Journal of Computers Vol. 29 No. 1, 2018, pp. 160-165
doi:10.3966/199115992018012901014

On the Node Searching Spanning Tree Problem

Nopadon Juneam[1], Ondrej Navrátil[2], Sheng-Lung Peng[2*]

[1] Department of Computer Science, Faculty of Science, Kasetsart University, Bangkok 10900, Thailand
nopadon.j@ku.ac.th

[2] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien 97401, Taiwan
{810321005, slpeng}@ndhu.edu.tw

Received 22 July 2017; Revised 22 December 2017; Accepted 8 January 2018

Abstract. Spanning tree is an extensively studied topic in the field of graph theory. Graph searching, on the other hand, is a novel and modern concept that extends the notion of traditional graph traversal. This paper introduces a problem which is a combination of spanning tree and *node searching*, a basic variant of graph searching. Our problem is called *node searching spanning tree problem*. In this problem, we are interested in a spanning tree regarding its *search number*. Let G be a simple undirected graph and $s(G)$ denote the search number of G, *i.e.*, the minimum number of searchers needed to traverse the graph G. Indeed, the node searching spanning tree problem considers a spanning tree T of G such that $s(T)$ is minimized. In this paper, we show that to decide whether there exists a spanning tree $s(T)$ of G such that $s(T) \leq k$ is NP-complete, for any fixed integer $k \geq 2$. In addition, we propose an $O(\log n)$-approximation algorithm for the node searching spanning tree problem on graphs with n vertices.

Keywords: graph searching, node searching, node searching spanning tree, NP-complete, spanning tree

1 Introduction

Let $G = (V, E)$ be a simple undirected graph where V is the set of n vertices and E is the set of m edges. A *spanning tree* of graph G is a connected subgraph of G that is a tree and contains every vertex of G. A graph G may have several spanning trees in general. The number of all possible spanning trees of G may vary depending on the graph structure of G. For example, if G is a tree, then G has only one spanning tree; if G is a cycle, then G has n possible spanning trees. However, the number of spanning trees in some graphs can be very large, namely, beyond polynomial in the number of the graph vertices. For example, a complete graph of n vertices has n^{n-2} spanning trees (by Carley's formula). In fact, conventional graph traversal techniques such as breath-first search (BFS) and depth-first search (DFS) both can give a spanning tree of a graph G in linear time, assuming G is connected. In optimization, it is often desirable to find a particular spanning tree of a given graph regarding some objective functions defined over all possible spanning trees. For example, when each edge of the graph G has an associated numerical value called *edge weight*, a famous optimization problem called *minimum spanning tree problem* is to find a spanning tree of G whose sum of its edge weights is at the minimum. The minimum spanning tree problem was known by efficient algorithms for minimum spanning tree construction due to the independent works of Kruskal [1] and Prim [2]. Although the minimum spanning tree problem can be solved in polynomial, most other optimization problems related to spanning tree in decision version appeared to be NP-complete according to the literature [3]. These are problems such as finding a spanning tree with the maximum number of leaves, finding a spanning tree with maximum vertex degree, etc.

This paper introduces an optimization problem·which is a combination of spanning tree and *node*

[*] Corresponding Author

searching, a basic variant of graph searching. The problem is called *node searching spanning tree problem*. Essentially, graph searching is a modern concept that extends the notion of traditional graph traversals. In graph searching we are given a simple undirected graph G in which initially all the edges of G are in the *contaminated* state. To change the state of edges from contaminated to *clear*, a set of clearing units, called *searchers,* must be sent to traverse the edges. The goal of graph searching is to reach the final stage in which all the edges are eventually in the clear state. Many fundamental questions about graph searching concern how many searchers are necessary to guarantee that it is possible to achieve the final stage. This quantity is defined in terms of the *search number* of graph. In this paper, we are interested in a spanning tree of graph G whose search number is at the minimum.

The graph searching problem was first mentioned by Breisch [4] in an unrelated Caver's journal as a need of a strategy to rescue a lost person in the system of caves. Parsons [5] and Petrov [6] later formalized the notion of this problem independently. As the field of graph searching emerged, several variants of graph searching including *node searching* [7] and *mixed searching* [8] were introduced, while the most common crucial result in this field is that most graph searching problems regarding the search number are known to be NP-complete [5, 9]. There have also been a number of research devoted to efficient algorithms for certain graph classes such as trees [9-12].

For this work, we will focus on node searching, a basic variant of graph searching. Let G be a simple undirected graph and $s(G)$ denote the search number of G, *i.e.*, the minimum number of searchers needed to traverse the graph G. In the node searching spanning tree problem, we consider a spanning tree T of G such that $s(T)$ is at the minimum. To discuss the complexity of the node searching spanning tree problem, we will define a decision version of the problem called k-SEARCHABLE SPANNING TREE PROBLEM. For a fixed positive integer k, the decision version is to decide whether there exists a spanning tree $s(T)$ of G such that $s(T) \leq k$. We show this decision problem is NP-complete for $k \geq 2$. In addition, we propose an $O(\log n)$-approximation algorithm for the node searching spanning problem.

The rest of this paper is organized as follows: The preliminaries and definitions related to the node searching spanning tree problem are given in Section 2. We present the NP-completeness proof of the k-SEARCHABLE SPANNING TREE PROBLEM in Section 3. Our proposed approximation algorithm is discussed in Section 4. Finally, the conclusion of this work is given in Section 5.

2 Preliminaries and Definitions

This section provides definitions and notation which will be used to define the node searching spanning tree problem as well as its decision variant. Some additional notation and terminologies that are useful for the later discussion are also given at the end of this section. Throughout this paper, unless mentioned otherwise, we let $G = (V, E)$ be a simple undirected graph with $|V| = n$ vertices and $|E| = m$ edges. Let the set of vertices of G be denoted by $V(G)$ and the set of edges of G denoted by $E(G)$. For a subset $S \subseteq V$, let $G[S]$ denote a subgraph induced by S in G.

2.1 Node Searching

Given a graph G, we define a *search step* on G as one of the following actions: place a searcher on a node $v \in V$; or remove an existing searcher from node v. At any moment, an edge $e \in E$ is in exactly one of the possible states: either *clear* or *contaminated.* In the initial stage, all the edges of G are contaminated. A *search strategy* is defined as a series of valid search steps of G that results in the final stage in which all the edges are clear. In node searching, an edge $e = \{u, v\}$ changes its state from contaminated to clear *if and only if* both of its end vertices u and v contain a searcher. There is also a reverse process called *recontamination* in which a clear edge may change its state to contaminated. More precisely, an edge e changes its state from clear to contaminated *if and only if* there exists a path from an end vertex of e to an end vertex of some contaminated edge e' without a searcher on any of the path's vertices.

Given a search strategy S for G, we define the *cost* of S, denoted by $c(S)$, as the maximum number of searchers which present in G at any of its steps. The *search number* of G, denoted by $s(G)$, is the minimum cost of all possible strategies for G. A search strategy S is *optimal if and only if* $c(S) = s(G)$. For any integer k, we say that a graph G is k-searchable *if and only if* $s(G) \leq k$. Observe that $s(G) = 1$ *if and only if* all the vertices of G are isolated.

2.2 Node Searching Spanning Tree

Given a graph $G = (V, E)$, we define a *spanning tree* of G, as any of its connected subgraph $T = (V, E')$ such that T is a tree, *i.e.*, T has no cycle. A spanning tree T of G does not need to be unique. Let Γ denote the set of all possible spanning trees of G. The node searching spanning tree problem considers a spanning tree T of G such that $s(T) = min_{\tau\in\Gamma}\{s(\tau)\}$.

Let k be a fixed positive integer, we define a decision variant of the node searching spanning tree problem called k-SEARCHABLE SPANNING TREE PROBLEM (or k-SSTP for short). The definition of the problem is as follows:

***k*-SEARCHABLE SPANNING TREE PROBLEM (*k*-SSTP):**
Given: a graph $G = (V, E)$ and positive integer k.
Problem: Does G·have a spanning tree T such that $s(T) \leq k$?

2.3 Avenue and Hub

The notions of hub and avenue were first introduced in [11, 12]. Hub and avenue are two crucial components of graph searching on trees as they can lead to an optimal search strategy or the computation of the search number. Intuitively, an avenue is a path that splits the tree into subtrees that can be clear by using fewer searchers. Similarly, a hub is simply a vertex that disconnects the tree into subtrees that can be cleared by using fewer searchers. In what follows we give the precise definition of hub and avenue.

Let $T = (V, E)$ be a tree. For any vertex u with edge $\{u, v\}$, we define a *branch* T_{uv} as the connected component in graph $T[V\backslash\{u\}]$ which includes v in its vertex set. Note that T_{uv} is simply a subtree of T. Let P be a path in T. For $u\in P$, we call T_{uv} *a path branch* if $v\in P$; otherwise, T_{uv} is called a *non-path branch.* A path P is called an *avenue* if for every path branch B at P, $s(B) = s(T)$; and for every non-path branch B' at P, $s(B') < s(T)$. A vertex u is called a hub of T if every branch T' at u satisfies $s(T') < s(T)$. Note that a tree has either a hub or a unique avenue (see the result in [9, 12]).

3 NP-Completeness Results

We will present the NP-completeness proof of the k-SSTP in this section. Our goal is to show that the k-SSTP is NP-complete for $k \geq 2$. First, we begin to show that the k-SSTP is in NP for any fixed $k \geq 2$.

Theorem 1. For any fixed $k \geq 2$, k-SSTP $\in$ NP.
Proof. For a given graph G = (V, E), we can use a subgraph H = (V, E') of G, where E' $\subseteq$ E, as a certificate for G. The verification algorithm affirms that H is a tree. Moreover, the search number of H can be computed in linear time using algorithms by Peng et al. [12].

Next, we will establish the NP-hardness results for the k-SSTP. The NP-hardness of the k-SSTP follows from a polynomial-time reduction from the HAMILTONIAN PATH PROBLEM (or HPP for short), one of the famous NP-complete problems [3]. Recall that a *Hamiltonian path* is a path that visits every vertex of the graph exactly once. The HPP is to decide whether there exists a Hamiltonian path in a given graph.

In the remaining of this section, we will let $G = (V, E)$ denote a graph instance of the HPP. Before presenting the reduction, we would like to give some outline of the idea behind it first.

Our reduction transforms a graph G to a corresponding graph $G' = (V', E')$. The construction of G' is to grow a k-searchable subtree at each vertex $u\in V$ so that u becomes a hub of the subtree. In other words, our reduction constructs G' in such a way that $s(T_{uv}) = k$, for any corresponding subgraph (branch) T_{uv} of graph G' to an edge $\{u,v\}\in E$. If a Hamiltonian path P exists in G, then P will denote an avenue of any spanning tree T of G' with $s(T) = k$. Conversely, if G' has no Hamiltonian path, then G must include a complete bipartite graph $K_{1,3}$ (or a *claw*) as its subgraph. This induced claw will appear as a subtree of any spanning tree T of G'. Moreover, the center of the claw will denote a hub of T such that $s(T) \geq k+1$. Hence, G has a Hamiltonian path if and only if G' has a k-searchable spanning tree.

Theorem 2. For any fixed $k \geq 2$, HPP $\leq_p$ k-SSTP.
Proof. Let k $\geq$ 2 be a constant. We construct a graph G' = (V', E') from a graph G = (V, E) as follows:

Let T^k denote a subtree gadget such that $s(T^k) = k$. Let T^2 be a tree that contains only one edge as the base with $s(T^2) = 2$. For $k \geq 3$, we can construct a subtree gadget T^k by introducing a new vertex and connecting three copies of T^{k-1} to the new vertex individually. It is clear that the new vertex will denote a hub of subtree T^{k-1} with $s(T^k) = k$. Moreover, T^k contains $O(3^k)$ vertices. Fig. 1. depicts the construction of subtree gadgets T^2, T^3, and T^4.

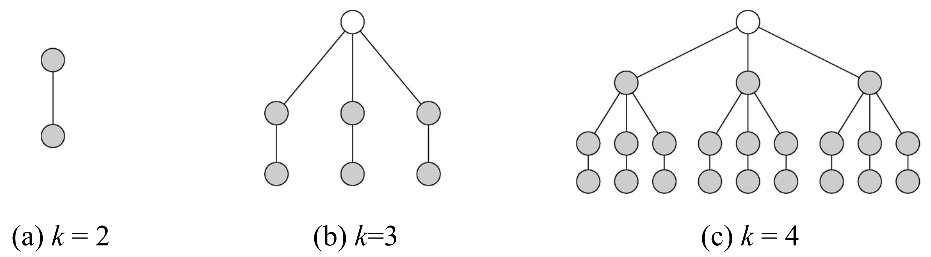

Fig. 1. The construction of subtree gadget T^k

For each vertex $u \in V$, we construct three copies of subtree gadget T^{k-1} and then connect u to any vertex of each copy by·an edge. Therefore, $|V'| = O(3^k n)$ and $|E'| = O(3^k n) + m$. Since k is constant, the construction of graph G' can be done in $O(n+m)$ time. Fig. 2 illustrates an example of the construction of graph G'.

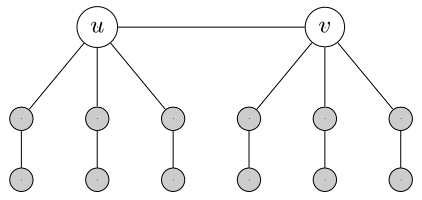

Fig. 2. The construction of G' for an edge $\{u, v\} \in E$ regarding $k = 3$

Suppose a Hamiltonian path P exists in G. Let $E(P)$ denote the edge set of such path. Then, a spanning tree T of G' is formed by $T = (V', (E(G') \setminus E(G)) \cup E(P))$. For an edge $\{u, v\} \in E(P)$, let B_{uv} denote the corresponding connected component in $T[V' \setminus \{u\}]$ that includes v in its vertex set. Clearly, B_{uv} is a path branch at P in G'. Further, let $B'_{uv} = T[V(T_{uv}) \setminus \{v\}]$. Observe that B'_{uv} is simply a forest that contains three copies of T^{k-1}, each of which denotes a non-path branch at P. By construction, $s(B_{uv}) = k$ and $s(B'_{uv}) = k-1 < k$, for all $\{u, v\} \in E(P)$. Therefore, path P is an avenue of T with $s(T) = k$.

Conversely, suppose G has no Hamiltonian path. Then, G must contain a claw $K_{1,3}$ as its subgraph and this induced claw must also appear as a subtree of any spanning tree T of G'. Let C denote such claw in G. For an edge $\{u,v\} \in E(C)$, let B_{uv} denote the corresponding connected component in $T[V' \setminus \{u\}]$ that includes v in its vertex set. That is, B_{uv} is a branch at u. By construction, $s(B_{uv}) = k$, for all $\{u, v\} \in E(C)$. Let $B_u = T[\bigcup_{\{u,v\} \in E(C)} V(B_{uv})]$ denote the subtree of T induced by the set of vertices in every branch at u. Observe that $s(B_u) = k+1$. Thus, u is a hub of B_u with $s(B_u) = k+1 > s(B_{uv}) = k$, for all $\{u, v\} \in E(C)$. Since any spanning tree T must contain C as its subtree, it follows that $s(T) \geq k+1$. Hence, G·has a Hamiltonian path if and only if G' has a k-searchable spanning tree. ■

Theorem 3. For any fixed $k \geq 2$, the k-SSTP is NP-complete.

Proof. This result follows Theorem 1 and Theorem 2. ■

4 Approximation Algorithm

In this section, we propose an $O(\log n)$-approximation algorithm for the node searching spanning tree problem. Our approximation approach is based on the notion of using hub as a base for constructing a spanning tree.

As an entry point of our proposed approximation algorithm, we will examine a given graph $G = (V, E)$ thoroughly. For any pair of vertices $u, v \in V$, we compute the distance $d(u, v)$ in $O(n^3)$ time using the well-known Floyd-Warshall algorithm. For each node $u \in V$, we compute the *eccentricity* of *u*, $e(u) = max_{v \in V}\ d(u, v)$, in $O(n^2)$ time. We determine the graph's radius $r(G) = min_{u \in V} e(u)$ and the graph's centers $c(G) = \{u \in V \mid e(u) = r(G)\}$, all in $O(n)$ time.

Let T^* denote an optimal spanning tree such that there exists a search strategy S for T^* with $c(S) = s(T^*)$. Let us assume that T^* has a hub. Our approximation algorithm guesses a proper node to be a hub. The guesses are based on the notion of hub that a hub should disconnect the tree into a collection of smaller subtrees that can be cleared by using the fewer number of searchers. First, we will try to find a node that trends to behave this way. To make a good guess, we will select a maximum degree node from the graph's centers as our hub. Then, we will construct a spanning tree by using a BFS tree rooted at this node. Our approximation algorithm is given in the following.

HUB BASED APPROXIMATION ALGORITHM:

```
Input: a graph $G$.
Output: a spanning tree $T$ of $G$.
Method:
Step 1: Compute the centers $c(G)$.
Step 2: $c = max_{u \in c(G)} deg(u)$.
Step 3: $T$ = BFSTree($G$, $v$).
```

Theorem 4. The approximation ratio of the HUB BASED APPROXIMATION ALGORITHM is $O(\log n)$.

Proof. Let T be a spanning tree constructed by the HUB BASED APPROXIMATION ALGORITHM. Since T is a BFS tree rooted at the center c, the height of T will never exceed r(G). To clear T, we put a searcher on c. Then, we recursively clear a branch at c by treating c as a hub. Thus, to clear a branch of height r(G), we need to use $f(r(G)) = f(r(G)/2)+1$ searchers. By solving recurrence, we obtain $s(T) = f(r(G)) = O(\log r(G))$. Moreover, $s(T^*) \geq 2$ whenever $|V(G)| > 1$. It follows that $s(T)/s(T^*) \leq O(\log r(G))/2 \leq O(\log n)$. The theorem holds.

5 Conclusion and Discussion

This paper introduced a problem which is a combination of spanning tree and node searching, a basic variant of graph searching. The problem is called node searching spanning tree problem. We consider a spanning whose *search number* is at the minimum in this problem. To discuss the problem's complexity, we defined the problem in decision version of the problem called *k*-SEARCHABLE SPANNING TREE PROBLEM (*k*-SSTP). For a fixed positive integer *k*, the *k*-SSTP is to decide whether there exists a spanning tree *T* of *G* such that $s(T) \leq k$. We showed that the *k*-SSTP is NP-complete for $k \geq 2$. In addition, to cope with the NP-hardness of the node searching spanning tree problem, we proposed an $O(\log n)$-approximation algorithm.

Since this paper is one of the first written that ties together spanning tree and graph searching, there are numerous opportunities for future work. First, it is worthwhile to expand the scope of research further to other basic variants of graph searching. Accordingly, we believe that there will be more interesting results certain graph classes that are likely to be motivated by fields of application such as logistics, networking, or security. Finally, the approximation algorithm can be improved.

Acknowledgements

The work was partially supported by the Ministry of Science and Technology of Taiwan under Contract No. MOST 104-2221-E-259-006.

References

[1] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical Society 7(1956) 48-50.

[2] R.C. Prim, Shortest connection networks and some generalizations, Bell System Technology Journal 36(1957) 1389-1401.

[3] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, 1990.

[4] R. Breisch, An intuitive approach to speleotopology, Southwestern Cavers (Published by the Southwestern Region of the National Speleological Society) VI(1967) 72-78.

[5] T.D. Parsons, The search number of a connected graph, Proceedings of the Ninth Southeastern Conference on Combinatorics, Graph Theory, and Computing XXI(1978) 549-554.

[6] N.N. Petrov, A problem of pursuit in the absence of information on the pursued, Differentsial'nye Uravneniya 18(1982) 1345-1352.

[7] L.M. Kirousis, C.H. Papadimitriou, Interval graphs and searching, Discrete Math. 55(1985) 181-184.

[8] D. Bienstock, P. Seymour, Monotonicity in graph searching, J. Algorithms 12(1991) 239-245.

[9] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, C.H. Papadimitriou, The complexity of searching a graph, Journal of the Association Computing Machinery 35(1988) 18-44.

[10] A. Takahashi, S. Ueno, Y. Kajitani, Mixed searching and proper-path-width, Theoretical Computer Science 137(1995) 253-268.

[11] 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, COCOON'98, Lecture Notes in Computer Science 1440(1998) 279-288.

[12] S.-L. Peng, C.-W. Ho, T.-s. Hsu, M.-T. Ko C. Y. Tang, Edge and node searching problems on trees, Theoretical Computer Science 240(2000) 429-446.