vertex *c*. Vertex *c* separates *T* into a number of branches rooted at *c*. Two branches T_1^c, T_2^c with height $h(T_{1,2}^c)\geq k$, one branch T_3^c with $h(T_3^c)=k$ and an arbitrary amount of other branches T_i^c with $h(T_i^c)\leq k$. Any other composition either generates U_{k+1}, or averts existence of U_k. See Figure 6 for the decomposition of *T* into these branches.

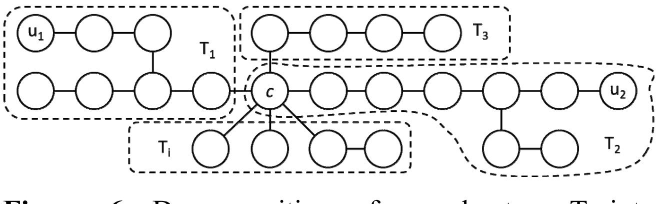

Figure 6. Decomposition of sample tree T into branches

We start creating our strategy around the path between the deepmost leaves $u_{1,2}$ of $T_{1,2}^c$. This path marks the search avenue. Observe that the number of searchers needed to clear a rooted tree, i.e. a branch rooted at some of the avenue vertices, is delimited by its height. Every non-path branch $T^{v,x}$ rooted at some path vertex *v* is guaranteed to be no higher than *k*, since for the existence of a branch $T^{v,x}$ with height $\geq k$ we have these contradictions:

1. If *v* is on path $u_1\rightarrow c$ and $d(u_1,v)\leq k$, then u_1 is

not the deepmost vertex in T_1^c.

2. If *v* is on path $u_1 \to c$ and $d(u_1, v) > k$, then *v* is a central vertex of U_{k+1} with branches $T^{v,x}, T^{v,u_1}, T^{v,c}$, each of which has height > *k*.

3. Analogic situation appears when *v* is on $c \to u_2$.

From here, it is easy to see that our strategy will proceed along the path $u_1 \to u_2$, clearing the non-path branches at each avenue node. **Q.E.D.**

We mark several important consequences of our findings.

Corollary 1. *Urchin U_k defines the lower bound for Phalanx node-search number on trees.*

Corollary 2. *For any tree T on $n \geq 4$ vertices, $ps(T) \leq (\frac{n+2}{3})$, or more generally $ps(T) \in O(n)$ and this boundary is tight.*

Corollary 3. *For any tree $T, ps(T) = mps(T)$.*

4.2 Algorithm for $ps(T)$

Knowing the patterns to look for, we proceed by designing an algorithm to determine $ps(T)$ in tractable time. Our goal is to find the central vertex of the largest urchin U_k, as suggested by Equation 1. We will denote k^c the degree of maximum urchin U_{k^c} that has *c* as a central vertex and is a subgraph of *T*. First, we observe that any vertex *c*, value of k^c is determined by the following equation:

$$k^c = \max\{i \in N \mid \exists T_{1,2,3}^C, h(T_{1,2,3}^C) \geq i\} \quad \textbf{(2)}$$

That is, the degree of the maximum urchin centered at *c* is the height of the 3rd highest branch of T rooted at *c*.

Hence, we will adjust the existing tree-sweep algorithm to determine k^c for every vertex of *T*. Later, we determine the search number:

$$ps(T) = \max\{k^c \mid c \in V\} \quad \textbf{(3)}$$

The concept is described in Algorithm 1. The core task of the algorithm is to calculate $h[u][v]$, i.e. the height of every $T^{u,v}$ branch. Initially, we calculate the degree of each node and enqueue leaves into *Q*. We will utilize the value of $d[v]$ as the number of incoming edges for which the height of the respective subtree rooted at *v* has not been calculated yet. Every time new $h[u][v]$ is calculated, $d[v]$ is decreased. Once $d[v]$ reaches 1, we are able to calculate $h[u][v]$ for the only vertex *u* where $h[u][v]$ is still unknown. As $d[v]$ reaches zero, all of the outgoing $h[v][u]$ can be completed.

Algorithm 1. Calculate $\boldsymbol{ps(T)}$

```
Input: Tree $T = (V, E)$
Data: Queue $Q$, arrays $h[V][V], d[V]$
Result: $ps(T)$
/* Initialize $d[v], h[v]$ */
for $v \in V$ do $d \in [v] \leftarrow 0$;
for $e \in \{u, v\} \in E$ do {
  $d[u]++$;
  $d[v]++$;
  $h[u,v] \leftarrow h[v,u] \leftarrow -1$;
}
/* Select leaves */
for $v \in V$ such that $d[v] = 1$ do {
  enqueue the only edge $(v, u)$ to $Q$;
}
/* Calculate $h[v][v]$ */
while $(u, v)$ = dequeue $Q$ do {
  $E_{in} \leftarrow \{h[w,u](w,u) \in E \wedge w \neq v\}$;
  $h[u,v] \leftarrow 1 + \max(\{0\} \cup E_{in})$;
  $d[v]--$;
  if { $d[v] = 1$ } then
    enqueue the only $(v, w)$ with $h[w,v] = -1$ to $Q$;
  else if $d[v] = 0$ then
    enqueue all of the $(v, w)$ with $h[v,w] = -1$ to $Q$;
}
/* Heights of the branch subtrees for each node
 are in $h[u][v]$ */
$c \leftarrow argmax_{v \in V}\{k^v\}$;
return $c$;
```

For the calculation of $h[v][u]$, we take the maximal $h[x][v], x \neq u$ increased by 1, or a value of 1 if no such *x* exists. Eventually, we use Equations 2 and 3 to determine the center vertex of the maximum urchin.

When analyzing the algorithm complexity, we observe that each for cycle is directed either by *m* or *n*. For the core while cycle, every edge is inserted to the queue *Q* exactly twice (once for each direction), and the adjacency list of each node is explored no more than twice (once when $d[v] = 1$, once when $d[v] = 0$). Since for any tree *T*, $m = n - 1 \in O(n)$, we derive the complexity of our algorithm to be $O(n)$.

Also, it is trivial to retrieve the avenue used to construct phalanx strategy *S* from Algorithm 1 by locating *c* and backtracing the highest branches for u_1, u_2.

4.3 Urchins and Avenues

In the previous section, we discussed the class of urchin subgraphs and its $ps(T)$. It appears vital to ask whether the maximal urchin U_k is unique in a given tree; that is, whether the vertex with maximal k^c is unique. A simple example illustrated in Figure 7 shows that the maximal urchin U_k is not necessarily unique. The concept of multiple urchins can be scaled in an obvious way. Nevertheless, we present a different observation with regards to multiple urchins of the same size.

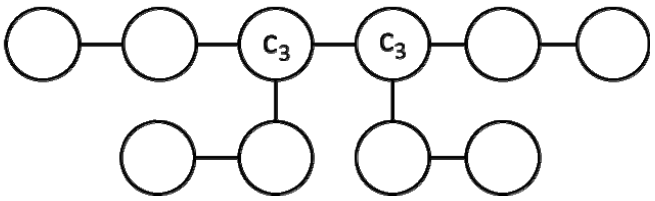

Figure 7. Centers of two U_3 in a single $T, ps(T)=3$

Proposition 2: *For any tree T with $ps(T)=k$, the centers $c_k, c'_k, \ldots$ of maximal urchins $U_k, U'_k, \ldots$ lay on a simple path.*

Proof: By contradiction. Consider three U_k, U'_k, U''_k with c_k, c'_k, c''_k as their respective centers. If the centers do not lay on a path, there must be some distinct vertex *v* such that each the center resides in a different branch rooted at *v*. Observe that all of the centers have at least two branches excluding *v* with height at least *k*. Hence, these centers along with their branches create an at least $k+1$ high branch at *v*. This makes *v* a center c_{k+1} of U_{k+1}, which contradicts our assumption that U_k is the maximum urchin. The situation is illustrated in Figure 8. **Q.E.D.**

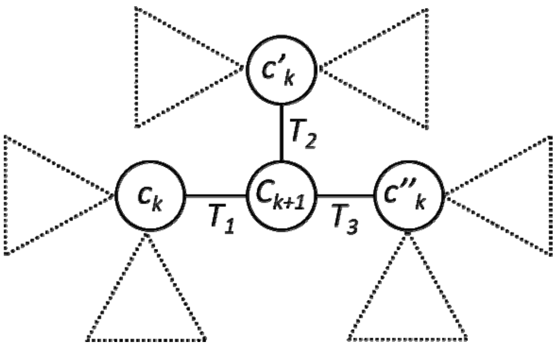

Figure 8: U_{k+1} is formed from c_k, c'_k, c''_k, with its branches $T_{1,2,3}$

From here, we have a clear idea of the Phalanx version of avenues or avenue systems, which control optimal search strategies for all major variants of graph searching on trees. In Phalanx search, the avenue is a path that traverses through all of the center of the largest urchin subgraphs, and for which the distance to every nonpath vertex is no more than $ps(T)$.

5 Minimum Spanning Trees

Since we expect that the Phalanx Search Problem can find applications in various areas of network security and incident containment, we proceed to combine it with the famous Minimum Spanning Tree (MST) Problem, which is a valuable tool in the field of network design. In order to simplify the following analysis, we will omit some very small graphs that are exceptions to the general rules.

We extend our previous research with the following definition:

Definition 4: Given an integer $k \geq 2$, the k-Phalanx Search Spanning Tree Problem (k-PSSTP) is defined as follows:

For a given graph G, does G have a spanning tree T with $ps(T) \leq k$?

We assume that this problem may find application in network security design and evaluation.

5.1 Complexity

Despite the existence of a linear algorithm for $ps(T)$ and an almost linear-time algorithm for $MST(G)$, we present the following theorem.

Theorem 4: *k-PSSTP is NP-hard.*

Proof: We will construct a reduction from the HAMILTONIAN PATH PROBLEM (HPP) to *k*-PSSTP. First, we construct an augmented graph G' by attaching a P_{k-1} every vertex in *G*. We claim that there is a Hamiltonian Path (HP) in *G* if and only if G' has a Spanning Tree *T*, $ps(T)=k$.

First, realize that if there is a HP in *G*, this path, along with the attached paths, forms an ST in G'. The HP can become the avenue of our search strategy, and since all of the non-avenue branches (namely, the augmented paths) are of height *k*, the search strategy will deploy no more than *k* searchers.

Conversely, if there exists some spanning tree *T* in *G* and a Phalanx strategy *S* of *T* with *k* searchers, we claim that its avenue contains all of the vertices from *G*, thus forming a HP in *G*. For the purpose of contradiction, we assume that the avenue does not contain all of the vertices from *G*. This implies that at least one vertex in *G* is not in the avenue. Furthermore, there must exist some vertex *c* that connects each one of some vertices v_1, v_2, and v_3 in *G* in a distinct branch rooted at *c*. In Figure 9, v_1 and v_2 are on the avenue but v_3 is not. Since there are *k*-1 vertex paths attached to all of the vertices v_1, v_2, and v_3, the three branches are both of height $\geq k$ and *c* forms a center c_{k+1} of a U_{k+1} in *T*. By Theorem 3, *T* is not *k*-searchable, which is a contradiction. Thus, we consider our reduction to be complete. **Q.E.D.**

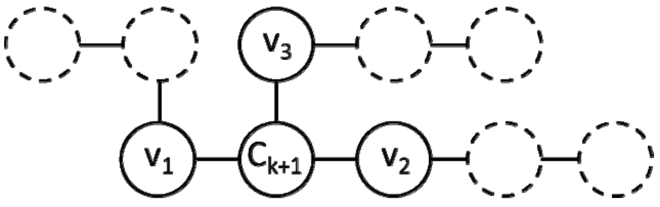

Figure 9. Formation of U_{k+1}, augmented path nodes dashed

Further, it is straightforward to show membership of *k*-PSSTP in *NP*.

Proposition 3: *k-PSSTP is in NP.*

Proof: We design a simple "guessing and checking" algorithm. First, nondeterministically guess the $n-1$ edges $E' \subseteq E$ that will be represent the spanning tree.

As for the checking phase, verify that $G[E']$ is connected and contains all vertices (thus also acyclic and forms a spanning tree). Use Algorithm 1 and return YES if $ps(G[E']) \leq k$. **Q.E.D.**

We are ready to conclude our investigation of the problem complexity.

Corollary 4. *k-PSSTP is in NP-complete.*

5.2 Approximating Minimum PSSTP

Previous research suggests using BSTree/Graph Center as a $O(\log n)$ approximation of node search spanning tree problem. With respect to this approximation, we have the following observation.

Proposition 4: *There are graphs on* $3k-2, k \geq 3$ *vertices for which the BSTree/Graph Center approximation creates* U_k*, while the optimal Phalanx search spanning tree is a Hamiltonian path* $P_{3k-2}, ps(P_{3k-2}) = 2$.

Proof: Assume a C_{2k-1} with a P_{k-1} attached to one of its vertices, *c*. Alternatively, take U_k and connect two of its leaves by an edge, producing the same graph. Clearly, *c* is the graph center, breadth first search tree originating from *c* creates U_k, yet there is a HP P_{3k-2} in the original graph, which implies $ps(P_{3k-2}) = 2$. See Figure 10 for an example. **Q.E.D.**

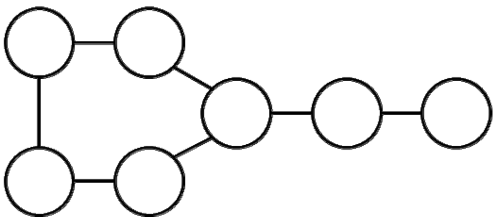

Figure 10. Graph combining C_5 with P_2 spawning U_3 on BFS

Observe that for the case investigated, the optimal solution is the best possible case on n vertices (2-searchable spannig tree), yet the approximation reaches the upper bound of tree search number on n vertices defined in Corollary 2. Therefore, in the worst case, we get the worst possible approximation ratio.

The discourse why the same approximation that yields a "satisfactory" $O(\log n)$ optimization for the node-search strategy yields a less desirable $O(n)$ approximation for Phalanx search is answered not by the specifics of Phalanx search, but rather by the size of the lower bounds on search numbers for both. For unconditioned node-search, the search number of a tree is bounded logarithmically, i.e. $s(T) \in O(\log n)$, while for phalanx search, this boundary is linear, $ps(T) \in O(n)$. Dually, the size of the minimum obstacle for unconditioned node search on trees grows exponentially with the search number (*k*-searchable tree contains 3 copies of $k-1$ searchable tree), while the same obstacle for Phalanx search grows linearly (extending urchin by 3 extra vertices increases the search number).

To prove our point, we take subtree gadget $T^k, k \geq 4$ as defined in [7]. Mark *c* to be the central vertex and consider it a root of T^k. Furthermore, let v_1 be the *i*-th vertex visited by level-order traversal of T^k. Create an augmented graph $\overline{T}^k$ by adding an edge between each v_i, v_{i+1} that lay on the same level. Eventually, delete the edge $\{v_3, v_4\}$. An example of $\overline{T}^k$ is illustrated in Figure 11.

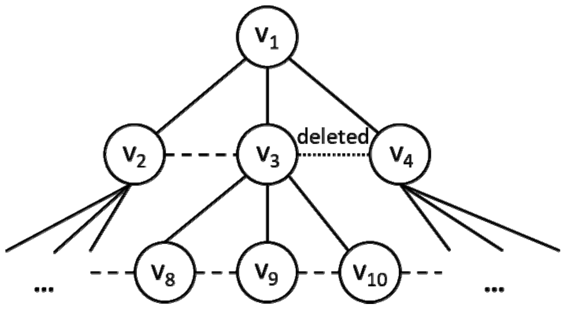

Figure 11. $\overline{T}^k$, deleted edge dotted, added edges dashed

Theorem 5: *Vertex c is still center of* T^k. *Furthermore, there is a HP in* $\overline{T}^k$*, therefore* $s(\overline{T}^k) = 2$*, but the breadth first search tree from vertex c produces* T^k.

Proof: Clearly, *c* is still the center of $\overline{T}^k$ since the distance from any other vertex to either the leftmost or the rightmost vertex in the lowest level is always larger than the tree height.

Moreover, our HP can start with $v_3, v_2, v_1 = c, v_4$. Then, iteratively descend into the leftmost/rightmost vertex of the next level and traverse all of its vertices using the augmented edges, until arriving at the last vertex.

Finally, since all of the edges added during the augmentation process connect vertices on the same level and thus with the same distance from *c*, they do not interfere with construction of BST from c. **Q.E.D.**

Hence we conclude that the approximation factor of BSTree/Graph Center is not a consequence of suitability of this approximation, but rather of the order of growth of the lower bound on the search number.

Instead of finding another, more suitable approximation, we make other observation of the spanning trees. For both k-SSTP [17] and k-PSSTP, we used reduction from Hamiltonian Path Problem to prove its hardness. Hamiltonian Path Problem is a specific case of the Longest Path Problem, which is notoriously famous for being tremendously difficult to approximate. While we do not have a clear link to this problem on hand, we provide some evidence to support our conjecture.

Definition 5: *For any G, we define the k-Maximum Phalanx Search Spanning Tree Problem (k-MxPSSTP) as follows: Is there a spanning tree T of G such that* $ps(T) \geq k$ *?*

It comes with no surprise that this dual problem is hard as well.

Theorem 6: *The k-MxPSSTP is NP-hard.*

Proof: Denote n the number of vertices in G. Create augmented G^+ by adding a P_{2n+1} and connecting the $(n+1)$-th (middle) vertex P_{n+1} of the path with every other vertex in the original G. Figure 12 illustrates this process.

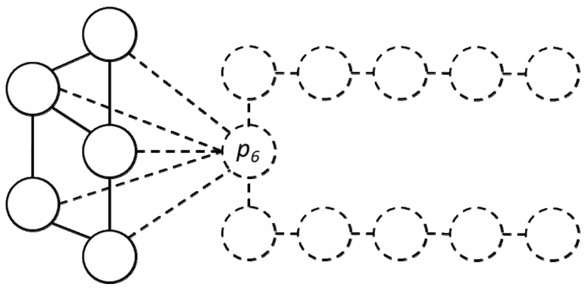

Figure 12. Augmented G^+ for a sample graph G on five vertices, augmentation in dashed stroke

It is easy to see that there is a HP in G if and only if there is a spanning tree T^+ in $G^+, ps(T^+) = n+1$. The Hamiltonian Path finalizes the third branch of U_{n+1} centered at p_{n+1}. **Q.E.D.**

The interesting part about this problem is the fact that it directly extends to the longest path problem.

Corollary 5: *Any approximation of MsPSSTP in* G^+ *is an approximation of the Longest Path problem in G.*

Hence the boundaries for graph search spanning trees and their approximations are yet to be established.

6 Conclusion

In this article we have addressed a potential problem between some graph searching problems and their real-life counterparts by introducing a novel search variant, Phalanx Graph Search. We believe that its main distinction in its simplicity has the potential to outperform existing versions of graph searching in various applications. We have also introduced the connection of this search problem with minimum spanning trees, extending previous results and setting up new milestones and observations.

There are numerous directions for future research. Most importantly, the effect of monotony on Phalanx Search Number is currently unknown. Furthermore, its relation to connected search and the tidiness property could be explored. In addition, we suggest that a combination with other restrictions as well as binding Phalanx Search Number in the graph search problem hierarchy may yield valuable insights.

Finally, let us highlight one inspiring remark of our research. We have shown that certain properties and concepts applicable to other search variants have their counterparts in Phalanx Search. Moreover, the Phalanx Search analogies are often smaller and simpler, which makes them easier to define and examine. Therefore, we believe that Phalanx Search, so to say, bears many characteristics of the original search variants, yet in a much more humble garment. Therefore, studying problems that are open for other search variants on Phalanx Search and later translating the results into the original search variant may help further the progress of the state of the art knowledge in this field.

Acknowledgements

This research is partially funded by the Thailand Research Fund grant number RSA6080029. The second author is thankful for this support.

References

[1] R. Breisch, An Intuitive Approach to Speleotopology, *Southwestern Cavers*, Vol. VI, pp. 72-78, June, 1967.

[2] T. D. Parsons, Pursuit-evasion in a Graph, in: Y. Alavi, D. R. Lick (Eds.), *Theory and Applications of Graphs, Lecture Notes in Mathematics*, Springer, Berlin, Heidelberg, 1978, pp. 426-441.

[3] P. A. Golovach, Equivalence of Two Formalizations of a Search Problem on a Graph, *Vestnik Leningradskogo Universiteta Seriya Matematika Mekhanika Astronomiya,* pp. 10-14, March, 1989.

[4] P. A. Golovach, A Topological Invariant in Pursuit Problems, *Differential Equations,* Vol. 25, No. 6, pp. 657-661, May, 1989.

[5] M. Maamoun, H. Meyniel, On a Game of Policemen and Robber, *Discrete Applied Mathematics*, Vol. 17, No. 3, pp. 307-309, June, 1987.

[6] L. Blin, P. Fraigniaud, N. Nisse, S. Vial, Distributed Chasing of Network Intruders, in: P. Flocchini, L. Gąsieniec (Eds.), *Structural Information and Communication Complexity,*

Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2006, pp. 70-84.

[7] M. Kress, K. Y. Lin, R. Szechtman, Optimal Discrete Search with Imperfect Specificity, *Mathematical Methods of Operations Research*, Vol. 68, No. 3, pp. 539-549, December, 2008.

[8] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, C. H. Papadimitriou, The Complexity of Searching a Graph, *Journal of the ACM*, Vol. 35, No. 1, pp. 18-44, January, 1988.

[9] L. M. Kirousis, C. H. Papadimitriou, Searching and Pebbling, *Theoretical Computer Science*, Vol. 47, pp. 205-218, January, 1986.

[10] A. S. LaPaugh, Recontamination Does Not Help to Search a Graph, *Journal of the ACM*, Vol. 40, No. 2, pp. 224-245, April, 1993.

[11] L. Barriere, P. Flocchini, P. Fraigniaud, N. Santoro, Capture of an Intruder by Mobile Agents, *Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, Winnipeg, Manitoba, Canada, 2002, pp. 200-209.

[12] B. Yang, D. Dyer, B. Alspach, Sweeping Graphs with Large Clique Number, *Discrete Mathematics*, Vol. 309, No. 18, pp. 5770-5780, September, 2009.

[13] S.-L. Peng, C.-W. Ho, T.-S. Hsu, M.-T. Ko, C.-Y. Tang, A Linear-Time Algorithm for Constructing an Optimal Node-Search Strategy of a Tree, *Computing and Combinatorics*, Taipei, Taiwan, 1998, pp. 279-289.

[14] S.-L. Peng, *A Study of Graph Searching on Special Graphs*, Ph.D. Thesis, National Tsing Hua University, Hsinchu, Taiwan, 1999.

[15] L. Barriere, P. Flocchini, F. V. Fomin, P. Fraigniaud, N. Nisse, N. Santoro, D. M. Thilikos, Connected Graph Searching, *Information and Computation*, Vol. 219, pp. 1-16, October, 2012.

[16] C.-F. Lin, O. Navrátil, S.-L. Peng, On the Cooperative Graph Searching Problem, *Structured Object-Oriented Formal Language and Method*, Xi'An, China, 2017, pp. 39-47.

[17] N. Juneam, O. Navrátil, S.-L. Peng, On the Node Searching Spanning Tree Problem, *Journal of Computers*, Vol. 29, No. 1, pp. 160-165, February, 2018.

Biographies

Ondřej Navrátil is a Ph.D. Candidate at NDHU, Taiwan, graduated Master in Mathematical Methonds in CS at FIT BUT, Czechia. Expertise in Graph Theory, Complexity and Formal Languages. Experienced full stack developer, currently working as a C++ developer for Monet+, Czechia.

Sanpawat Kantabutra earned his Ph.D. in Theoretical Computer Science from Tufts University in the United States of America. He is currently an associate professor in the Theory of Computation Group in Chiang Mai University, Chiang Mai, Thailand, and a Thailand Research Fund scholar.

Sheng-Lung Peng is a Professor at Department of Computer Science and Information Engineering at National Dong Hwa University, Taiwan. He received the Ph.D. degree in Computer Science from the National Tsing Hua University, Taiwan. He is an honorary Professor of Beijing Information Science and Technology University, China, and a visiting Professor of Ningxia Institute of Science and Technology, China. He is also an adjunct Professor of Mandsaur University, India. Dr. Peng has edited several special issues at journals, such as Soft Computing, Journal of Internet Technology, Journal of Real-Time Image Processing, International Journal of Knowledge and System Science, MDPI Algorithms, and so on. His research interests are in designing and analyzing algorithms for Bioinformatics, Combinatorics, Data Mining, and Networks areas in which he has published over 100 research papers.