Computational Complexity of Robot Arm Simulation Problems

Feng Tianfeng[1,a)] Yoshio Okamoto[2] Yota Otachi[3]
Takashi Horiyama[4] Toshiki Saitoh[5] Takeaki Uno[6]
Ryuhei Uehara[1,b)]

Abstract: For a given path P and a graph G with edge lengths, we consider the weighed Eulerian path problem that asks if there's an Eulerian path of G spanned by P with edge length constraints. We first show that this problem is strongly NP-hard even if edge lengths are quite restricted. Then we consider two different variants of this problem. In the first variant, we allow the edges in P to be elastic to fit the vertices of P to ones of G, our goal is to minimize the elastic ratio. When G is a path, this problem can be solved in polynomial time by dynamic programming. In the other variant, we allow P to cover an edge of G twice or more. We first show that it is weakly NP-hard even if G is an edge. We thus assume that each edge of G is covered by P exactly twice, and on one hand, we show this problem is still strongly NP-hard even in quite restricted cases. On the other hand, the problem is polynomial time solvable when the target graph G is a star and its edge lengths are of k different values.

Keywords: Linkage, robot arm, spanning problem

1. Introduction

A *robot arm* is a type of programmable mechanical arm, which can be modeled by a *linkage*. A linkage is a collection of fixed-length 1D segments joined at their endpoints forming a path. (See [2] for further details.) Namely, a linkage is a path $P = (v_0, v_1, \ldots, v_n)$ with length function $\ell : E \to R$, where v_i is an endpoint, and $e_i = \{v_{i-1}, v_i\}$ is an edge in $E = \{\{v_i, v_{i+1}\} \mid 0 \leq i < n\}$, and its length is given by $\ell(e_i)$. Now we consider the following situation. You are given a general target mechanism which is modeled by a graph $G = (V', E')$, and a robot arm modeled by a linkage $P = (V, E)$ as above with length function $\ell : E \cup E' \to R$. Our mission is *simulating* the target graph G by the given linkage P. The joints in P are programmable, and each joint (or vertex) of G should be simulated by a joint of P, but we can also put some joints of P on an internal point of an edge of G because they can be fixed. Therefore, our mission can be formalized to find the following mapping ϕ from P to G:

- Each edge of G should be mapped by a subpath of P.
- Each vertex of G should be mapped from some vertices of P.

The decision problem asks if there exists a mapping ϕ from P to G. That is, it asks if there is an Eulerian path of G spanned by P such that (1) when P visits a vertex in G, a vertex of P should be put on it, and (2) some vertices in P can be put on internal points of edges of G. When all edges have the same length, it is easy to solve that in linear time since the problem is the ordinary Eulerian path problem. In the context of formal language, there are some variants of the Eulerian path problem with some constraints (see [3] for a comprehensive survey). However, as far as the authors know, the *robot arm simulation problem*, our variant of the Eulerian path problem has not been investigated, while it is quite a natural situation.

The first interesting result is that this problem is strongly NP-hard even if edge lengths are quite re-

[1] Japan Advanced Institute of Science and Technology
[2] University of Electro-Communications
[3] Kumamoto University
[4] Saitama University
[5] Kyushu Institute of Technology
[6] National Institute of Informatics
[a)] ftflluy@jaist.ac.jp
[b)] uehara@jaist.ac.jp

stricted. Precisely, it is strongly NP-hard even if P and G consist of edges of length only 1 or 2 (Theorem 1). We remind that if they consist of unit length edges, the problem is linear time solvable. We thus consider two different variants of this problem.

The first variant is an optimization problem. In this variant, we consider a linkage is *elastic*, that is, the length of one line segment is not fixed and can be changed little a bit. This situation is natural in the context of the robot arm simulation. Formally, we allow the edges in P to be elastic to fit the vertices of P to ones of G. Our goal is to minimize the stretch/shrink ratio of each edge of P. We show that when G is a path, this can be solved in polynomial time by dynamic programming.

In the second variant, we allow P to cover an edge of G twice or more. In this case, we do not allow P to be elastic, or its ratio is fixed to 1. We first show that it is weakly NP-hard even if G is an edge. In fact, this problem is similar to the ruler folding problem (see, e.g., [2]). Therefore, we introduce another restriction that each edge of G is covered by P exactly twice. We first mention that this problem is quite easy when each edge has unit length. The answer is yes if and only if G is connected and P contains a certain number of edges. When G is connected, P can traverse every edge twice in the way of depth first search. This idea brought us a natural restricition that asks if we can cover G by P by traversing edges of G exactly twice. That is, even if G has no Eulerian path and hence P cannot simulate G properly, we can find the feasible way to simulate G by P in this way. From the practical viewpoint, it seems to be reasonable when we simulate a mechanism by a robot arm. From the viewpoint of graph theory, it is natural to consider the case that G is a tree. When G is a tree, the problem is in a simple form; P simulates G by traversing each edge twice in the unique spanning tree of G, or G itself. However, this problem is still strongly NP-hard even in quite restricted cases; (1) G is a star, and P consists of edges of only two different lengths, and (2) G is a spider, and all edges are of two different lengths. On the other hand, the problem is polynomial time solvable when G is a star and its edge lengths are of k different values.

2. Preliminaries

In this paper, we only consider a simple undirected graph $G = (V, E)$. A *path* $P = (v_0, v_1, \ldots, v_n)$ consists of $n + 1$ vertices with n edges joining v_i and v_{i+1} for each $i = 0, \ldots, n - 1$. The vertices v_0 and v_n of the path are called *endpoints*. Let $K_{n,m}$ denote a *complete bipartite graph* $G = (X, Y, E)$ such that $|X| = n$, $|Y| = m$, and every pair of a vertex in X and a vertex in Y is joined by an edge. A graph $G = (V, E)$ is a *tree* if it is connected and acyclic. Here a graph G is a *star* if and only if it is a complete bipartite graph $K_{1,n-1}$, and a graph G is a *spider* if and only if G is a tree that has only one vertex of degree greater than 2. In a star or a spider, the unique vertex of degree greater than 3 is called *center*.

Let $G = (V', E')$ and $P = (V, E)$ be a graph and a path $(v_0, v_1, \ldots, v_n)$. Let $\ell : E' \cup E \to R$ be an edge-length function of them. Then we say the linkage P can *simulate* the mechanism G if each edge in G is spanned by one subpath of P. More precisely, we can formalize the notion of simulation by a mapping ϕ that maps each vertex V to a *point* in G as follows. For any edge $e = \{u, v\} \in E'$, we consider e is a line segment (u, v) of length $\ell(e)$. Then the *intermediate point* p of distance $t\ell(e)$ from u is denoted by $p = tv + (1 - t)u$, where $0 < t < 1$. We note that the endpoints of an edge e are not considered intermediate points of e. Now we first define a set of *points* in G by V' and all intermediate points on edges of E'. Then we define a mapping ϕ from V to points of G as follows. To make it clear, we first divide V into two subsets V_e and V_i such that each vertex in V_e is mapped to a vertex in V', and each vertex in V_i is mapped to an intermediate point of G. In our problem, we assume that $\phi(v_0)$ and $\phi(v_n)$ should be in V_e. That is, P should start and end at a vertex in G. The mapping ϕ from V_e to V' is defined as follows; (1) for every $v' \in V'$, at least one vertex $v \in V_e$ with $\phi(v) = v'$. (2) for each edge $e' = \{v', u'\} \in E'$, there is a pair of vertices v_i and v_j in V_e such that (2a) between v_i and v_j, there is no other vertex v_k is in V_e, and (2b) $\ell(e') = \sum_{k=i+1}^{j} \ell(e_k)$. Each vertex in V_i is mapped to some intermediate point in G. Intuitively, when P simulate G, the corresponding joint of the robot arm is fixed. The mapping ϕ from V_i to the set of intermediate points in G is defined as follows. Let v_i and v_j any consecutive vertices in V_e, that is, there are no other vertex $v_k \in V_e$ with $i < k < j$, or $v_k \in V_i$ for all $i < k < j$. Then the mapping ϕ maps each $v_k \in V_i$ to an intermediate point of $\{\phi(v_i), \phi(v_j)\}$ in this ordering with length constraint. That is, for each $i < k < j$, let $\phi(v_k) = t\phi(v_j) + (1 - t)\phi(v_i)$. Then we always have $0 < t < 1$ and $\sum_{h=i}^{k-1} \sum_{\ell}(\{v_h, v_{h+1}\}) = t\ell(\{\phi(v_i), \phi(v_j)\})$. Then we say that the linkage P can simulate the mechanizm G if there is a mapping

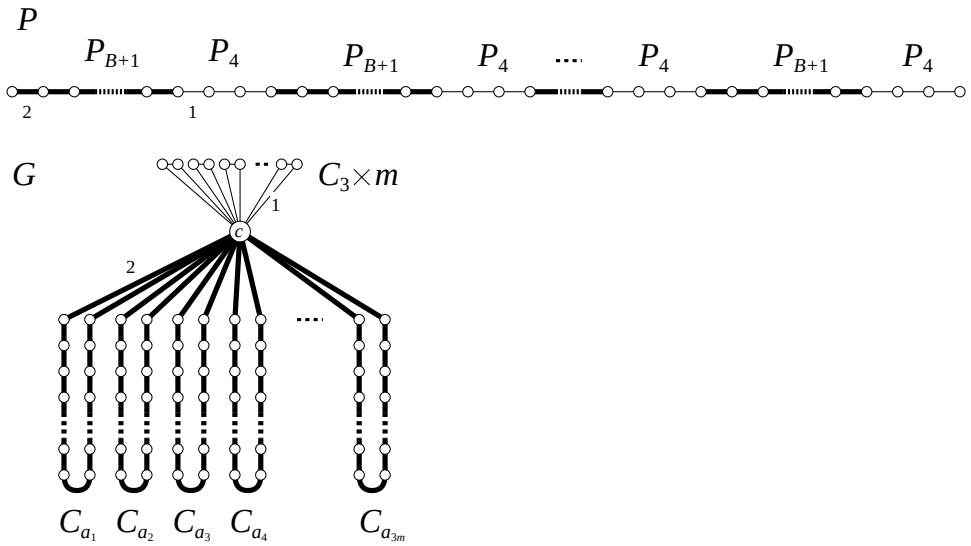

Fig. 1 Construction of P and G; bold lines are of length 2, and thin lines are of length 1. Each P_{i+1} consists of i edges and each C_i consists of i edges.

ϕ satisfying above conditions.

In this paper, we will often use the following problem to show hardness of our problems:

3-Partition

: An integer B and a multiset A of $3m$ integers $A = a_1, a_2, \ldots a_{3m}$ with $B/4 < a_i < B/2$.

Output: Determine if A can be partitioned into m multisets $S_1, S_2, \ldots, S_m$ such that $\sum_{a_j \in S_i} a_j = B$ for every i.

Without loss of generality, we can assume that $\sum_{a_i \in A} a_i = mB$, and $|S_i| = 3$. It is well known that the 3-Partition problem is strongly NP-complete [1].

3. Weighted Eulerian path problem

Now we show the main theorem in this section.

Theorem 1 Let P, G, ℓ be a path, an undirected graph, and a length function, respectively. Then the weighted Eulerian path problem is strongly NP-hard even if $\ell(e)$ is either 1 or 2 for any e in P and G.

Proof. It is easy to see that the problem is in NP. Therefore we show the hardness. We reduce the 3-Partition problem to the weighted Eulerian path problem.

Let P_{B+1} be a path that consists of B consecutive edges of length 2, and P_4 be a path that consists of 3 consecutive edges of length 1. Then the path P is obtained by joining m subpaths P_{B+1} and m subpaths P_4 alternatively, that is, P is constructed by joining P_{B+1}, P_4, P_{B+1} P_4, …, P_4, P_{B+1}, and P_4. The graph G is constructed as follows. For each i with $1 \leq i \leq 3m$, we construct a cycle C_{a_i} of a_i edges of length 2. We also construct m cycles C_3 of 3 edges of length 1. Then these $4m$ cycles share a special vertex c in common. That is, G is a cactus consists of $4m$ cycles, and all vertices have degree 2 except the common vertex c that has degree $8m$. The construction is illustrated in **Fig. 1**.

It is easy to see that it is polynomial time reduction. Thus, we show that A has a solution if and only if P can cover G along an Euler tour in G with satisfying the condition of the linkage simulation. We first observe that no edge of length 2 in P_{B+1} in P can cover a cycle C_3 in G. Therefore, when P cover G, every C_3 of G has to be covered by P_4 in P. Thus, each endpoint of P_4 should be on c in G, and no edge in P_{B+1} can cover edges in C_3. Hence, each subpath P_{B+1} in P covers exactly B edges in the set of cycles C_{a_i} that consists of edges of length 2. Since $B/4 < a_i < B/2$ for each i, each subpath P_{B+1} covers exactly three cycles C_{a_i}, C_{a_j} and C_{a_k} for some i, j, k with $a_i + a_j + a_k = B$. Clearly, each cover for a subpath P_{B+1} gives a subset of A, and collection of these subsets gives us a solution of the 3-Partition problem and vice versa. ∎

4. Elastic linkage problem

In this section, we consider the following problem:

Elastic linkage problem from path to path

Input: Two paths $G = (V', E')$ and $P = (V, E)$ with length function ℓ.

Output: a mapping ϕ with minimum elastic (or stretch/shrink) ratio.

In this problem, we allow all edges in P to be elastic to simulates the path G by the path P. The *ratio* of an edge e is defined by l'/l, where l is the length

of the edge e in P, l' is the length of e in P after being stretched or shrank on G. The *elastic ratio* of an edge e is defined by $\max\{r, 1/r\}$, where r is the ratio of the edge e in P. Since each vertex in G should be mapped from only one vertex in P, the first and last vertices in G should be mapped from the first and last vertices in P, otherwise the elastic ratio will be infinity. The elastic ratio of a mapping is the maximum among elastic ratios of all edges in P.

We show a polynomial time algorithm for this problem based on a dynamic programming.

First, we show a technical lemma when G is just an edge. In this case, the optimal value is achieved when all ratios are even.

Lemma 2 Assume that G consists of an edge $e = (u_0, u_1)$. When $P = (V, E)$ is a path, the minimum elastic ratio is achieved when each ratio of $e \in E$ takes the same value.

Proof. Assume the length of the edge in G is L, $E = \{e_1, e_2, \ldots, e_n\}$, each length of e_i is l_i, each ratio of e_i is r_i on a mapping ϕ. On the ϕ, we have $r_1 l_1 + r_2 l_2 + r_3 l_3 + \cdots + r_n l_n = L$.

Assume the maximum among r_i for each $1 \leq i \leq n$ is r_k, the minimum among r_i for each $1 \leq i \leq n$ is r_h. So it is obvious that $r_k \geq L/(l_1 + l_2 + \cdots + l_n)$, $1/r_h \geq (l_1 + l_2 + \cdots + l_n)/L$.

According to the definition, the elastic ratio er of this mapping is the maximum among r_i and its reciprocal for each $1 \leq i \leq n$. That is, er equals the maximum between r_k and $1/r_h$.

When $r_1 = r_2 = ... = r_n$, r_k and $1/r_h$ can take the minimum. That means the minimum elastic ratio can be achieved if and only if each ratio of $e \in E$ takes the same value.

∎

Now we turn to the main theorem.

Theorem 3 We can solve the elastic linkage problem from path to path in $O(n^3)$ time.

Proof. We assume path P=$(v_1, v_2, \ldots, v_n)$, the length of each edge $\{v_i, v_{i+1}\}$ is l_i, path G=$(u_1, u_2, \ldots, u_{n'})$, the length of each edge $\{u_j, u_{j+1}\}$ is w_j, and $n \geq n' \geq 2$.

We define two functions as follows for $i > i' \geq j$:

$$dist(v_{i'}, v_i) = l_{i'} + l_{i'+1} + \cdots + l_{i-1}$$

$$Ser(v_{i'}, v_i, w_j) = \max\left\{\frac{w_j}{dist(v_{i'}, v_i)}, \frac{dist(v_{i'}, v_i)}{w_j}\right\}$$

That is, $dist(v_{i'}, v_i)$ is the length of the path $(v'_i, \ldots, v_i)$, and $Ser(v_{i'}, v_i, w_j)$ is the minimum elastic ratio of all edges in the subpath $P' = (v_{i'}, v_{i'+1}, \ldots, v_i)$ of P that covers the edge $\{u_j, u_{j+1}\}$.

We first precompute this function as a table which will be referred in our polynomial time algorithm. The computation of $Ser[(v_{i'}, v_i), w_j]$ can be done as follows: (1) for each $(v_{i'}, v_i)$ with $i' < i$, compute $dist(v_{i'}, v_i)$ and fill in the table $dist[v_{i'}, v_i]$, (2) for each $j = 1, 2, \ldots, n'-1$, compute $Ser(v_{i'}, v_i, w_j)$ and fill in the table $Ser[(v_{i'}, v_i), w_j]$.

In (1), each $dist(v_{i'}, v_i)$ can be computed in a constant time by using $dist(v_{i'}, v_i) = dist(v_{i'}, v_{i-1}) + \ell(e_i)$ when we compute the values of this table in the order of $(i - i') = 1, 2, 3, \ldots$. On the other hand, in (2), each $Ser(v_{i'}, v_i, w_j)$ can be computed in a constant time. Therefore, the precomputation can be done in $O(n^3)$ time in total. To solve the elastic linkage problem efficiently, we define two more functions $ER(v_i, u_j)$ and $M(v_i, u_j)$ as follows. $ER(v_i, u_j)$ is the minimum elastic ratio of the mappings from the subpath $P' = (v_1, v_2, \ldots, v_i)$ of P to the subpath $G' = (u_1, u_2, \ldots, u_j)$ of G. Then we have the following:

$$ER(v_i, u_j) = \begin{cases} Ser(v_1, v_i, w_1) & \text{when } j = 2 \\ \min\{\max\{ER(v_k, u_j - 1), Ser(v_k, v_i, w_{j-1})\} \quad k = j-1, j, \ldots, i-1\} & \text{when } j > 2 \end{cases}$$

Our goal is to obtain the mapping from P to G with elastic ratio $ER(v_n, u_{n'})$. $M(v_i, u_j)$ is a sequence of j vertices of path P that represents the mapping with minimum elastic ratio from the subpath P' to the subpath G'. The first and last vertices in $M(v_i, u_j)$ is v_1 and v_i. Then we have the following:

$$M(v_i, u_j) = \begin{cases} (v_1, v_i) & \text{when } j = 2 \\ (M(v_\tau, u_{j-1}), v_i) & \text{when } j > 2, \end{cases}$$

where τ is determined by the following equation:
$ER(v_i, u_j) = \max\{ER(v_\tau, u_{j-1}), Ser(v_\tau, v_i, w_{j-1})\}$

Our goal is to obtain $M(v_n, u_{n'})$. The $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be obtained simultaneously by dynamic programming technique. In the tables of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$, $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ are easy to get if the values in the $(n'-1)$-st row are available. The table $ER(v_n, u_{n'})$ is filled from $j = 2$, that is, for $v_i = v_2, v_3, \ldots, v_n$, $ER(v_i, u_2) = Ser(v_1, v_i, w_1)$ have already precomputed, and accordingly, the first row of table $M(v_n, u_{n'})$ is $M(v_i, u_2) = (v_1, v_i)$. After filling in

the first row of the tables, it is easy to get the values in the second row, the third row, up to the $(n' - 2)$-nd row and finally get $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$. Each element of the table $ER(v_n, u_{n'})$ can be computed in $O(n)$ time, each element of the table $M(v_n, u_{n'})$ can be computed in a constant time. That is, the computation of $ER(v_n, u_{n'})$ and $M(v_n, u_{n'})$ can be done in $O(n^3)$ time, the precomputation can be done in $O(n^3)$ time too. Therefore, the algorism runs in $O(n^3)$ time, that means the elastic linkage problem can be solved in polynomial time. ∎

5. Covering problem of a tree by a path

In this section, we focus on the traverse problem of G by P. In this variant, we allow to P to cover an edge of G twice or more, but each length is fixed (or, in other words, the elastic ratio is fixed to 1).

However, in general case, this problem is similar to the following ruler folding problem:

Ruler Folding: Given a polygonal chain with links of integer length $\ell_0, \ldots, \ell_{n-1}$ and an integer L, can the chain be folded flat so that its total folded length is L?

The details of this problem and related results can be found in [2]. For the general cover problem of G by P, we have the following theorem:

Theorem 4 The general cover problem of G by P is NP-complete even if G is an edge.

Proof. We can reduce the ruler folding problem to our problem by just letting G be an edge of length L. ∎

We note that the ruler folding problem is weakly NP-complete, and we have a simple pseudo-polynomial time algorithm that runs in $O(nL)$ time as follows;

Input: Set of integers $S = \ell_0, \ldots, \ell_{n-1}$ and an integer L
Output: Determine if there is $S' \subseteq S$ with $\sum_{i \in S'} \ell_i = L$
begin
  Initialize array $a[0], \ldots, a[L]$ by 0;
  Set $a[0] = 1$;
  foreach $i = 0, \ldots, n-1$ **do**
    foreach $j = 0, \ldots, L$ **do**
      if $a[j] == 1$ *and* $j + \ell_i \leq L$ **then**
        $a[j + \ell_i] = 1$;
    end
  end
  if $a[L] == 1$ **then** output "Yes";
  else output "No";
end

However, when we have no restriction on the traverse of an edge in G, it is quite hard to solve. Therefore we here introduce a reasonable restriction on covers of G by P. We restrict the target graph G is a tree T, and P can cover each edge in T at most (or possibly exactly) twice. This idea comes from the real simulation of a general mechanism G by a robot arm P. In this case, a reasonable general way consists of two steps; first, we make a spanning tree T of G, and next, we cover T by P in a depth first search manner. Then P covers each edge in T exactly twice. That is, hereafter, we consider the following *tree traversal problem*:

Input: A path $P = (V, E)$ that forms a path $(v_0, v_1, \ldots, v_n)$ with length function $\ell : E \to R$, and a tree $T = (V', E')$ with length function $\ell : E' \to R$. (We do not distinguish the length function ℓ for P and T.)

Output: A traversal of T by P such that each edge in T is spanned by exactly two subpaths of P, or "No" if it does not exist.

We first observe that it is linear time solvable when each edge has the unit length just by depth first search. Therefore, it is an interesting question that asks the computational complexity when ℓ maps few distinct values, especially, ℓ maps two distinct values, and G is a simple tree, e.g., a star or a spider.

We give three hardness results about the traversal problem of T even if T and P are quite restricted.

Theorem 5 The traversal problem of a tree T by a path P is strongly NP-complete in each of the following cases: (1) T is a star $K_{1,n-1}$, and P consists of edges of two different lengths. (2) T is a spider and all edges in G and P are of length p and q, where (2a) p and q are any two positive integers that are relatively prime, or (2b) $p = 1$ and $q = 2$.

Proof. It is clear that each of the problems is in NP, we show hardness. We will give polynomial time reductions from the 3-Partition to our problems.

(1) T is a star $K_{1,4m+1}$. Among $4m + 1$ edges, the length of $m + 1$ edges is B, and the other $3m$ edges have length a_i for each $i = 1, 2, \ldots, 3m$ (Fig. 2). The construction of P is as follows. Let P' be a path that consists of $2B$ edges of length 1, and P'' be a path that consists of 2 edges of length B. Then the path P is obtained by joining $m + 1$ subpaths P'' and m subpaths P' alternatively, that is, P is constructed by joining P'', P', P'', P' P'', ..., P', P'' as shown in Fig. 2. The construction is done in polynomial time. Thus we show that the 3-Partition problem has a solution if and only if the constructed cover problem has a solution. We first observe that P'' cannot cover any short edge of length a_i in T. Therefore, each P'' should cover each edge of length

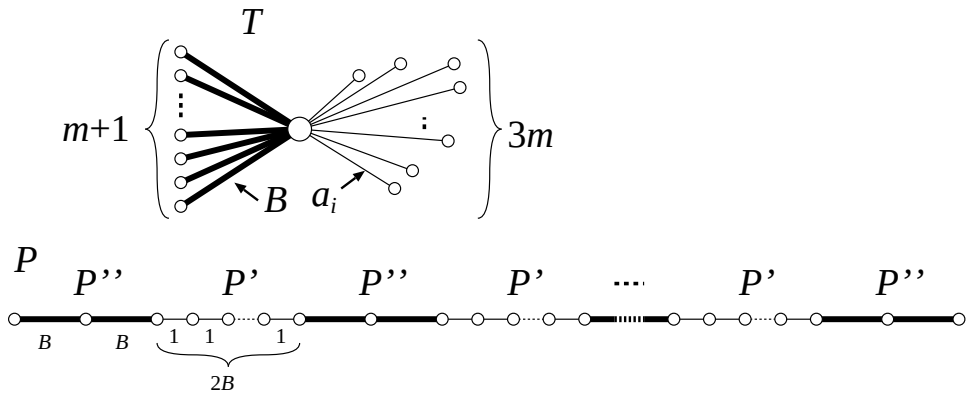

Fig. 2 Reduction to $K_{1,4m+1}$ and a path P.

B in T twice. Hence all of the endpoints of P''s (and hence P') are on the central vertex of T. Therefore, if P can cover T properly, it is easy to see that each P' should cover three edges of length a_i, a_j, and a_k with $a_i + a_j + a_k = B$ exactly twice. This concludes the proof of (1).

(2a) This reduction is similar to (1). Let P' be a path that consists of $2B$ edges of length p, and P'' be a path that consists of 2 edges of length q. Then the path P is obtained by joining $m+1$ subpaths P'' and m subpaths P' alternatively. On the other hand, the spider T is obtained by sharing the central vertex of $4m+1$ sub paths (Fig. 3). Among $4m+1$ subpaths, $m+1$ paths are just edges of length q. The other $3m$ subpaths are of a_i edges for each $1 \leq i \leq 3m$, and each edge has length p. Since p and q are relatively prime, P'' cannot cover each of edges of length p. Therefore, their endpoints (and the endpoints of P') share the central vertex of T. Thus each P' gives us the solution of the 3-Partition as in (1), which completes the proof of (2a).

(2b) The reduction itself is the same as (2a) but $p = 1$ and $q = 2$. In this case, we observe that no edge of length 1 can be covered by any edge of length 2 in P''. Therefore, each edge of P'' of length 2 should cover the edges of T of length 2. Thus each P' gives us the solution of the 3-Partition as in (2a), which completes the proof of (2b). ∎

In Theorem 5, we show that the cover problem of a tree by a path is NP-hard even if we strictly restrict ourselves. Now we turn to show a polynomial time algorithm for the case that we furthermore restrict.

Theorem 6 Let T be a star $K_{1,n'}$ and the number of distinct lengths of its edges is k. Let P be any path of length n. Without loss of generality, we suppose $2n' \leq n$ Then the cover problem of T by P can be solved in $O(n^{k+1})$ time and $O(n^k)$ space. That is, it is polynomial time solvable when k is a constant.

Proof. We suppose that each edge of T has of length in $L = \{\ell_1, \ell_2, \ldots, \ell_k\}$, and T contains L_i edges of length ℓ_i for each i. For a vertex v_i in P and length ℓ_j in L, we define a function $\mathrm{pre}(v_i, \ell_j)$ as follows;

$$\mathrm{pre}(v_i, \ell_j) = \begin{cases} v_k & \text{there is a vertex } v_k \text{ with } k < i \text{ on } P \text{ such that } \ell(e_k) + \ell(e_{k+1}) + \cdots + \ell(e_i) = \ell_j \\ \phi & \text{otherwise} \end{cases}$$

We first precompute this function as a table which will be referred in our polynomial time algorithm. We refer this table as $\mathrm{pre}[v_i, \ell_j]$ which uses $O(nk)$ space. The computation of pre[] can be done as follows; (0) initialize pre[] by ϕ in $O(nk)$ time, (1) sort L in $O(k \log k)$ time, and (2) for each $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, k$, the vertex v_i fills the table $\mathrm{pre}[v_{i'}, \ell_j] = v_i$. In (2), the vertex v_i can fill $\mathrm{pre}[v_{i'}, \ell_j] = v_i$ in $O(n+k)$ time. Therefore, the precomputing takes $O(n(n+k) + k \log k)$ time in $O(nk)$ space.

Now we turn to the computation for the cover problem. To do that, we define a predicate $F(d_1, d_2, \ldots, d_k, v_i)$ which is defined as follows: When there is a cover of a subtree T' of T that consists of d_1 edges of length ℓ_1, d_2 edges of length ℓ_2, ..., and d_k edges of length ℓ_k by the subpath $P' = (v_0, v_1, \ldots, v_i)$ when v_0 and v_i are put on the center of T, $F(d_1, d_2, \ldots, d_k, v_i)$ is *true*, and *false* otherwise. (For notational convenience, we define

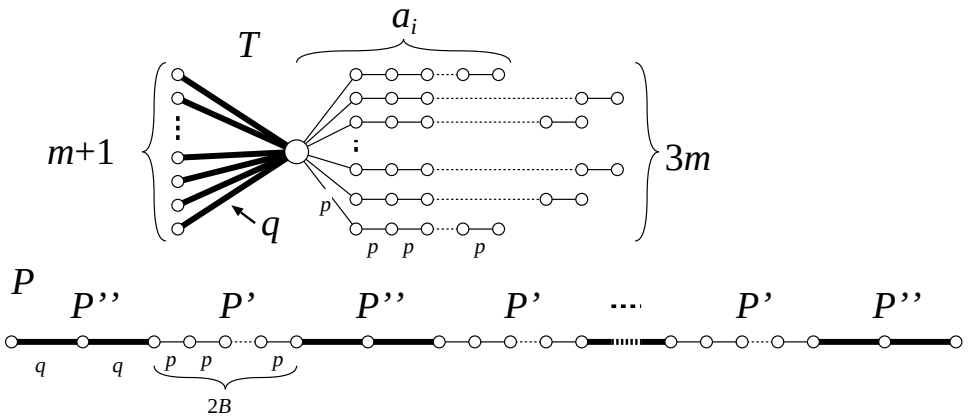

Fig. 3 Reduction to spider of two different lengths.

that $F(d_1, d_2, \ldots, d_k, \phi)$ is always false.) Thus, our goal is to determine if $F(L_1, L_2, \ldots, L_k, v_n)$ is true or false. The predicate $F(d_1, d_2, \ldots, d_k, v_i)$ is determined by the following recursively;

$$F(d_1, d_2, \ldots, d_k, v_i) = \bigvee_{1 \le j \le k} ((\mathrm{pre}(v_i, \ell_j) \neq \phi) \wedge F(d_1, \ldots, d_j - 2, \ldots, d_k, \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j)))$$

That is, for the vertex v_i, we have to have two vertices $v_{i'} = \mathrm{pre}(v_i, \ell_j)$ and $v_{i''} = \mathrm{pre}(\mathrm{pre}(v_i, \ell_j), \ell_j)$ such that $\ell(e_{i'}) + \ell(e_{i'+1}) + \cdots + \ell(e_i) = \ell_j$ and $\ell(e_{i''}) + \ell(e_{i''+1}) + \cdots + \ell(e_{i'}) = \ell_j$ for some j with $1 \le j \le k$. The correctness of this recursion is trivial.

The predicate $F(L_1, L_2, \ldots, L_k, v_n)$ is computed by a dynamic programming technique. That is, the table $F[d_1, d_2, \ldots, d_k, v_i]$ is filled from $d_1 = 0, d_2 = 0, \ldots, d_k = 0$ for the center vertex c, which is true. Then, we increment in the bottom up manner; that is, we increment as $(d_1, d_2, \ldots, d_k) = (0, 0, \ldots, 0, 1)$, $(0, 0, \ldots, 1, 0)$, ..., $(0, 1, \ldots, 0, 0)$, $(1, 0, \ldots, 0, 0)$, $(0, 0, \ldots, 0, 2)$, $(0, 0, \ldots, 1, 1)$, ..., $(0, 1, \ldots, 0, 1)$, $(1, 0, \ldots, 0, 1)$, and so on. The number of combinations of $(d_1, d_2, \ldots, d_k)$ is $L_1 \cdot L_2 \cdot \cdots \cdot L_k \le n'^k = O(n^k)$, and the computation of $F[d_1, d_2, \ldots, d_k, v_i]$ for the $(d_1, d_2, \ldots, d_k)$ can be done in linear time. Therefore, the algorithm runs in $O(n^{k+1})$ time $O(n^k)$ space.

∎

References

[1] Demaine, E. D. and O'Rourke, J.: *Geometric folding algorithms*, Cambridge university press Cambridge (2007).
[2] Garey, M. R. and Johnson, D. S.: Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences (1979).
[3] Kupferman, O. and Vardi, G.: Eulerian Paths with Regular Constraints, *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 58, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016).