Shortest Reconfiguration Sequence for Sliding Tokens on Spiders

Duc A. Hoang[1] **Amanj Khorramian**[2] **Ryuhei Uehara**[1]

[1]School of Information Science, JAIST, Japan
{hoanganhduc, uehara}@jaist.ac.jp
[2]University of Kurdistan, Sanandaj, Iran
khorramian@gmail.com

Suppose that two independent sets I and J of a graph with $|I| = |J|$ are given, and a token is placed on each vertex in I. The Sliding Token problem is to determine whether there exists a sequence of independent sets which transforms I into J so that each independent set in the sequence results from the previous one by sliding exactly one token along an edge in the graph. It is one of the representative reconfiguration problems that attract the attention from the viewpoint of theoretical computer science. For a yes-instance of a reconfiguration problem, finding a shortest reconfiguration sequence has a different aspect. In general, even if it is polynomial time solvable to decide whether two instances are reconfigured with each other, it can be NP-hard to find a shortest sequence between them. In this paper, we show that the problem for finding a shortest sequence between two independent sets is polynomial time solvable for spiders (i.e., trees having exactly one vertex of degree at least three).

1 Introduction

Recently, the *reconfiguration problems* attracted the attention from the viewpoint of theoretical computer science. The problem arises when we like to find a step-by-step transformation between two feasible solutions of a problem such that all intermediate results are also feasible and each step abides by a fixed reconfiguration rule, that is, an adjacency relation defined on feasible solutions of the original problem. The reconfiguration problems have been studied extensively for several well-known problems, including Independent Set, Satisfiability, Set Cover, Clique, Matching, and so on. (See the surveys [4, 7] for more details.) In general, the reconfiguration problems tend to be PSPACE-complete, and some polynomial time algorithms are shown in restricted cases. Finding a shortest sequence in the context of the reconfiguration problems is a new trend in theoretical computer science because it has a great potential to characterize the class NP from a different viewpoint from the classic ones.

One of the important NP-complete problems is the Independent Set problem. For this notion, the natural reconfiguration problem is called the Sliding Token problem introduced by Hearn and Demaine [3]. Suppose that we are given two independent sets I and J of a graph $G = (V, E)$ such that $|I| = |J|$, and imagine that a *token* (coin) is placed on each vertex in I. For convenience, sometimes we identify the token with the vertex it is placed on and simply say "a token in an independent set." Then, the Sliding Token problem is to determine whether there exists a sequence $S = \langle I_1, I_2, \ldots, I_\ell \rangle$ of independent sets of G such that

(a) $I_1 = I$, $I_\ell = J$, and $|I_i| = |I| = |J|$ for all i, $1 \leq i \leq \ell$; and

(b) for each i, $2 \leq i \leq \ell$, there is an edge xy in G such that $I_{i-1} \setminus I_i = \{x\}$ and $I_i \setminus I_{i-1} = \{y\}$.

That is, I_i can be obtained from I_{i-1} by sliding exactly one token on a vertex $x \in I_{i-1}$ to its adjacent vertex $y \in I_i$ along an edge $xy \in E(G)$. Such a sequence S, if exists, is called a TS-*sequence* in G between I and J.

For the Sliding Token problem, some polynomial time algorithms have been provided as follows: Linear time algorithms have been shown for cographs (also known as P_4-free graphs) and trees. Polynomial time algorithms are shown for bipartite permutation graphs, claw-free graphs, and cacti. On the other hand, PSPACE-completeness is also shown for graphs of bounded tree-width, planar graphs, and planar graphs with bounded bandwidth. (See [5] for more details.)

In this context, we aim to find a shortest sequence of the Sliding Token problem, which is called the Shortest Sliding Token problem, for these graph classes. There are two variants of this problem. One is the *decision variant*, that is, another integer ℓ is also given as a part of input, and we have to decide whether there exists a shortest sequence between I and J of length at most ℓ. The other one (*non-decision variant*) asks us to output all independent sets of the shortest sequence itself. The length ℓ is not necessarily polynomial in $|V(G)|$ in general. When ℓ is not polynomial, we may have that the decision variant is in P, while the non-decision one is not in P since it takes non polynomial time to the output sequence. We also note that even when G is a perfect graph and ℓ is polynomial in $|V(G)|$, the decision variant of Shortest Sliding Token is NP-complete (see [6, Theorem 5]).

From this viewpoint, the length of a token sliding is a key feature of the Shortest Sliding Token problem. If the length is not in polynomial in total, there exists at least one token that slides non polynomial times. That is, the token visits the same vertex many times in its slides. That is, some tokens make *detours* in the sequence (the notion of detour is important and precisely defined later). In order to concentrate on the detours of tokens, it is a natural constraint that the graph itself has no cycle, that is, the graph is a tree. The decision variant of the Sliding Token problem on a tree can be solved in linear time [1]. Polynomial-time algorithms for the Shortest Sliding Token problem were first investigated in [10]. Among them, they give a polynomial time algorithm for caterpillars, which form quite simple trees, and this is the first graph class that required detours to solve the Shortest Sliding Token problem. A caterpillar is a tree that consists of a "backbone" called a *spine* with many *pendants*, or leaves attached to the spine. Each pendant can be used to escape a token, however, the other tokens cannot pass through it. Therefore, the ordering of tokens on the spine is fixed. In this paper, we consider the Shortest Sliding Token problem on a spider, which is a tree with one central vertex of degree more than 2. On this graph, we can use each "leg" as a stack and exchange tokens using these stacks. Therefore, we have many ways to handle the tokens, and hence we need more analyses to find a shortest sequence. In this paper, we give $O(n^2)$ time algorithms for the Shortest Sliding Token problem on a spider, where n is the number of vertices. The algorithm is constructive, and the sequence itself can be output in $O(n^2)$ time. As mentioned in [10], the length of a sequence can be $\Omega(n^2)$, hence our algorithm is optimal for the length of the sequence. Due to space restriction, several statements are omitted. For more information, the readers are referred to the full version of this extended abstract [5].

Note Recently, it is announced that the Shortest Sliding Token problem on a tree can be solved in polynomial time by Sugimori [9]. His algorithm is based on a dynamic programming on a tree [8]: though it runs in polynomial time, it seems to have much larger degree comparing to our case-analysis based algorithm.

2 Preliminaries

For common use graph theoretic definitions, we refer the readers to the textbook [2]. Throughout this paper, we denote by $V(G)$ and $E(G)$ the vertex-set and edge-set of a graph G, respectively. We always use n for denoting $|V(G)|$. For a vertex $x \in V(G)$, we denote by $N_G(x)$ the set $\{y \in V(G) : xy \in E(G)\}$ of *neighbors* of x, and by $N_G[x]$ the set $N_G(x) \cup \{x\}$ of *closed neighbors* of x. In a similar manner, for an induced subgraph H of G, the set $N_G[H]$ is defined as $\bigcup_{x \in V(H)} N_G[x]$. The *degree* of x, denoted by $\deg_G(x)$, is the size of $N_G(x)$. For $x, y \in V(G)$, the *distance* $\mathsf{dist}_G(x, y)$ between x and y is simply the length (i.e., the number of edges) of a shortest xy-path in G.

For a tree T, we denote by P_{xy} the (unique) shortest xy-path in T, and by T_y^x the subtree of T induced by y and its descendants when regarding T as the tree rooted at x. A *spider graph* (or *starlike tree*) is indeed a tree having exactly one vertex (called its *body*) of degree at least 3. For a spider G with body v and a vertex $w \in N_G(v)$, the path G_w^v is called a *leg* of G. By definition, it is not hard to see that two different legs of G have no common vertex.

Let (G, I, J) be an instance of Shortest Sliding Token. A *target assignment* from I to J is simply a bijective mapping $f : I \to J$. A target assignment f is called *proper* if there exists a TS-sequence in G between I and J that moves the token on w to $f(w)$ for every $w \in I$. Given a target assignment $f : I \to J$ from I to J, one can also define the target assignment $f^{-1} : J \to I$ from J to I as follows: for every $x \in J$, $f^{-1}(x) = \{y \in I : f(y) = x\}$. Let $\mathcal{F}$ be the set of all target assignments from I to J. We define $M^*(G, I, J) = \min_{f \in \mathcal{F}} \sum_{w \in I} \mathsf{dist}_G(w, f(w))$.

Let $S = \langle I_1, I_2, \ldots, I_\ell \rangle$ be a TS-sequence between two independent sets $I = I_1$ and $J = I_\ell$ of a graph G. Indeed, one can describe S in term of token-slides as follows: $S = \langle x_1 \to y_1, x_2 \to y_2, \ldots, x_{\ell-1} \to y_{\ell-1} \rangle$, where x_i and y_i ($i \in \{1, 2, \ldots, \ell - 1\}$) satisfy $x_i y_i \in E(G)$, $I_i \setminus I_{i+1} = \{x_i\}$, and $I_{i+1} \setminus I_i = \{y_i\}$. The *reverse* of S (which reconfigures J to I), denoted by $\mathsf{rev}(S)$, is defined by $\mathsf{rev}(S) = \langle I_\ell, \ldots, I_2, I_1 \rangle$. One can also describe $\mathsf{rev}(S)$ in term of token-slides: $\mathsf{rev}(S) = \langle y_{\ell-1} \to x_{\ell-1}, \ldots, y_2 \to x_2, y_1 \to x_1 \rangle$. For an edge $e = xy \in E(G)$, we say that S *makes detour over* e if both $x \to y$ and $y \to x$ are members of S. The *number of detours* S *makes over* e, denoted by $D_G(S, e)$, is defined to be twice the minimum between the number of appearances of $x \to y$ and the number of appearances of $y \to x$. The *total number of detours* S *makes in* G, denoted by $D_G(S)$, is defined to be $\sum_{e \in E(G)} D_G(S, e)$. Let $\mathcal{S}$ be the set of all TS-sequences in G between two independent sets I, J. We define $D^*(G, I, J) = \min_{S \in \mathcal{S}} D_G(S)$.

3 Shortest Sliding Token for spiders

In this section, we show that Shortest Sliding Token for spiders can be solved in polynomial time. For an independent set I of a graph G, the token on $u \in I$ is (G, I)-*rigid* if for any I' with $I \overset{G}{\leftrightsquigarrow} I'$, $u \in I'$. We note that *rigid token* plays an important role in designing a linear-time algorithm for deciding whether there is a TS-sequence between two independent sets of a tree [1]. As a spider is also a tree, for an instance (G, I, J) of Shortest Sliding Token for spiders, we can assume without loss of generality that $I \overset{G}{\leftrightsquigarrow} J$ and there are no (G, I)-rigid and (G, J)-rigid tokens.

The content of this sections is organized as follows. In Section 3.1, we prove some useful observations for trees, which clearly also hold for spiders. Then, in Section 3.2, we claim that given an instance (G, I, J) of Shortest Sliding Token for spiders, one can construct a target assignment $f : I \to J$ that minimizes $\sum_{w \in I} \mathsf{dist}_G(w, f(w))$. Finally, in Section 3.3, we show how to use such a target assignment f for explicitly constructing a TS-sequence of shortest length between I and J.

3.1 Observations for trees

Lemma 1. *Let I, J be two independent sets of a tree T such that $I \overset{T}{\leftrightsquigarrow} J$. Then, for every* TS*-sequence S between I and J, $\mathsf{len}(S) \geq M^*(T, I, J) + D^*(T, I, J)$.*

3.2 Target assignment

In this section, we claim that for an instance (G, I, J) of Shortest Sliding Token for spiders, one can construct a target assignment $f : I \to J$ such that $M^*(G, I, J) = \sum_{w \in I} \mathsf{dist}_G(w, f(w))$ in polynomial time.

For convenience, we always assume that the given spider G has body v and $\deg_G(v)$ legs $L_1, \ldots, L_{\deg_G(v)}$. Moreover, we assume without loss of generality that these legs are labeled such that $|I \cap V(L_i)| - |J \cap V(L_i)| \leq |I \cap V(L_j)| - |J \cap V(L_j)|$ for $1 \leq i \leq j \leq \deg_G(v)$; otherwise, we simply re-label them. For each leg L_i ($i \in \{1, 2, \ldots, \deg_G(v)\}$), we define the corresponding independent sets I_{L_i} and J_{L_i} as follows: $I_{L_1} = (I \cap V(L_1)) \cup (I \cap \{v\})$; $J_{L_1} = (J \cap V(L_1)) \cup (J \cap \{v\})$; and for $i \in \{2, \ldots, d\}$, we define $I_{L_i} = I \cap V(L_i)$ and $J_{L_i} = J \cap V(L_i)$. In this way, we always have $v \in I_{L_1}$ (resp. $v \in J_{L_1}$) if $v \in I$ (resp. $v \in J$).

Under the above assumptions, we claim that Algorithm 1 indeed constructs our desired target assignment. More formally,

Algorithm 1 Find a target assignment between two independent sets I, J of a spider G such that $M^*(G, I, J) = \sum_{w \in I} \mathsf{dist}_G(w, f(w))$.

Input: Two independent sets I, J of a spider G with body v.
Output: A target assignment $f : I \to J$ such that $M^*(G, I, J) = \sum_{w \in I} \mathsf{dist}_G(w, f(w))$.

for $i = 1$ to $\deg_G(v)$ **do**
  while $I_{L_i} \neq \emptyset$ and $J_{L_i} \neq \emptyset$ **do**
    Let $x \in I_{L_i}$ be such that $\mathsf{dist}_G(x, v) = \max_{x' \in I_{L_i}} \mathsf{dist}_G(x', v)$.
    Let $y \in J_{L_i}$ be such that $\mathsf{dist}_G(y, v) = \max_{y' \in J_{L_i}} \mathsf{dist}_G(y', v)$.
    $f(x) \leftarrow y$; $I_{L_i} \leftarrow I_{L_i} \setminus \{x\}$; $J_{L_i} \leftarrow J_{L_i} \setminus \{y\}$.
  end while
end for
while $\bigcup_{i=1}^{\deg_G(v)} I_{L_i} \neq \emptyset$ and $\bigcup_{i=1}^{\deg_G(v)} J_{L_i} \neq \emptyset$ **do** ▷ From this point, for any leg L, either $I_L = \emptyset$ or $J_L = \emptyset$.
  Take a leg L_i such that there exists $x \in I_{L_i}$ satisfying $\mathsf{dist}_G(x, v) = \min_{x' \in \bigcup_{i=1}^{\deg_G(v)} I_{L_i}} \mathsf{dist}_G(x', v)$.
  Take a leg L_j such that there exists $y \in J_{L_j}$ satisfying $\mathsf{dist}_G(y, v) = \max_{y' \in \bigcup_{i=1}^{\deg_G(v)} J_{L_i}} \mathsf{dist}_G(y', v)$.
  $f(x) \leftarrow y$; $I_{L_i} \leftarrow I_{L_i} \setminus \{x\}$; $J_{L_j} \leftarrow J_{L_j} \setminus \{y\}$.
end while
return f.

Lemma 2. *Let (G, I, J) be an instance of* Shortest Sliding Token *where I, J are independent sets of a spider G with body v. Let $f : I \to J$ be a target assignment produced from Algorithm 1. Then,*

(i) Algorithm 1 constructs f in $O(|I|)$ time; and

(ii) for an arbitrary target assignment $g : I \to J$, $\sum_{w \in I} \mathsf{dist}_G(w, g(w)) \geq \sum_{w \in I} \mathsf{dist}_G(w, f(w))$. In other words, f satisfies $M^*(G, I, J) = \sum_{w \in I} \mathsf{dist}_G(w, f(w))$.

Note that from Algorithm 1, one can naturally define a total ordering $<$ on vertices of I as follows: for $x, y \in I$, set $x < y$ if x is assigned before y.

3.3 Construction of a shortest TS-sequence for spiders

Algorithm 2 Construct a total ordering $\prec$ of vertices in I.

Input: The natural ordering $<$ on vertices of I derived from Algorithm 1.
Output: A total ordering $\prec$ of vertices in I.

while there exists w such that $K(w,<) \neq \emptyset$ **do**
  Let w be the smallest element of I with respect to $<$ such that $K(w,<) \neq \emptyset$.
  Let L be the leg of G such that $w \in I_L$.
  for $x \in K(w,<)$ **do** ▷ Originally, $x > w$
    Set $x \prec w$.
  end for
  if $\left|K^2(w,<)\right| \geq 2$ **then** ▷ $K^2(w,<)$ in reverse order
    for $x, y \in K^2(w,<)$ **do**
      if $x < y$ **then**
        Set $x \succ y$.
      end if
    end for
  end if
  if $\min\{\left|K^1(w,<)\right|, \left|K^2(w,<)\right|\} \geq 1$ **then**
    for $x \in K^1(w,<)$ and $y \in K^2(w,<)$ **do** ▷ Originally, $x > y$
      Set $x \prec y$.
    end for
  end if
  if $\min\{\left|K^1(w,<)\right|, \left|I^1_L \setminus K^1(w,<)\right|\} \geq 1$ **then**
    for $x \in I^1_L \setminus K^1(w,<)$ and $y \in K^1(w,<)$ **do** ▷ Originally, $x < y$
      Set $x \prec y$.
    end for
  end if
  for $x, y \in I$ whose ordering has not been defined in $\prec$ **do**
    if $x < y$ **then**
      Set $x \prec y$.
    end if
  end for
  for $x, y \in I$ **do** ▷ Re-define $<$ to use in the next iteration
    if $x \prec y$ **then**
      Set $x < y$.
    end if
  end for
end while
return The total ordering $\prec$ of vertices in I.

In this section, we describe a polynomial-time algorithm for solving Shortest Sliding Token for spiders, provided that a target assignment f produced from Algorithm 1 is given.

Let (G, I, J) be an instance of Shortest Sliding Token for spiders. Assume that the body v of the given spider G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. In this case, we claim that one can construct a TS-sequence S in G of length $M^*(G, I, J)$ between I and J. (Recall that we assumed $I \overset{G}{\leftrightsquigarrow} J$ and no (G, I)-rigid and (G, J)-rigid tokens exist.)

We first define some useful notations. Let $f : I \to J$ be a target assignment produced from Algorithm 1. For each leg L of G, we define $I^1_L = \{w \in I_L : f(w) \notin J_L\}$ and $I^2_L =$

$\{w \in I_L : f(w) \in J_L\}$. Given a total ordering $\lhd$ on vertices of I and a vertex $x \in I$, we define $K(x, \lhd) = N_G[P_{xf(x)}] \cap \{y \in I : x \lhd y\}$. If $x \in I_L$ for some leg L of G, we define $K^1(x, \lhd) = K(x, \lhd) \cap I_L^1$ and $K^2(x, \lhd) = K(x, \lhd) \cap I_L^2$. By definition, it is not hard to see that I_L^1 and I_L^2 (resp. $K^1(x, \lhd)$ and $K^2(x, \lhd)$) form a partition of I_L (resp. $K(x, \lhd)$).

Starting from the natural total ordering $<$ produced from Algorithm 1, one can construct a total ordering $\prec$ on vertices of I as described in Algorithm 2. Indeed, we claim that under the above assumptions, Algorithm 2 correctly produces a total ordering $\prec$ on vertices of I such that $K(w, \prec) = \emptyset$ for every $w \in I$. More formally,

Lemma 3. *Let (G, I, J) be an instance of* Shortest Sliding Token *for spiders, where the body v of G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. Let $f : I \to J$ be a target assignment produced from Algorithm 1, and $<$ be the corresponding natural total ordering on vertices of I. Assume that $I = \{w_1, w_2, \ldots, w_{|I|}\}$ is such that $w_1 < w_2 < \cdots < w_{|I|}$. Let w_i be the smallest element in I (with respect to the ordering $<$) such that $K(w_i, <) \neq \emptyset$, and L be the leg of G such that $w_i \in I_L$. Then,*

- *(i) $K(w_i, <) \subseteq I_L$. Additionally, $w_i \in I_L^2$.*
- *(ii) Let $\prec$ be the total ordering of vertices in I defined as in lines 2–28 of Algorithm 2, where the corresponding vertex w is replaced by w_i. Then,*
  - *(ii-1) If $x \in K(w_i, <)$ then $x > w_i$ and $x \prec w_i$.*
  - *(ii-2) If $x, y \in K^1(w_i, <)$ then $x < y$ if and only if $x \prec y$.*
  - *(ii-3) If $x, y \in K^2(w_i, <)$ then $x < y$ if and only if $x \succ y$.*
  - *(ii-4) If $x \in K^1(w_i, <)$ and $y \in K^2(w_i, <)$ then $x > y$ and $x \prec y$.*
  - *(ii-5) If $x \in I_L^1 \setminus K^1(w_i, <)$ and $y \in K^1(w_i, <)$ then $w_i < x < y$ and $x \prec y \prec w_i$.*
  - *(ii-6) If $x \in K(w_i, <) \cup I_L^1 \cup \{w_i\}$ and $y \in I \setminus (K(w_i, <) \cup I_L^1 \cup \{w_i\})$ then $x < y$ if and only if $x \prec y$.*
  - *(ii-7) If $x, y \in I \setminus (K(w_i, <) \cup I_L^1 \cup \{w_i\})$ then $x < y$ if and only if $x \prec y$.*
- *(iii) Let $\prec$ be the total ordering of vertices in I described in (ii). Then, $K(w_i, \prec) = \emptyset$. Moreover, if w_j is the smallest element in I (with respect to the ordering $\prec$) such that $K(w_j, \prec) \neq \emptyset$, then $K(w_j, \prec) = K(w_j, <)$.*

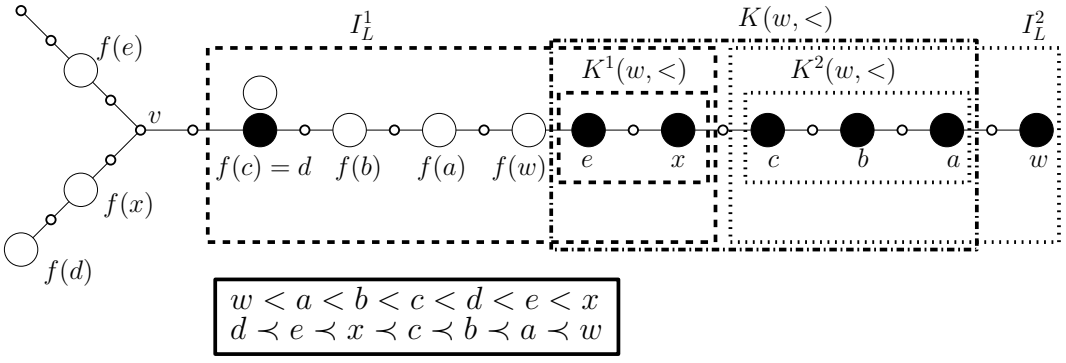

Figure 1: An example of the orderings $<$ and $\prec$ described in lines 2–28 of Algorithm 2. Tokens in I (resp. J) are of black (resp. white) color.

We informally explain why Lemma 3 guarantees that the total ordering $\prec$ on vertices of I produced from Algorithm 2 satisfies $K(w, \prec) = \emptyset$ for every $w \in I$. Intuitively, Lemma 3(i) and

(ii) say that if $w_i \in I_L$ is the "chosen" vertex in line 2 of Algorithm 2 for some leg L of G, then only a subset $K(w_i, <) \cup I_L^1 \cup \{w_i\}$ of I_L contains "candidates" for "re-ordering". Lemma 3(iii) guarantees that after "re-ordering", w_i will never be chosen again[1], and the next iteration of the main **while** loop can be initiated. As Algorithm 2 can "choose" at most $|I|$ vertices, and each iteration involving the "re-ordering" of at most $O(|I|)$ vertices, it will finally stop and produce the desired ordering in $O(|I|^2)$ time.

Now, we are ready to prove the following lemma.

Lemma 4. *Let (G, I, J) be an instance of* Shortest Sliding Token *for spiders where the body v of G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. Assume that there exists a leg L of G with $|I_L| \neq |J_L|$. Then, in $O(n^2)$ time, one can construct a* TS*-sequence S between I and J such that $\mathsf{len}(S) = M^*(G, I, J)$.*

In Lemma 4, we assumed that there is some leg L of G with $|I_L| \neq |J_L|$. In the next lemma, we consider the case $|I_L| = |J_L|$ for every leg L of G (regardless of whether $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$).

Lemma 5. *Let (G, I, J) be an instance of* Shortest Sliding Token *for spiders. Let v be the body of G. Assume that $|I_L| = |J_L|$ for every leg L of G. Then, in $O(n^2)$ time, one can construct a* TS*-sequence S between I and J such that $\mathsf{len}(S) = M^*(G, I, J)$.*

Using Lemma 5, we can assume without loss of generality that for an instance (G, I, J) of Shortest Sliding Token for spiders, there must be some leg L of G such that $|I_L| \neq |J_L|$.

Next, we show that when the body v of the given spider G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$, under certain conditions, either a TS-sequence of length $M^*(G, I, J)$ exists, or $D^*(G, I, J) \geq 2$ and a TS-sequence of length $M^*(G, I, J) + 2$ exists.

Lemma 6. *Let (G, I, J) be an instance of* Shortest Sliding Token *for spiders where the body v of G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. Assume that $|I_L| \neq |J_L|$ for some leg L of G. Let $x \in I$ (resp. $y \in J$) be such that $I \cap N_G(v) = \{x\}$ (resp. $J \cap N_G(v) = \{y\}$), provided that $I \cap N_G(v) \neq \emptyset$ (resp. $J \cap N_G(v) \neq \emptyset$). Then,*

(i) If x and y both exist, and $x \in I_L$ and $y \in J_L$ for some leg L of G with $|I_L| = |J_L|$, then for every TS*-sequence S between I and J, $D_G(S) \geq 2$. Consequently, $D^*(G, I, J) \geq 2$. Moreover, one can construct in $O(n^2)$ time a* TS*-sequence between I and J of length $M^*(G, I, J) + 2$.*

(ii) Otherwise, one can construct in $O(n^2)$ time a TS*-sequence between I and J of length $M^*(G, I, J)$.*

For the rest of this section, we consider the case when the body v of the given spider G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \geq 2$. More precisely, we claim that

Lemma 7. *Let (G, I, J) be an instance of* Shortest Sliding Token *for spiders where the body v of G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \geq 2$. Assume that $|I_L| \neq |J_L|$ for some leg L of G. Then, in $O(n^2)$ time, one can construct a* TS*-sequence S between I and J of shortest length. Moreover, the value of $D_G(S)$ can be explicitly calculated.*

Before proving Lemma 7, we define an useful notation for calculating the number of detours. For an instance (T, I, J) of Shortest Sliding Token for trees, we define a directed *auxiliary graph* $A(T, I, J)$ as follows: $V(A(T, I, J)) = V(T)$; and $E(A(T, I, J)) = \{(x, y) : xy \in E(T)$ and $\left|I \cap T_y^x\right| \leq \left|J \cap T_y^x\right|\}$. By definition, the auxiliary graph $A(G, J, I)$ can be obtained from $A(G, I, J)$ by simply reversing the directions of its edges.

Combining Lemmas 1, 5, 6, and 7, we have

[1] $K(w_i, \prec) = \emptyset$ always holds, since none of the members of $K(w_i, <)$ will ever be larger than w_i in the new orderings $\prec$ produced in the next iterations.

Theorem 8. *Given an instance* (G, I, J) *of* Shortest Sliding Token *for spiders, one can construct a shortest* TS*-sequence between* I *and* J *in* $O(n^2)$ *time.*

4 Conclusion

In this paper, we have shown that one can indeed construct a TS-sequence of shortest length between two given independent sets of a spider graph (if exists). Along the way, we proved several interesting observations that remain true even when the input graph is a tree (Section 3.1). We conjecture that these observations along with the structure of the auxiliary graph defined in Section 3.3 will provide an useful framework for improving the polynomial-time algorithm for Shortest Sliding Token for trees [9].

Acknowledgment R. Uehara was partially supported by JSPS KAKENHI Grant Number JP17H06287 and 18H04091.

References

[1] Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. "Linear-time algorithm for sliding tokens on trees". In: *Theoretical Computer Science* 600 (2015), pp. 132–142. DOI: 10.1016/j.tcs.2015.07.037.

[2] Reinhard Diestel. *Graph Theory*. 4th. Vol. 173. Graduate Texts in Mathematics. Springer, 2010.

[3] Robert A. Hearn and Erik D. Demaine. "PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation". In: *Theoretical Computer Science* 343.1-2 (2005), pp. 72–96. DOI: 10.1016/j.tcs.2005.05.008.

[4] Jan van den Heuvel. "The Complexity of Change". In: *Surveys in Combinatorics*. Vol. 409. London Mathematical Society Lecture Note Series. Cambridge University Press, 2013, pp. 127–160. DOI: 10.1017/CBO9781139506748.005.

[5] Duc A. Hoang, Amanj Khorramian, and Ryuhei Uehara. "Shortest Reconfiguration Sequence for Sliding Tokens on Spider". In: *arXiv preprints* (2018). arXiv: 1806.08291.

[6] Marcin Kamiński, Paul Medvedev, and Martin Milanič. "Complexity of independent set reconfigurability problems". In: *Theoretical Computer Science* 439 (2012), pp. 9–15. DOI: 10.1016/j.tcs.2012.03.004.

[7] Naomi Nishimura. "Introduction to Reconfiguration". In: *Algorithms* 11.4 (2018). (article 52). DOI: 10.3390/a11040052.

[8] Ken Sugimori. Personal communications. May 2018.

[9] Ken Sugimori. "Shortest Reconfiguration of Sliding Tokens on a Tree". In: *AAAC 2018*. May 2018.

[10] Takeshi Yamada and Ryuhei Uehara. "Shortest reconfiguration of sliding tokens on a caterpillar". In: *Proceedings of WALCOM 2016*. Ed. by Mohammad Kaykobad and Rossella Petreschi. Vol. 9627. LNCS. Springer, 2016, pp. 236–248. DOI: 10.1007/978-3-319-30139-6_19.