Polynomial-Time Algorithms for Path Movement Problems on Trees and Unicyclic Graphs

Varin Chouvatut, Wattana Jindaluang

Department of Computer Science, Chiang Mai University, Thailand
{varinchouv, wjindaluang}@gmail.com

Abstract

In the *path movement problem*, we are given an undirected graph $G = (V, E)$, a source vertex s, a destination vertex t, and a set of movable objects, called *pebbles*, which are placed on a subset of the vertices of a graph. We want to move a subset of the pebbles along the edges to a subset of the vertices of the graph such that there exists a path from s to t in graph G in such a way that all the vertices in that path are occupied by at least one pebble. The PathMax and the PathSum problems are the path movement problems in which we want to minimize the maximum movements and the total movements made by the pebbles, respectively. In [7], the researchers proved that both the problems are NP-hard in general graphs. In this paper, the PathMax and the PathSum problems are considered on special classes of graph G, that is, G is a tree and a unicyclic graph. More precisely, this paper shows that PathMax and PathSum problems can be easily solved in polynomial time when the input graph is a tree or a unicyclic graph.

Keywords: Polynomial-time algorithm, Movement problem, Tree, Unicyclic graph

1 Introduction

Motion planning is the problem in which we want to move a set of movable objects, for example, robots, people, sensor nodes, etc., from their initial positions to their final positions such that the movements made by the movable objects obey some constraint. And the problem goal is to optimize some measurement. For example, the two sensor nodes in a wireless sensor network (WSN) want to communicate with each other for cooperative communication but they have limited communication ranges. Therefore, in this communication, the movable sensor nodes have to move to a communication path to relay a signal (or a message) from the source node to the destination node. An example of a multiple mobile relays research is [1]. In this research, the authors proposed the low cost, low complexity, and collision-free algorithms with the condition relaxations of the other existing methods for the problem of position optimization in mobile relays. Moreover, motion planning is implemented in wireless sensor networks in terms of mobile sensor networks, mobile opportunistic networks, and mobile ad hoc networks. There are many pieces of research about a barrier coverage problem in mobile sensor networks such as [2] where the mobile sensor nodes have to move for minimizing the number of barrier paths in such a way that the total moving distances made by the mobile nodes and the number of sensor nodes are minimized. In [3], Jia et al. proposed the theoretical analysis and simulations for redistributing mobile sensor nodes to guarantee that a barrier coverage property is satisfied. In an area of opportunistic networks, Wang et al. [4] considered the contact duration between mobile terminals and their adjacent nodes with respect to the node mobility and short communication range. In the rescue situations, the rescuers are assisted by the robots to access to hazardous regions. In [5], Baroudi et al. provided an approach with a simulation of a dynamic coverage problem in situations with the constraint of connectivity, the shortage of available robots, and a time limitation. Finally, in a mobile ad hoc network, Yu et al. [6] presented a broadcasting algorithm to prevent a broadcast storm problem and increase the broadcast reliability and network lifetimes by using a broadcasting ratio and energy balance of nodes.

In theoretical computer science, motion planning is studied and defined as a *movement problem*. Demaine et al. [7] were the first group of researchers who introduced the class of movement problem. In this problem, we are given a simple undirected unweighted graph $G = (V, E)$ where V is a set of n vertices and E is a set of m edges. In addition, we are given a set of pebbles which are placed on the set of vertices of a graph. The pebbles can move along the edges of the graph. The goal of the movement problem is to move the set of pebbles from the initial positions of the pebbles to a set of vertices such that the pebble movement satisfies some criteria such as the induced subgraph on the final positions of the pebbles is a connected graph: this problem's name will start with

*Corresponding Author: Wattana Jindaluang; E-mail: wjindaluang@gmail.com
DOI: 10.3966/160792642019102006005

Con; there is at the most one pebble occupying each vertex and for each adjacent vertex, there is at the most one vertex that is occupied by a pebble: this problem's name starts with *Ind*; and there exists a path from the source vertex *s* to the destination vertex *t* and each vertex in this path is occupied by at least one pebble: this problem's name starts with *Path*. The cost movement measurements are carried out in many ways, such as minimizing the maximum movement, that is, the maximum distance moved by any pebble; minimizing the overall movement, that is, the sum of the distances traversed by all the pebbles; and minimizing the number of moving pebbles, that is, the number of pebbles that move from their initial positions. For these three measurements, the names of the problems will end with *Max*, *Sum,* and *Num*, respectively (for more details, see [7]).

In a graph theory, a finite sequence of distinct vertices which connected by the edges is called a *simple path*. A simple path that begins and ends with the same vertex is called a *cycle*. A graph is a *connected graph* if there exists a path between any its two vertices. A *tree* is a connected graph with no cycle. A *unicyclic graph* is a connected graph that comprises exactly one cycle.

1.1 Related Work

The class of movement problems was first defined by Demaine et al. [7]. This paper showed that most of them are NP-hard and also provided the approximation algorithms for them. Later, various other versions of the movement problems were studied and simultaneously the approximation ratios of some movement problems were improved through many research papers such as [8-10], and [11].

With regard to the aspect of the easy problem, there are many research papers that discussed the polynomial-time algorithms for the movement problems. In [7], Demaine et al. showed that a PathNum problem can be solved in polynomial time on *n*. The movement problem where the final positions of the moved pebbles agree with the property that there exists a perfect matching of the pebbles in a graph where two pebbles are placed on two adjacent vertices if they are far from each other by at most one hop in the input graph and we want to minimize the maximum movement is called a *MatchMax* problem. As for the purposes of minimizing the overall movement and the number of moved pebbles of the same property, the movement problems are called *MatchSum* problem and *MatchNum* problem, respectively. Demaine et al. [7] showed that MatchMax, MatchSum, and MatchNum can solve in polynomial time on *n*. Additionally, they showed that when we fix an input graph to a tree, a ConMax problem can be solved in polynomial time. In 2016, Bilo et al. [12] presented a new category of movement problems and showed that a ConSum problem and a ConNum problem on a tree can be efficiently solved by the dynamic programming technique. What this means is that this result combined with the ConMax result in [7] can complete all versions of the connectivity property when the input graph is a tree. In addition, Bilo et al. [12] presented efficient algorithms for solving an IndMax on a path and an IndSum and an IndNum on a tree.

1.2 Contributions

This paper focuses on a PathMax problem and a PathSum problem when the input graph is a tree or a unicyclic graph. The paper shows that both a PathMax problem and a PathSum problem can be solved in polynomial time on both the input graphs. For the tree, we use the maximum cardinality matching algorithm and the minimum weighted matching algorithm as the subroutine for the PathMax problem and the PathSum problem, respectively. We proceed by showing that our algorithm's running times are dominated by the running times of the matching algorithms which run in the polynomial time on *n*. Thus, we can conclude that our proposed algorithms are efficient algorithms for solving a PathMax and a PathSum on a tree. For the unicyclic graph, we prove that the number of the simple path between two specific vertices is at most two. So, we can find the optimum movement on each path by calling the algorithms for the PathMax problem and the PathSum problem on the tree as the subroutine. Finally, we choose the best one among them as the solution of the PathMax problem and the PathSum problem on a unicyclic graph.

The paper is organized as follows. Section 2 gives the notations and the terminology used in this paper. The polynomial-time algorithm for a PathMax problem on a tree is shown in section 3. Section 4 presents the polynomial-time algorithm for a PathSum problem on a tree. The polynomial-time algorithms for the PathMax problem and the PathSum problem on the unicyclic graph are presented in section 5. Finally, the conclusion is provided in section 6.

2 Notations and Terminology

The basic notations and terminology used by this paper are defined in this section. In this paper, we are given a simple undirected unweighted graph $G = (V, E)$. It is composed of a collection V of n vertices and a collection E of m edges. In addition, we are given two distinguished notations, *source* vertex s and *sink* vertex t. We assume that vertex s and vertex t are in the same connected component. In addition, there is a set of k *pebbles,* $Peb = \{p_1, p_2, \ldots, p_k\}$, with movable items that can move along the edges of the graph, that is, they can move from vertex u to vertex v, where $(u, v) \in E$. Finally, we are given a mapping function, $\phi : Peb \rightarrow V$, that maps each pebble to its *initial position*. The goal of the *movement problem* is to find a mapping function

$\delta : Peb \rightarrow V$ which assigns each pebble to its *final position*. When pebble p is mapped to vertex v by function ϕ or function δ, we say that pebble p *occupies* vertex v and vertex v is *occupied by* pebble p. An *occupied vertex* v is a vertex that is occupied by at least one pebble. Otherwise, we call vertex v an *unoccupied vertex*. The *occupied number* of vertex v is the number of pebbles that occupies vertex v. Regarding more definitions, we define a finite sequence of vertices $P = \{v_1, v_2, \ldots, v_j\}$, where $(v_i, v_{i+1}) \in E$; $1 \le i < j$ is a *path* in graph G. The path from vertex $v_1 \in V$ to vertex $v_j \in V$ is called the v_1-v_j path. The *length* of the v_1-v_j path P, denoted by $l_P(v_1, v_j)$, is the number of its composed edges. The movement cost of pebble p is the length of its traverse path P which starts from the initial position of p to the final position of it, that is, $l_P(\phi(p), \delta(p))$. The *maximum movement cost*, denoted by $c\_max(\delta)$, is the maximum movement cost made by the moved pebbles, that is, $c\_max(\delta) = max_{p \in Peb} \{ l_P(\phi(p), \delta(p))\}$. The *total movement cost*, denoted by $c\_total(\delta)$, is the sum of the movement costs made by all the moved pebbles, that is, $c\_total(\delta) = \sum_{p \in Peb} \{ l_P(\phi(p), \delta(p))\}$. The *s–t path movement problem* is a movement problem for which we want to find a mapping function δ such that there exists an *s–t* path in an induced subgraph on the final positions of the pebbles. The *PathMax* problem is an *s–t* path movement problem such that the maximum movement cost is minimized, that is, we want to minimize the $c\_max(\delta)$ cost. The *PathSum* problem is an *s–t* path movement problem such that the total movement cost is minimized, that is, we want to minimize the $c\_total(\delta)$ cost.

We note that function ϕ does not have to be an injective surjective function. This means that the problem definition allows more than one pebble to be initialized at each vertex and that there exist unoccupied vertices in an initialization. But for the sake of simplicity, in this paper, we assume that each vertex in graph G is occupied by at the most one pebble in an initialization. In addition, function δ also does not have to be an injective surjective function. To eliminate the non-injective property of function δ, we will prove a very useful lemma in the next section.

3 PathMax Problem on Tree

In this section, we will present a polynomial-time algorithm for solving the PathMax on a tree.

The solution opt_δ which minimizes $c\_max(\delta)$ is said to be *optimal*. Let the cost of this solution be equal to *OPT*, that is, $c\_max(opt_\delta) = OPT$. We will eliminate the non-injective property of function δ which states that we can move more than one pebble to the same vertex in a solution by proving Lemma 1.

Lemma 1: If there exists an optimal solution $opt_{\delta 1}$ that allows more than one pebble to occupy one vertex, then there exists another optimal solution $opt_{\delta 2}$ such that no more than one pebble occupies the same vertex, and $c\_max(opt_{\delta 2}) \le c\_max(opt_{\delta 1})$.

Proof: We will prove this lemma by a construction. We will construct the optimal solution $opt_{\delta 2}$ from the optimal solution $opt_{\delta 1}$, as follows. Let δ_1 and δ_2 be the mapping functions of the pebbles to their final positions in the optimal solution $opt_{\delta 1}$ and the optimal solution $opt_{\delta 2}$, respectively. For each vertex v where $\delta_1(p) = v$ and its occupied number is equal to one, we then assign $\delta_2(p) = v$, too. We see that in this part, $c\_max(opt_{\delta 2}) = c\_max(opt_{\delta 1})$. Consider vertex u which has an occupied number more than one. We assume that its occupied number is equal to two. Thus, a mapping function δ_1 maps two distinct pebbles to vertex u, that is, $\delta_1(p_1) = u = \delta_1(p_2)$ and $p_1 \ne p_2$. In the optimal solution $opt_{\delta 2}$, we can choose an arbitrary pebble, say, pebble p_1, to move to vertex u. Since pebble p_2 is also moved to vertex u in the optimal solution $opt_{\delta 1}$, there exists a path $P = \{\phi(p_2), v_1, v_2, \ldots, u\}$ in the graph. In the optimal solution $opt_{\delta 2}$, we will move pebble p_2 back to its initial position, that is, $\delta_2(p_2) = \phi(p_2)$. We can see that the movement cost of pebble p_2 in the optimal solution $opt_{\delta 2}$ did not increase. For every other vertex w which has an occupied number more than one, we can modify its final position as we do with vertex u. This is because in an initialization, we assume that each vertex is occupied by at the most one pebble. Thus, this method terminates in the final. In addition, we can conclude that there still exists an *s–t* path on an induced subgraph of occupied vertices in the optimal solution $opt_{\delta 2}$. Moreover, its cost does not increase, that is, $c\_max(opt_{\delta 2}) \le c\_max(opt_{\delta 1})$.

From Lemma 1, we see that we need to move exactly one pebble to each unoccupied vertex in the *s–t* path. This is the reason why we can use the matching for our proposed algorithm. For more details of our proposed algorithm, let us define more definitions. A *bipartite graph* $B = (X + Y, F)$ is a graph that is composed of a set of vertices $U = \{X \cup Y\}$ and a set of edges F where X and Y are disjoint sets, and if $(u, v) \in F$, then $u \in X$, $v \in Y$. The *matching* M is a subset of edges such that each vertex is incident with at most one edge in M. If one edge in a matching is incident to vertex v, then we call vertex v is *matched*. Otherwise, vertex v is called *unmatched*. The matching M' is a *maximal matching* if no other edges can be added to M', and if the added M' is no longer matching. The *maximum matching* M^* is a maximal matching with the maximum size. For a bipartite graph $B = (X + Y, F)$, an *X-saturating matching* is a matching in which all the vertices $v \in X$ are matched.

We reduce the PathMax problem to a problem of finding an X-saturating matching in the bipartite graph $B = (X + Y, F)$. For a given input tree $T = (V, E)$, we find an *s–t* path on it. From the definition of the tree T,

we see that there exists exactly one path. Recall that *OPT* is the cost of the optimal solution opt_δ, that is, $c\_max(opt_\delta) = OPT$. Initially, we guess an *OPT* value. We can do this because we know that this value ranges between 0 and *n*. Thus, we can perform the binary search to find them quickly. We construct a bipartite graph $B = (X + Y, F)$, where *X* is a set of unoccupied vertices in the *s*–*t* path on an input tree; *Y* is a set of the initial positions of all the pebbles, that is, $Y = \{v \in V | \phi(p) = v \text{ for all } p \in Peb\}$; and there exists an edge between two vertices if and only if there is a path *P* between these two vertices on tree *T* with length at the most *OPT*, that is, $(u, v) \in F$: $u \in X$, $v \in Y$ if and only if $l_P(u, v) \leq OPT$.

From the above-mentioned bipartite graph *B* construction and the fact that the path from any two vertices in a tree is unique, we have an observation, as follows.

Observation 2: *The bipartite graph B cannot have multiple edges*.

For the correctness of the algorithm, we will prove the following theorem.

Theorem 3: *A PathMax has an optimal solution* opt_δ *where* $c\_max(opt_\delta) = OPT$ *if and only if the bipartite graph* $B = (X + Y, F)$ *has an X-saturating matching M.*

Proof: Going forward, we will assume that the PathMax problem has an optimal solution opt_δ where $c\_max(opt_\delta) = OPT$ and we will prove that the bipartite graph *B* has an *X*-saturating matching *M*. Since the PathMax has the optimal solution opt_δ, each unoccupied vertex in an *s*–*t* path on a tree *T* has at least one pebble moved to it with a movement cost by at most *OPT*. Moreover, these pebbles are all distinct. For a subset $W \subseteq X$, we define a neighbor set of *W*, denoted by *N*(*W*), which is a set of vertices in set *Y* that is incident with a vertex in set *W*. From the bipartite graph *B* construction, we see that each vertex $x_i \in X$ has at least one degree. Thus, for every subset $W \subseteq X$, we can see that $|W| \leq |N(W)|$. We can conclude that the bipartite graph *B* has an *X*-saturating matching *M* by using Hall's theorem.

Now, we will assume that the bipartite graph *B* has an *X*-saturating matching *M* and we will prove that the PathMax problem has an optimal solution opt_δ where $c\_max(opt_\delta) = OPT$. Since graph *B* has an *X*-saturating matching, each vertex $x_i \in X$ has vertex $y_i \in Y$, where $(x_i, y_i) \in M$. We will construct an optimal solution opt_δ by moving pebble *p*, where $\phi(p) = y_i$, to vertex x_i, that is, assign $\delta(p) = x_i$. Because $(x_i, y_i) \in F$, $l_P(x_i, y_i) \leq OPT$. Moreover, since *M* is an *X*-saturating matching, all vertices in set *X* will have assigned pebbles. We can conclude that the PathMax problem has an optimal solution opt_δ where $c\_max(opt_\delta) = OPT$.

Next, we will prove the running time of our proposed algorithm.

Theorem 4: *A PathMax on a tree can be solved in* $O(k^2 \sqrt{n})$ *time.*

Proof: The running time of our proposed algorithm is dominated by the *X*-saturating matching algorithm. The best running time for computing the maximum matching in a bipartite graph with flow technique is the Hopcroft–Karp algorithm which runs in $O(m\sqrt{n})$ time for a graph with *n* vertices and *m* edges [13]. We know that the total number of pebbles, which is *k*, is at least the length of an *s*–*t* path on the tree, that is, $k \geq l_P(s, t)$. From the bipartite graph $B = (X + Y, F)$ construction, we see that $|X| \leq k = |Y|$. Thus, there is a maximum of k^2 edges on the graph *B*. This means that our proposed algorithm can solve a PathMax on tree in $O(k^2 \sqrt{n})$ time, as claimed.

4 PathSum Problem on Tree

We begin this section by an example of a PathSum problem on a tree.

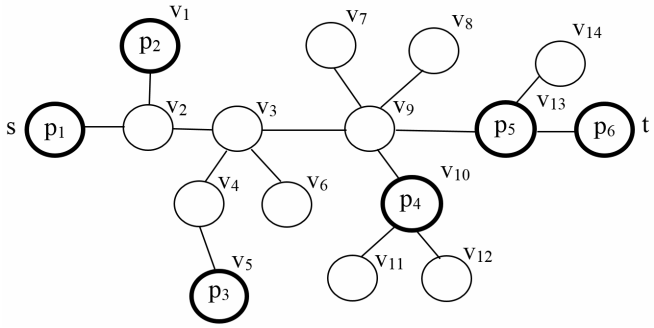

Figure 1. An example of a PathSum problem on a tree

From Figure 1, in an initialization, the occupied vertices are designated by the circles with bold lines, the vertices *s*, v_1, v_5, v_{10}, v_{13} and *t*. They occupied by the pebbles which label inside them. These vertices are also the initial positions of the pebbles p_1, p_2, p_3, p_4, p_5 and p_6 too, that is, $\phi(p_1) = s$, $\phi(p_2) = v_1$, $\phi(p_3) = v_5$, $\phi(p_4) = v_{10}$, $\phi(p_5) = v_{13}$ and $\phi(p_6) = t$, respectively. A possible optimal solution of this problem is assigning the pebbles p_1, p_2, p_3, p_4, p_5 and p_6 to the final positions *s*, v_2, v_3, v_9, v_{13} and *t*, that is, $\delta(p_1) = s$, $\delta(p_2) = v_2$, $\delta(p_3) = v_3$, $\delta(p_4) = v_9$, $\delta(p_5) = v_{13}$ and $\delta(p_6) = t$, respectively. The movement costs of the pebbles p_1, p_2, p_3, p_4, p_5 and p_6 are 0, 1, 2, 1, 0 and 0. Thus a total movement cost, $c\_total(\delta)$, is 0 + 1 + 2 + 1 + 0 + 0 = 4. We also note that a maximum movement cost, $c\_max(\delta)$, is max{0, 1, 2, 1, 0, 0} = 2.

Next, we will show that a function δ for a PathSum problem is an injective function by proving Lemma 5.

Lemma 5: *In any optimal solution* opt_δ *of the PathSum problem, there is exactly one pebble mapped to each vertex in the s-t path.*

Proof: We will prove this lemma by a contradiction. Suppose, for the sake of contradiction, that there exists an optimal solution opt_δ that allows two pebbles p_1 and p_2 to be mapped to a vertex *v* where *v* is a vertex on an *s*-*t* path. Consider another solution $opt_{\delta'}$ that moves

pebble p_1 to its initial position. We see that $c\_total(opt_{\delta'}) = c\_total(opt_\delta) - l_P(\phi(p_1), \delta(p_1)) < c\_total(opt_\delta)$, and that there still exists an s–t path in the induced subgraph in the final positions of the pebbles. Thus, this contradicts the assumption that opt_δ is the optimal solution.

To solve a PathSum problem on a tree, we will construct a bipartite graph $B = (X + Y, F)$ with weight function W: $F \rightarrow N$, where N is a natural number, as follows. Given a tree $T = (V, E)$, we construct a bipartite graph $B = (X + Y, F)$, where X is a set of unoccupied vertices in an s–t path on tree T; and Y is a set of the initial positions of all the pebbles, that is, $Y = \{v \in V \mid \phi(p) = v \text{ for all } p \in Peb\}$; there exists an edge between two vertices $x \in X$ and $y \in Y$ if and only if there is a path P between vertex x and vertex y on tree T and the weight of the edge $(x, y) \in F$ is equal to the length of the path P from vertex x to vertex y on tree T, that is, $W((x, y)) = l_P(x, y)$. To solve a PathSum on a tree, we find the minimum weighted matching saturated X on the bipartite graph B. In particular, we want the matching M that saturated X, with the sum of the weights of all the edges in the matching M minimized, that is, minimized $\sum_{(u, v)\in M}\{W((u, v))\}$.

For the correctness of the algorithm, we will use an analogous reason as we used in the proof of Theorem 3. Thus, we can obviously conclude Theorem 6.

Theorem 6: *A PathSum has an optimal solution opt_δ with cost $c\_total(opt_\delta)$ if and only if a bipartite graph $B = (X + Y, F)$ has an X-saturating matching M with minimum weight equal to $c\_total(opt_\delta)$.*

Next, we will prove the running time of our proposed algorithm.

Theorem 7: *A PathSum on a tree can be solved in $O(n^2 \log n + nk^2)$ time.*

Proof: The proof is from the fact that the minimum weighted matching in a bipartite graph can be solved in $O(n^2 \log n + nm)$ time by the Hungarian algorithm with the Dijkstra algorithm and Fibonacci heap [14] and an analogous reason as we used in the proof of Theorem 4. Recall that there is at the most k^2 edges on a bipartite graph B. Thus, our proposed algorithm can solve a PathSum on a tree in $O(n^2 \log n + nm) = O(n^2 \log n + nk^2)$ time, as claimed.

5 PathMax and PathSum Problems on Unicyclic Graph

The example of a PathMax and a PathSum problem on a uniclyclic graph are shown in Figure 2.

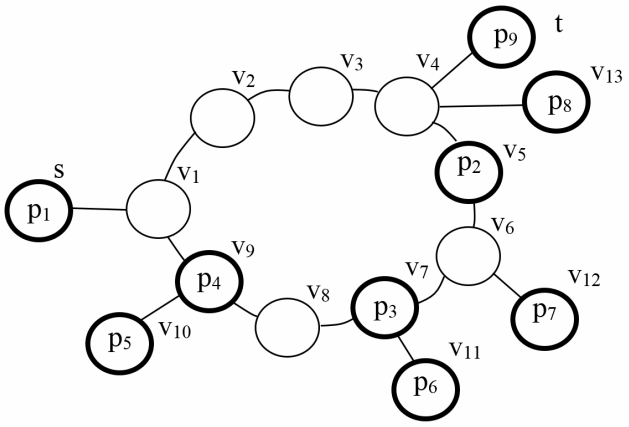

Figure 2. The examples of PathMax and PathSum problems on a unicyclic graph

From Figure 2, a possible optimal solution of this example is assigning the pebbles p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 and p_9 to the vertices s, v_5, v_8, v_1, v_9, v_7, v_6, v_4 and t, respectively. The movement costs of these pebbles are 0, 0, 1, 1, 1, 1, 1, 1, 0. Thus, a maximum movement cost, $c\_max(\delta)$, of a PathMax problem is 1 and a total cost, $c\_total(\delta)$, is 0 + 0 + 1 + 1 + 1 + 1 + 1 + 1 + 0 = 6. We note that another solution that move pebble p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 and p_9 to the vertices s, v_5, v_7, v_9, v_1, v_8, v_6, v_4 and t give a total movement cost 6 but give a maximum movement cost 2. Thus, this solution is an optimal solution to a PathSum problem but it is not an optimal solution of a PathMax problem in Figure 2.

Next, we will present an algorithm for the PathMax and the PathSum problems on a unicyclic graph by state the next lemma.

Lemma 8: *The maximum possible number of simple paths from vertex s to vertex t on a unicyclic graph G is two.*

A proof detail of this lemma is omitted because it follows from a definition of a unicyclic graph.

From Lemma 8, we propose an algorithm for a PathMax (PathSum) problem on a unicyclic graph, as follows. Find the two s–t paths P_1 and P_2 in the input unicyclic graph G. Then, run a tree algorithm which is presented in section 3 (section 4) on path P_1 and path P_2. Finally, the best of them, that is, the X-saturating matching (with minimum weight), is chosen as the solution of the PathMax (PathSum) problem on the unicyclic graph G.

Because we run the algorithm for a PathMax or a PathSum on a tree two times, we can conclude about the algorithm running time on a unicyclic graph as follows.

Theorem 9: *There is an $O(k^2 \sqrt{n})$ algorithm and an $O(n^2 \log n + nk^2)$ algorithm for solving a PathMax and a PathSum on a unicyclic graph, respectively.*

Proof: Since the two s-t paths P_1 and P_2 are the trees. From Theorem 4, we see that running a PathMax algorithm on the s-t paths P_1 and P_2 give $O(k^2 \sqrt{n})$ and $O(k^2 \sqrt{n})$ time. Thus, a total running time is $O(k^2 \sqrt{n} + k^2 \sqrt{n}) = O(k^2 \sqrt{n})$, as claimed. In the case of a PathSum

algorithm. From Theorem 7, we see that running a PathSum algorithm on the *s*-*t* paths P_1 and P_2 provide the same bounds, that is, $O(n^2 \log n + nk^2)$ time. So, a total running time is $O(n^2 \log n + nk^2 + n^2 \log n + nk^2) = O(n^2 \log n + nk^2)$, as claimed.

6 Conclusion

In this paper, we considered a PathMax problem and a PathSum problem on a tree and a unicyclic graph. *PathMax* problem and *PathSum* problem are the path movement problems whose maximum movements made by pebbles and total movements made by pebbles that we wanted to minimize, respectively. When we consider general graphs, the researchers in [13] proved that both the problems are NP-hard. In this paper, we show that PathMax problems and PathSum problems can be solved in $O(k^2\sqrt{n})$ time and $O(n^2 \log n + nk^2)$ time when the input graph is a tree. For a unicyclic graph, we proved that there are at most two simple *s*–*t* paths on the graph. Thus, we can use the algorithm for a tree as a subroutine when we want to solve PathMax problems and PathSum problems on a unicyclic graph. Since we call the subroutine two times, one for each simple path, the running time values of PathMax algorithms and PathSum algorithms for a unicyclic graph are the same as those of their algorithms on a tree, which are $O(k^2\sqrt{n})$ and $O(n^2 \log n + nk^2)$.

Acknowledgments

The authors would like to thank Assistant Professor Dr. Jittat Fakcharoenphol for the most valuable initial concept to solve this problem. The second author also a member of the Theory of Computation Group, Chiang Mai University and would like to thank this group for facility supports. Furthermore, the authors would like to thank the unknown reviewers for the insightful suggestions and supports.

References

[1] N. Xie, Y. Liang, J. Chen, Position Optimisation for Multiple Mobile Relays by Utilising One-bit Feedback Information, *IET Communications*, Vol. 12, No. 18, pp. 2266-2273, November, 2018.

[2] T. G. Nguyen, C. So-In, N. G. Nguyen, Barrier Coverage Deployment Algorithms for Mobile Sensor Networks, *Journal of Internet Technology*, Vol. 18, No. 7, pp. 1689-1699, December, 2017.

[3] J. Jia, X. Wu, J. Chen, X. Wang, An Autonomous Redeployment Algorithm for Line Barrier Coverage of Mobile Sensor Networks, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 16, No. 1, pp. 58-69, January, 2014.

[4] F. Wang, Z. Wang, Z. Yang, S. Chen, Contact Duration Aware Cache Refreshing for Mobile Opportunistic Networks, *IET Networks*, Vol. 5, No. 4, pp. 93-103, July, 2016.

[5] U. Baroudi, G. Sallam, M. Al-Shaboti, M. Younis, Self-deployed Wireless Actor Networks with Maximal Task Satisfaction, *IET Communications*, Vol. 11, No. 18, pp. 2713-2720, December, 2017.

[6] Y. Yu, L. Li, K. Liu, Y. Deng, X. Su, Broadcasting Algorithm Based on Successful Broadcasting Ratio and Energy Balance of Nodes in Mobile Ad Hoc Networks, *International Journal of Internet Protocol Technology*, Vol. 11, No. 1, pp. 1-11, January, 2018.

[7] E. D. Demaine, M. Hajiaghayi, H. Mahini, A. S. Sayedi-roshkhar, S. Oveisharan, M. Zadimoghaddam, Minimizing Movement, *ACM Transaction on Algorithms*, Vol. 5, No. 3, pp. 30:1-30:30, July, 2009.

[8] P. Berman, E. D. Demaine, M. Zadimoghaddam, O(1)-Approximations for Maximum Movement Problems, *Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and of the 15th International Workshop on Randomization and Computatio*n, Princeton, NJ, 2011, pp. 62-74.

[9] E. D. Demaine, M. Hajiaghayi, D. Marx, Minimizing Movement: Fixed-Parameter Tractability, *ACM Transactions on Algorithms*, Vol. 11, No. 2, pp. 14: 1-14: 29, November, 2014.

[10] N. Anri, M. Fazli, M. Ghodsi, M. Safari, Euclidean Movement Minimization, *Journal of Combinatorial Optimization*, Vol. 32, No. 2, pp. 354-367, February, 2015.

[11] W. Jindaluang, J. Chawachat, V. Chouvatut, J. Fakcharoenphol, S. Kantabutra, An Improved Approximation Algorithm for the s-t Path Movement Problem, *Chiang Mai Journal of Science*, Vol. 44, No. 1, pp. 279-286, January, 2017.

[12] D. Bilo, L. Guala, S. Leucci, G. Proietti, Exact and Approximate Algorithms for Movement Problems on (Special Classes of) Graphs, *Theoretical Computer Science*, Vol. 652, pp. 86-101, November, 2016.

[13] J. E. Hopcroft, R. M. Karp, An $n^{5/2}$ Algorithms for General Graph Matching Problems, *SIAM Journal on Computing*, Vol. 2, No. 4, pp. 225- 231, December, 1973.

[14] M. L. Fredman, R. E. Tarjan, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, *Journal of the ACM*, Vol. 34, No. 3, pp. 596-615, July, 1987.

Biographies

Varin Chouvatut graduated with B. Eng. (Honours) and M. Eng. in Computer Engineering and got her Ph.D. in Electrical and Computer Engineering from King Mongkut's University of Technology Thonburi since 2011. Recently, she is an assistant professor at Chiang Mai University. Her research interests include computer vision, image processing, computer graphics, and data science.

Wattana Jindaluang received a doctoral degree in Computer Engineering from Faculty of Engineering, Kasetsart University, Thailand. Presently, she is a lecturer in the Department of Computer Science, Faculty of Science, Chiang Mai University, Thailand. Her research interests are about Design and Analysis of Algorithms and Graph Algorithms and their applications.