

Shortest Reconfiguration Sequence for Sliding Tokens on Spiders

Duc A. Hoang^a, Amanj Khorramian^b, Ryuhei Uehara^{a,1}

^a*School of Information Science, JAIST,
1-1 Asahidai, Nomi, Ishikawa, 923-1292 Japan*

^b*Department of Electrical and Computer Engineering,
University of Kurdistan, Sanandaj, Iran*

Abstract

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).

Keywords: sliding token, shortest reconfiguration, independent set, spider tree, polynomial-time algorithm.

1. Introduction

Recently, the *reconfiguration problems* attracted the attention from the viewpoint of theoretical computer science. These problem arise 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 [11, 15, 16, 18], SATISFIABILITY [10, 17], SET COVER, CLIQUE, MATCHING [15], and so on.

A reconfiguration problem can be seen as a natural “puzzle” from the viewpoint of recreational mathematics. The *15-puzzle* is one of the most famous classic puzzles, that had the greatest impact on American and European societies (see [22] for its rich history). It is well known that the 15-puzzle has a parity, and one can solve the problem in linear time just by checking whether the parity of one placement coincides with the other or not. Moreover, the distance between any two reconfigurable placements is $O(n^3)$, that is, we can reconfigure from one to the other in $O(n^3)$ sliding pieces when the size of the board is $n \times n$. However, surprisingly, for these two reconfigurable placements, finding a shortest path is NP-complete in general [4, 20]. Namely, although we know that there is a path of length in $O(n^3)$, finding a shortest one is NP-complete. While every piece is a unit square in the 15-puzzle, we obtain the other famous classic puzzle when we allow to have rectangular pieces, which is called “Dad puzzle” and its variants can be found in the

Email addresses: hoanganhduc@jaist.ac.jp (Duc A. Hoang), khorramian@gmail.com (Amanj Khorramian), uehara@jaist.ac.jp (Ryuhei Uehara)

URL: <http://hoanganhduc.github.io/> (Duc A. Hoang), <http://www.jaist.ac.jp/~uehara/> (Ryuhei Uehara)

¹JSPS KAKENHI Grant Number JP17H06287 and 18H04091

whole world (e.g., it is called “hako-iri-musume” in Japanese). Gardner said that “these puzzles are very much in want of a theory” in 1964 [9], and Hearn and Demaine gave the theory after 40 years [11]; they are PSPACE-complete in general [12].

Summarizing up, these sliding block puzzles characterize representative computational complexity classes; the decision problem for unit squares can be solved in linear time just by checking parities, finding a shortest reconfiguration for the unit squares is NP-complete, and the decision problem becomes PSPACE-complete for rectangular pieces. That is, this simple reconfiguration problem gives us a new sight of these representative computational complexity classes.

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.

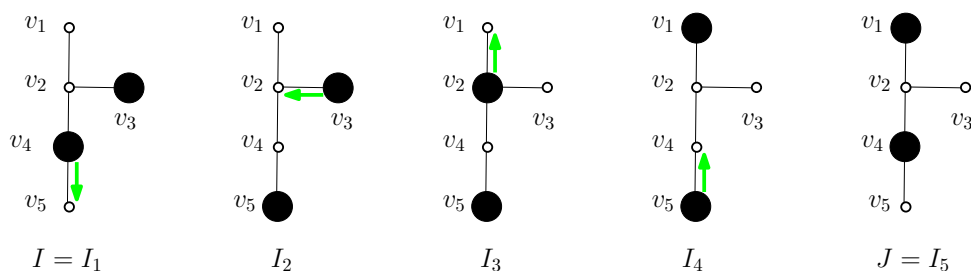


Figure 1: A sequence $\langle I_1, I_2, \dots, I_5 \rangle$ of independent sets of the same graph, where the vertices in independent sets are depicted by small black circles (tokens).

One of the important NP-complete problems is the INDEPENDENT SET problem. For this notion, a natural reconfiguration problem called SLIDING TOKEN was introduced by Hearn and Demaine [11]. (See [16] for an overview on different reconfiguration variants of INDEPENDENT SET.) 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 $S = \langle I_1, I_2, \dots, I_\ell \rangle$ 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 . We denote by a 3-tuple (G, I, J) an instance of SLIDING TOKEN problem. If a TS-sequence S in G between I and J exists, we say that I is *reconfigurable* to J (and vice versa), and write $I \overset{G}{\rightsquigarrow} J$. The sets I and J are the *initial* and *target* independent sets, respectively. For a TS-sequence S , the *length* $\text{len}(S)$ of S is defined as the number of independent sets in S minus one. In other words, $\text{len}(S)$ is the number of token-slides described in S . Figure 1 illustrates a TS-sequence of length 4 between two independent sets $I = I_1$ and $J = I_5$.

For the SLIDING TOKEN problem, linear-time algorithms have been shown for cographs (also known as P_4 -free graphs) [16] and trees [6]. Polynomial-time algorithms are shown for bipartite permutation graphs [8], claw-free graphs [3], cacti [13], and interval graphs [2]². On the other hand, PSPACE-completeness is also shown for graphs of bounded tree-width [19], planar graphs [11, 12], planar graphs with bounded bandwidth [25], and split graphs [1].

² We note that the algorithm for a block graph in [14] has a flaw, and hence it is not yet settled [21].

In this context, for a given YES-instance (G, I, J) of SLIDING TOKEN, we aim to find a shortest TS-sequence between I and J . Such a problem is called the SHORTEST SLIDING TOKEN problem. As seen for the 15-puzzle, the SHORTEST SLIDING TOKEN problem can be intractable even for these graph classes which the decision problem can be solved in polynomial time. Moreover, in the 15-puzzle, we already know that it has a solution of polynomial length for two configurations. However, in the SLIDING TOKEN problem, we have no upper bound of the length of a solution in general. To deal with this delicate issue, we have to distinguish two variants of this problem. In the *decision variant*, an integer ℓ is also given as a part of input, and we have to decide whether there exists a sequence between I and J of length at most ℓ . In the *non-decision variant*, we are asked to output a specific shortest TS-sequence. The length ℓ is not necessarily polynomial in $|V(G)|$ in general. When ℓ is super-polynomial, we may have that the decision variant is in P, while the non-decision one is not in P since it takes super-polynomial time to output the sequence. On the other hand, 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 [16, Theorem 5]). In short, in the decision variant, we focus on the *length* of a shortest TS-sequence, while in the non-decision variant, we focus on the *construction* of a shortest TS-sequence itself.

From this viewpoint, the length of a token sliding is a key feature of the SHORTEST SLIDING TOKEN problem. If the length is super-polynomial in total, there exists at least one token that slides super-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 general, it seems to be more difficult to analyze “detours of tokens” for graphs containing cycle(s). As a result, one may first consider the problem for trees. The SLIDING TOKEN problem on a tree can be solved in linear time [6]. Polynomial-time algorithms for the SHORTEST SLIDING TOKEN problem were first investigated in [26]. In [26], the authors gave polynomial-time algorithms for solving SHORTEST SLIDING TOKEN when the input graph is either a proper interval graph, a trivially perfect graph, or a caterpillar. We note that caterpillars 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 an $O(n^2)$ 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 [26], the number of required token-slides in a sequence can be $\Omega(n^2)$, hence our algorithm is optimal for the number of token-slides.

Note: Recently, it is announced that the SHORTEST SLIDING TOKEN problem on a tree can be solved in polynomial time by Sugimori [23]. His algorithm is based on a dynamic programming on a tree [24]: though it runs in polynomial time, it seems to have much larger degree comparing to our case-analysis based algorithm.

2. Preliminaries

For common graph theoretic definitions, we refer the readers to the textbook [7]. 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* $\text{dist}_G(x, y)$ 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 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. For example, the graph in Figure 1 is a spider with body $v = v_2$ and $\deg_G(v) = 3$ legs attached to v .

Let (G, I, J) be an instance of SHORTEST SLIDING TOKEN. A *target assignment* from I to J is simply a bijective mapping $f : I \rightarrow J$. 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 $f : I \rightarrow J$ from I to J , one can also define the target assignment $f^{-1} : J \rightarrow I$ 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 $M^*(G, I, J) = \min_{f \in \mathcal{F}} \sum_{w \in I} \text{dist}_G(w, f(w))$. Intuitively, observe that any TS-sequence between I and J in G (if exists) uses at least $M^*(G, I, J)$ token-slides.

Let $S = \langle I_1, I_2, \dots, I_\ell \rangle$ 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: $S = \langle x_1 \rightarrow y_1, x_2 \rightarrow y_2, \dots, x_{\ell-1} \rightarrow y_{\ell-1} \rangle$, where x_i and y_i ($i \in \{1, 2, \dots, \ell-1\}$) 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 $\text{rev}(S)$, is defined by $\text{rev}(S) = \langle I_\ell, \dots, I_2, I_1 \rangle$. One can also describe $\text{rev}(S)$ in term of token-slides: $\text{rev}(S) = \langle y_{\ell-1} \rightarrow x_{\ell-1}, \dots, y_2 \rightarrow x_2, y_1 \rightarrow x_1 \rangle$. For example, the TS-sequence $S = \langle I_1, \dots, I_5 \rangle$ described in Figure 1 can also be written as $S = \langle v_4 \rightarrow v_5, v_3 \rightarrow v_2, v_2 \rightarrow v_1, v_5 \rightarrow v_4 \rangle$. Similarly, $\text{rev}(S) = \langle I_5, \dots, I_1 \rangle = \langle v_4 \rightarrow v_5, v_1 \rightarrow v_2, v_2 \rightarrow v_3, v_5 \rightarrow v_4 \rangle$.

For an edge $e = xy \in E(G)$, we say that S *makes detour over* e if both $x \rightarrow y$ and $y \rightarrow x$ are members of S . We emphasize that the steps $x \rightarrow y$ and $y \rightarrow x$ is *not* necessarily made by the same token. 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 $x \rightarrow y$ and the number of appearances of $y \rightarrow x$. The *total number of detours* S *makes in* G , denoted by $D_G(S)$, is defined to be $\sum_{e \in E(G)} D_G(S, e)$. As an example, one can verify that the TS-sequence S described in Figure 1 satisfies $D_G(S, v_4 v_5) = 2$ and $D_G(S) = 2$. Let \mathcal{S} be the set of all TS-sequences in G between two independent sets I, J . We define by $D^*(G, I, J) = \min_{S \in \mathcal{S}} D_G(S)$ the smallest number of detours that a TS-sequence between I and J in G can possibly make.

For two TS-sequences $S_1 = \langle x_1 \rightarrow y_1, x_2 \rightarrow y_2, \dots, x_{\ell-1} \rightarrow y_{\ell-1} \rangle$ and $S_2 = \langle x'_1 \rightarrow y'_1, x'_2 \rightarrow y'_2, \dots, x'_p \rightarrow y'_p \rangle$ in a graph G , if the sequence of token-slides $S = \langle x_1 \rightarrow y_1, x_2 \rightarrow y_2, \dots, x_{\ell-1} \rightarrow y_{\ell-1}, x'_1 \rightarrow y'_1, x'_2 \rightarrow y'_2, \dots, x'_p \rightarrow y'_p \rangle$ forms a TS-sequence in G , we define $S = S_1 \oplus S_2$, and say that S is obtained by taking the *concatenation* of S_1 and S_2 .

3. SHORTEST SLIDING TOKEN for spiders

In this section, we show that SHORTEST SLIDING TOKEN for spiders can be solved in polynomial time. More precisely, we claim that

Theorem 1. *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, where n denotes the number of vertices of the given spider G .*

First of all, from the linear-time algorithm for solving SLIDING TOKEN for trees (which also applies for spiders as well) presented in [6], we can simplify our problem as follows. For an independent set I of a tree T , the token on $u \in I$ is said to be (T, I) -*rigid* if for any I' with $I \overset{T}{\leftrightarrow} I'$, $u \in I'$. Intuitively, a (T, I) -rigid token cannot be moved by any TS-sequence in T . One can find all (T, I) -rigid tokens in a given tree T in linear time. Moreover, a TS-sequence between I and J in T exists if and only if the (T, I) -rigid tokens and (T, J) -rigid tokens are the same, and for any component F of the forest obtained from T by removing all vertices where (T, I) -rigid tokens are placed and their neighbors, $|I \cap F| = |J \cap F|$. Thus, for an instance (G, I, J) of SHORTEST SLIDING TOKEN for spiders, we can assume without loss of generality that $I \overset{G}{\leftrightarrow} J$ and there are no (G, I) -rigid and (G, J) -rigid tokens.

3.1. General idea

We now give a brief overview of our approach. For convenience, from now on, let (G, I, J) be an instance of SHORTEST SLIDING TOKEN for spiders satisfying the above assumption. Rough speaking, we aim to construct a TS-sequence in G between I and J of *minimum* length $M^*(G, I, J) + D^*(G, I, J)$, where $M^*(G, I, J)$ and $D^*(G, I, J)$ are respectively the smallest number of token-slides and the smallest number of detours that a TS-sequence between I and J in G can possibly perform, as defined in the previous section. Indeed, the following lemma implies that any TS-sequence in G between I and J must be of length at least $M^*(G, I, J) + D^*(G, I, J)$.

Lemma 2. *Let I, J be two independent sets of a tree T such that $I \overset{T}{\leftrightarrow} J$. Then, for every TS-sequence S between I and J , $\text{len}(S) \geq M^*(T, I, J) + D^*(T, I, J)$.*

Proof. Let $I = \{w_1, w_2, \dots, w_{|I|}\}$. Let S be a TS-sequence between I and J that moves the token t_i on w_i to $f(w_i)$ for some target assignment $f : I \rightarrow J$. For each $i \in \{1, 2, \dots, |I|\}$, let S_i be the sequence of $\text{dist}_T(w_i, f(w_i))$ token-slides that moves t_i from w_i to $f(w_i)$ along the (unique) path $P_{w_i, f(w_i)}$. Note that S_i is not necessarily a TS-sequence.

Let consider the movements of t_i from w_i to $f(w_i)$ in the TS-sequence S . First of all, it is clear that t_i needs to make all moves in S_i . Since the path $P_{w_i, f(w_i)}$ is unique, if t_i makes any move $x \rightarrow y$ that is not in S_i for some edge $xy \in E(T)$, it must also make the move $y \rightarrow x$ later, hence forming detour over e . Let D_1 be the number of detours formed by the token-slides in $S \setminus \bigcup_{i=1}^{|I|} S_i$. Clearly, $\text{len}(S) = \sum_{i=1}^{|I|} \text{dist}_T(w_i, f(w_i)) + D_1$.

The token-slides in $\bigcup_{i=1}^{|I|} S_i$ may also form detour. Let $i, j \in \{1, 2, \dots, |I|\}$ be such that the sequence S_i moves t_i from w_i to $f(w_i)$ and at some point makes the move $x \rightarrow y$, and the sequence S_j moves t_j from w_j to $f(w_j)$ and at some point makes the move $y \rightarrow x$. Together, S_i and S_j form detour over an edge $e = xy \in E(P_{w_i, f(w_i)}) \cap E(P_{w_j, f(w_j)})$. Let D_2 be the number of detours formed by such token-slides. Clearly, $D_G(S) = D_1 + D_2$.

Suppose that for an edge $e = xy \in E(T)$, there exists k_e pairs $(i_1, j_1), (i_2, j_2), \dots, (i_{k_e}, j_{k_e})$ with $1 \leq i_p, j_p \leq |I|$, $i_p \neq j_p$, and for any two pairs (i_p, j_p) and (i_q, j_q) , $i_p \neq i_q$ and $j_p \neq j_q$ ($1 \leq p, q \leq k_e$) such that for each $p \in \{1, 2, \dots, k_e\}$, the sequence S_{i_p} at some point makes the move $x \rightarrow y$, and the sequence S_{j_p} at some point makes the move $y \rightarrow x$. It follows that the vertices $\{w_{i_p}\}_{1 \leq p \leq k_e}$ and $\{f(w_{j_p})\}_{1 \leq p \leq k_e}$ are in $V(T_x^y)$, and the vertices $\{w_{j_p}\}_{1 \leq p \leq k_e}$ and $\{f(w_{i_p})\}_{1 \leq p \leq k_e}$ are in $V(T_y^x)$. We note that $1 \leq k_e \leq \lfloor |I|/2 \rfloor$, and emphasize again that S_{i_p} and S_{j_p} are not necessarily TS-sequences. Let \mathcal{E}_f be the set of all edges of T satisfying the described property with respect to the target assignment f . Then, $D_2 = 2 \sum_{e \in \mathcal{E}_f} k_e$.

Let $e \in \mathcal{E}_f$ be an edge of T as described above. Let g be the target assignment defined as follows: for $1 \leq p \leq k_e$, $g(w_{i_p}) = f(w_{j_p})$, $g(w_{j_p}) = f(w_{i_p})$, and $g(w_i) = f(w_i)$ for $i \notin \{i_1, i_2, \dots, i_{k_e}, j_1, j_2, \dots, j_{k_e}\}$. Then, $\sum_{i=1}^{|I|} \text{dist}_T(w_i, f(w_i)) = \sum_{i=1}^{|I|} \text{dist}_T(w_i, g(w_i)) + 2k_e$, and $\mathcal{E}_g = \mathcal{E}_f \setminus \{e\}$. Using this property repeatedly, we can finally find a target assignment g such that $\mathcal{E}_g = \emptyset$ and $\sum_{i=1}^{|I|} \text{dist}_T(w_i, f(w_i)) = \sum_{i=1}^{|I|} \text{dist}_T(w_i, g(w_i)) + D_2$.

Therefore, $\text{len}(S) = \sum_{i=1}^{|I|} \text{dist}_T(w_i, f(w_i)) + D_1 = \sum_{i=1}^{|I|} \text{dist}_T(w_i, g(w_i)) + D_2 + D_1 \geq M^*(T, I, J) + D^*(T, I, J)$. \square

As a result, it remains to show that any TS-sequence in G between I and J must be of length at most $M^*(G, I, J) + D^*(G, I, J)$, and there exists a specific TS-sequence S in G between I and J whose length is exactly $M^*(G, I, J) + D^*(G, I, J)$. To this end, we shall analyze the following cases.

- **Case 1:** $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$.
- **Case 2:** $0 < \max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$.
- **Case 3:** $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \geq 2$.

In each case, we claim that it is possible to simultaneously determine $D^*(G, I, J)$ and construct a TS-sequence in G between I and J whose length is minimum. More precisely, in **Case 1**, we show that it is always possible to construct a TS-sequence between I and J of length $M^*(G, I, J)$, that is, no detours are

required. (Note that, no TS-sequence can use less than $M^*(G, I, J)$ token-slides.) However, this does not hold in **Case 2**. In this case, we show that in certain conditions, detours cannot be avoided, that is, any TS-sequence must make detours at least one time at some edge of G . More precisely, in such situations, we show that it is possible to construct a TS-sequence between I and J of length $M^*(G, I, J) + 2$, that is, the sequence makes detour at exactly one edge. Finally, in **Case 3**, we show that detours cannot be avoided at all, and it is possible to construct a TS-sequence between I and J of minimum length, without even knowing exactly how many detours it performs. As a by-product, we also describe how one can calculate this (smallest) number of detours precisely.

3.2. When $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$

As mentioned before, in this case, we will describe how to construct a TS-sequence S in G between I and J whose length $\text{len}(S)$ equals $M^*(G, I, J) + D^*(G, I, J)$. In general, to construct any TS-sequence, we need: (1) a target assignment f that tells us the final position a token should be moved to (say, a token on v should finally be moved to $f(v)$); and (2) an ordering of tokens that tells us which token should move first. From the definition of $M^*(G, I, J)$, it is natural to require that our target assignment f satisfies $M^*(G, I, J) = \sum_{w \in I} \text{dist}_G(w, f(w))$. As you will see later, such a target assignment exists, and we can always construct one in polynomial time. We also claim that one can efficiently define a total ordering \prec of vertices in I such that if $x, y \in I$ and $x \prec y$, then the token on x will be moved before the token on y in our desired TS-sequence. Combining these results, our desired TS-sequence will finally be constructed (in polynomial time).

Target assignment. We now describe how to construct a target assignment f such that $M^*(G, I, J) = \sum_{w \in I} \text{dist}_G(w, f(w))$. For convenience, we always assume that the given spider G has body v and $\deg_G(v)$ legs $L_1, \dots, L_{\deg_G(v)}$. 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 ($i \in \{1, 2, \dots, \deg_G(v)\}$), 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 $i \in \{2, \dots, d\}$, 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. $v \in J$). This definition will be helpful when considering tokens placed at the body vertex v .

Under the above assumptions, we design Algorithm 1 for constructing f as below. The next lemma says that Algorithm 1 efficiently produces our desired target assignment.

Lemma 3. *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 $f : I \rightarrow J$ 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 $g : I \rightarrow J$, $\sum_{w \in I} \text{dist}_G(w, g(w)) \geq \sum_{w \in I} \text{dist}_G(w, f(w))$. In other words, f satisfies $M^*(G, I, J) = \sum_{w \in I} \text{dist}_G(w, f(w))$.*

Before proving Lemma 3, we prove the following useful lemma.

Lemma 4. *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 be a target assignment produced from Algorithm 1. Let $I = \{w_1, w_2, \dots, w_{|I|}\}$ be such that $w_1 < w_2 < \dots < w_{|I|}$. Let $i, j, p \in \{1, 2, \dots, |I|\}$ be the indices such that $w_i < \min_{\prec}\{w_j, w_p\}$, i.e., w_i is assigned before w_j and w_p . Then, $\text{dist}_G(w_i, f(w_p)) + \text{dist}_G(w_j, f(w_i)) \geq \text{dist}_G(w_i, f(w_i)) + \text{dist}_G(w_j, f(w_p))$.*

Proof. If $w_i = f(w_i)$, the desired inequality becomes the famous triangle inequality. Thus, we can assume without loss of generality that $w_i \neq f(w_i)$.

Based on the possible relative positions of $w_i, w_j, f(w_i)$, and $f(w_p)$, we consider the following cases.

- **Case 1:** w_i and w_j are in I_L .

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

Input: Two independent sets I, J of a spider G with body v .

Output: A target assignment $f : I \rightarrow J$ such that $M^*(G, I, J) = \sum_{w \in I} \text{dist}_G(w, f(w))$.

```

1: for  $i = 1$  to  $\text{deg}_G(v)$  do
2:   while  $I_{L_i} \neq \emptyset$  and  $J_{L_i} \neq \emptyset$  do
3:     Let  $x \in I_{L_i}$  be such that  $\text{dist}_G(x, v) = \max_{x' \in I_{L_i}} \text{dist}_G(x', v)$ .  $\triangleright$   $x$  is the farthest vertex from  $v$  in
      $I_{L_i}$  that has not yet been assigned
4:     Let  $y \in J_{L_i}$  be such that  $\text{dist}_G(y, v) = \max_{y' \in J_{L_i}} \text{dist}_G(y', v)$ .  $\triangleright$   $y$  is the farthest vertex from  $v$  in
      $J_{L_i}$  that has not yet been assigned
5:      $f(x) \leftarrow y$ ;  $I_{L_i} \leftarrow I_{L_i} \setminus \{x\}$ ;  $J_{L_i} \leftarrow J_{L_i} \setminus \{y\}$ .  $\triangleright$  assign  $y = f(x)$  and remove them from the
     independent sets
6:   end while
7: end for
8: while  $\bigcup_{i=1}^{\text{deg}_G(v)} I_{L_i} \neq \emptyset$  and  $\bigcup_{i=1}^{\text{deg}_G(v)} J_{L_i} \neq \emptyset$  do  $\triangleright$  From this point, for any leg  $L$ , either  $I_L = \emptyset$  or
 $J_L = \emptyset$ .
9:   Take a leg  $L_i$  such that there exists  $x \in I_{L_i}$  satisfying  $\text{dist}_G(x, v) = \min_{x' \in \bigcup_{i=1}^{\text{deg}_G(v)} I_{L_i}} \text{dist}_G(x', v)$ .  $\triangleright$ 
 $x$  is a closest vertex from  $v$  in  $I_{L_i}$  that has not yet been assigned
10:  Take a leg  $L_j$  such that there exists  $y \in J_{L_j}$  satisfying  $\text{dist}_G(y, v) = \max_{y' \in \bigcup_{i=1}^{\text{deg}_G(v)} J_{L_i}} \text{dist}_G(y', v)$ .  $\triangleright$ 
 $y$  is a closest vertex from  $v$  in  $J_{L_i}$  that has not yet been assigned
11:   $f(x) \leftarrow y$ ;  $I_{L_i} \leftarrow I_{L_i} \setminus \{x\}$ ;  $J_{L_j} \leftarrow J_{L_j} \setminus \{y\}$ .
12: end while
13: return  $f$ .
```

- **Case 1.1:** $f(w_i)$ and $f(w_p)$ are in J_L . From Algorithm 1, we always have $\text{dist}_G(v, f(w_i)) > \text{dist}_G(v, f(w_p))$. Since $w_i \in I_L$ is assigned before $w_j \in I_L$, and $f(w_i) \in J_L$, it follows that $\text{dist}_G(v, w_i) > \text{dist}_G(v, w_j)$. (See Figure 2.) We note that the case $w_i = f(w_p)$ can be seen as a special case of Cases (a) or (b) in Figure 2. Similarly, the case $w_j = f(w_p)$ can be seen as a special case of Cases (b), (c), (d), or (f) in Figure 2; and the case $w_j = f(w_i)$ can be seen as a special case of Cases (d) or (e). Similar arguments hold for the next cases.

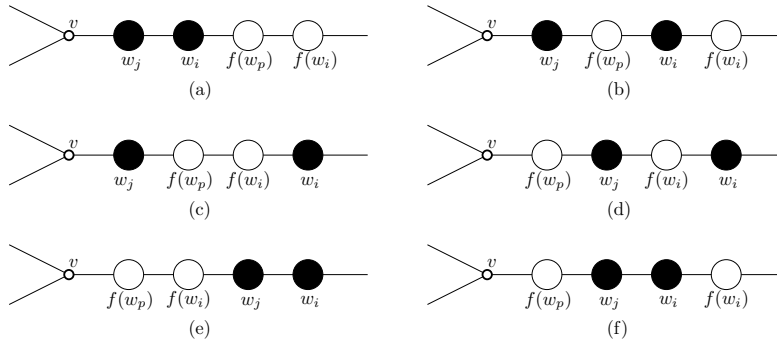


Figure 2: Possible relative positions of $w_i, w_j, f(w_i)$, and $f(w_p)$ when $w_i, w_j \in I_L$, and $f(w_i), f(w_p) \in J_L$.

- **Case 1.2:** $f(w_i)$ and $f(w_p)$ are in $J_{L'}$, $L' \neq L$. (See Figure 3.) From Algorithm 1, we always have $\text{dist}_G(v, f(w_i)) > \text{dist}_G(v, f(w_p))$. Note that if $p = j$, Case (a) of Figure 3 does not happen; otherwise, w_j must be assigned before w_i .
- **Case 1.3:** $f(w_i)$ is in J_L , $f(w_p)$ is in $J_{L'}$, $L' \neq L$. Since $w_i \in I_L$, $f(w_i) \in J_L$, and w_i is assigned before w_j , it follows that $\text{dist}_G(v, w_i) > \text{dist}_G(v, w_j)$. (See Figure 4.)

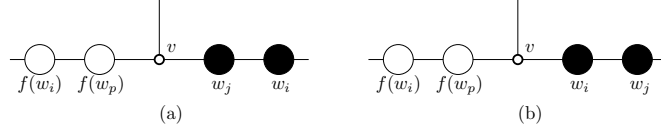


Figure 3: Possible relative positions of $w_i, w_j, f(w_i),$ and $f(w_p)$ when $w_i, w_j \in I_L,$ and $f(w_i), f(w_p) \in J_{L'}, L' \neq L.$

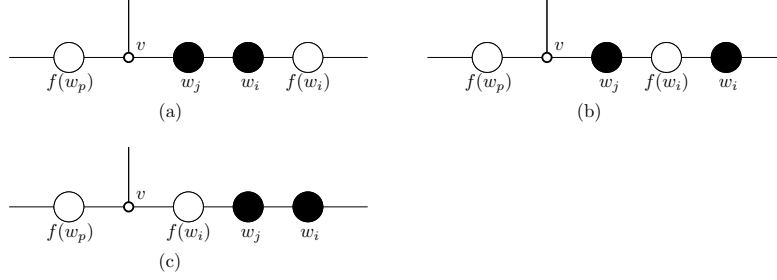


Figure 4: Possible relative positions of $w_i, w_j, f(w_i),$ and $f(w_p)$ when $w_i, w_j \in I_L, f(w_i) \in J_L,$ and $f(w_p) \in J_{L'}, L' \neq L.$

- **Case 1.4:** $f(w_p)$ is in $J_L, f(w_i)$ is in $J_{L'}, L' \neq L.$ Since $w_i \in I_L, f(w_i) \notin J_L,$ and w_i is assigned before $w_j,$ it follows that $\text{dist}_G(v, w_i) < \text{dist}_G(v, w_j)$ and $f(w_j) \notin J_L,$ which implies that $p \neq j.$ (See Figure 5.)

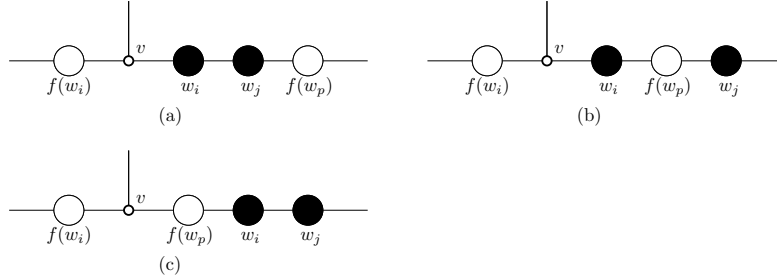


Figure 5: Possible relative positions of $w_i, w_j, f(w_i),$ and $f(w_p)$ when $w_i, w_j \in I_L, f(w_p) \in J_L,$ and $f(w_i) \in J_{L'}, L' \neq L.$

- **Case 1.5:** $f(w_i)$ is in $J_{L'}, L' \neq L, f(w_p)$ is in $J_{L''}, L'' \notin \{L, L'\}.$ (See Figure 6.) Note that if $p = j,$ Case (b) of Figure 6 does not happen; otherwise, w_j must be assigned before $w_i.$

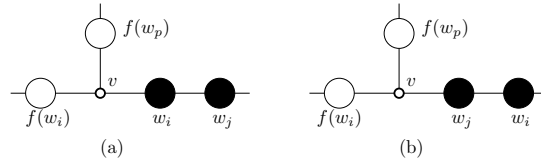


Figure 6: Possible relative positions of $w_i, w_j, f(w_i),$ and $f(w_p)$ when $w_i, w_j \in I_L, f(w_i) \in J_{L'}, L' \neq L,$ and $f(w_p) \in J_{L''}, L'' \notin \{L, L'\}.$

- **Case 2:** $w_i \in I_L$ and $w_j \in I_{L'}, L \neq L'.$

- **Case 2.1:** $f(w_i)$ and $f(w_p)$ are in $J_L.$ From Algorithm 1, we always have $\text{dist}_G(v, f(w_i)) > \text{dist}_G(v, f(w_p)).$ (See Figure 7.)
- **Case 2.2:** $f(w_i)$ and $f(w_p)$ are in $J_{L'}, L' \neq L.$ From Algorithm 1, we always have $\text{dist}_G(v, f(w_i)) > \text{dist}_G(v, f(w_p)).$ Since $w_i \in I_L$ and $f(w_i) \in J_{L'}, L' \neq L,$ Algorithm 1 will assign any vertex in

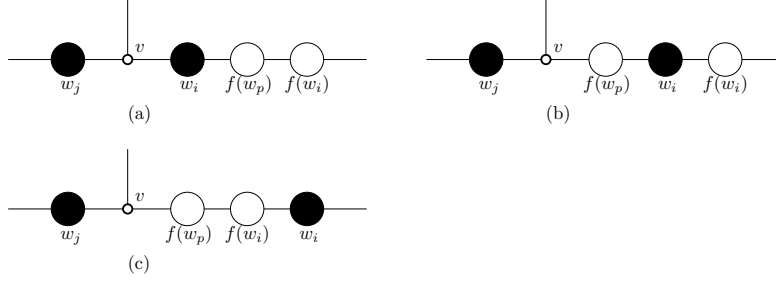


Figure 7: Possible relative positions of w_i , w_j , $f(w_i)$, and $f(w_p)$ when $w_i \in I_L$, $f(w_i), f(w_p) \in J_L$, and $w_j \in I_{L'}$, $L' \neq L$.

$I_{L'}$ to some vertex in $J_{L'}$, which implies $f(w_j) \in J_{L'}$. However, since $w_j \in I_{L'}$ and $f(w_j) \in J_{L'}$, Algorithm 1 must assign w_j before w_i , a contradiction. Thus, this case cannot happen.

- **Case 2.3:** $f(w_i)$ and $f(w_p)$ are in $J_{L''}$, $L'' \notin \{L, L'\}$. From Algorithm 1, we always have $\text{dist}_G(v, f(w_i)) > \text{dist}_G(v, f(w_p))$. (See Figure 8.)

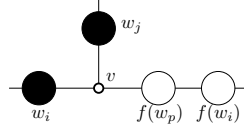


Figure 8: Possible relative positions of w_i , w_j , $f(w_i)$, and $f(w_p)$ when $w_i \in I_L$, $w_j \in I_{L'}$, $L' \neq L$, and $f(w_i), f(w_p) \in J_{L''}$, $L'' \notin \{L, L'\}$.

- **Case 2.4:** $f(w_i)$ is in J_L , $f(w_p)$ is in $J_{L'}$, $L' \neq L$. (See Figure 9.)

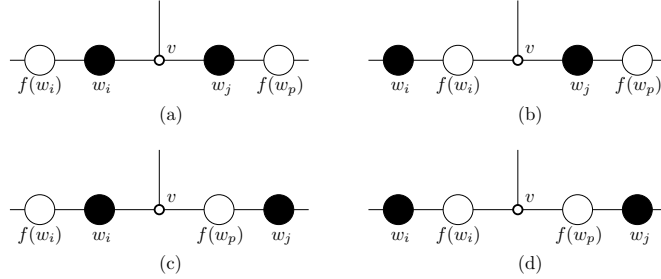


Figure 9: Possible relative positions of w_i , w_j , $f(w_i)$, and $f(w_p)$ when $w_i \in I_L$, $f(w_i) \in J_L$, $w_j \in I_{L'}$, and $f(w_p) \in J_{L'}$, $L' \neq L$.

- **Case 2.5:** $f(w_i)$ is in J_L , $f(w_p)$ is in $J_{L''}$, $L'' \notin \{L, L'\}$. (See Figure 10.)

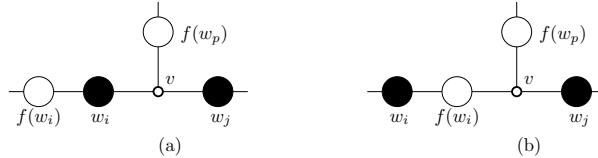


Figure 10: Possible relative positions of w_i , w_j , $f(w_i)$, and $f(w_p)$ when $w_i \in I_L$, $f(w_i) \in J_L$, $w_j \in I_{L'}$, $L' \neq L$, and $f(w_p) \in J_{L''}$, $L'' \notin \{L, L'\}$.

- **Case 2.6:** $f(w_i)$ is in $J_{L'}$, $f(w_p)$ is either in J_L or in $J_{L''}$, $L'' \notin \{L, L'\}$. Since $w_i \in I_L$ and $f(w_i) \in J_{L'} \neq J_L$, Algorithm 1 assigns any $w \in I_{L'}$ to some vertex in $J_{L'}$, which means

$f(w_j) \in J_{L'}$. However, this implies that w_j must be assigned before w_i , a contradiction. Thus, this case cannot happen.

- **Case 2.7:** $f(w_i)$ is in $J_{L''}$, $L'' \notin \{L, L'\}$, $f(w_p)$ is in $J_{L'''}$, $L''' \notin \{L, L', L''\}$. (See Figure 11.)

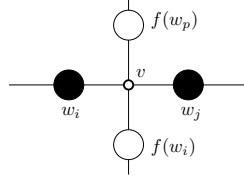


Figure 11: Possible relative positions of w_i , w_j , $f(w_i)$, and $f(w_p)$ when $w_i \in I_L$, $w_j \in I_{L'}$, $f(w_i) \in J_{L''}$, and $f(w_p) \in J_{L'''}$.

In all cases above, it is not hard to see that either our desired inequality holds or the case cannot happen. Thus, our proof is complete. \square

We are now ready to prove Lemma 3.

Proof of Lemma 3. Since Algorithm 1 assigns each vertex in I exactly once, (i) is trivial. It remains to show (ii). Without loss of generality, assume that $I = \{w_1, w_2, \dots, w_{|I|}\}$ is such that $w_1 < w_2 < \dots < w_{|I|}$.

For an arbitrary target assignment g and a target assignment f produced by Algorithm 1, define $k_{gf} = |\{w_i \in I : g(w_i) \neq f(w_i)\}|$. Note that for $g \neq f$, we have $2 \leq k_{gf} \leq |I|$. We prove (ii) by induction on k_{gf} .

Base case: $k_{gf} = 2$. It must happen that there exist i, j with $i < j$, $g(w_i) = f(w_j)$, $g(w_j) = f(w_i)$, and $g(w_\ell) = f(w_\ell)$ for $w_\ell \in I \setminus \{w_i, w_j\}$. It follows from Lemma 4 that $\text{dist}_G(w_i, f(w_j)) + \text{dist}_G(w_j, f(w_i)) \geq \text{dist}_G(w_i, f(w_i)) + \text{dist}_G(w_j, f(w_j))$. Hence, $\text{dist}_G(w_i, g(w_i)) + \text{dist}_G(w_j, g(w_j)) \geq \text{dist}_G(w_i, f(w_i)) + \text{dist}_G(w_j, f(w_j))$.

Inductive step: Given a target assignment f produced from Algorithm 1 and any target assignment g , suppose that for $2 \leq k_{gf} \leq k - 1$, $\sum_{i=1}^{|I|} \text{dist}_G(w_i, g(w_i)) \geq \sum_{i=1}^{|I|} \text{dist}_G(w_i, f(w_i))$. We show that for every target assignment f produced from Algorithm 1 and every target assignment g such that $k_{gf} = k \leq |I|$, the above inequality holds.

Suppose to the contrary that there exist a target assignment f produced from Algorithm 1 and a target assignment g such that $k_{gf} = k \leq |I|$ and $\sum_{i=1}^{|I|} \text{dist}_G(w_i, g(w_i)) < \sum_{i=1}^{|I|} \text{dist}_G(w_i, f(w_i))$. Let i be the smallest index such that $g(w_i) \neq f(w_i)$. Let $p > i$ be such that $g(w_i) = f(w_p)$. Let $j > i$ be such that $g(w_j) = f(w_i)$. We define the assignment g' as follows: $g'(w_i) = f(w_i)$, $g'(w_j) = f(w_p)$, and for $w_\ell \in I \setminus \{w_i, w_j\}$, $g'(w_\ell) = g(w_\ell)$. Thus, $k_{g'f} \leq k - 1$, and by inductive hypothesis, $\sum_{i=1}^{|I|} \text{dist}_G(w_i, g'(w_i)) \geq \sum_{i=1}^{|I|} \text{dist}_G(w_i, f(w_i))$. Hence, $\sum_{i=1}^{|I|} \text{dist}_G(w_i, g'(w_i)) \geq \sum_{i=1}^{|I|} \text{dist}_G(w_i, f(w_i)) > \sum_{i=1}^{|I|} \text{dist}_G(w_i, g(w_i))$. By definition of g' , it follows that $\text{dist}_G(w_i, g(w_i)) + \text{dist}_G(w_j, g(w_j)) < \text{dist}_G(w_i, g'(w_i)) + \text{dist}_G(w_j, g'(w_j))$. In other words, $\text{dist}_G(w_i, f(w_p)) + \text{dist}_G(w_j, f(w_i)) < \text{dist}_G(w_i, f(w_i)) + \text{dist}_G(w_j, f(w_p))$. However, this contradicts Lemma 4. Our proof is now complete. \square

We note that Algorithm 1 works even when the legs are labeled arbitrarily. However, our labeling of the legs of G will be useful when we use the produced target assignment for constructing a TS-sequence of length $M^*(G, I, J)$ between I and J in G .

Token ordering. Intuitively, we want to have a total ordering \prec of vertices in I such that if $x \prec y$, the token placed at x should be moved before the token placed at y . Ideally, once the token is moved to its final destination, it will never be moved again. From Algorithm 1, the following natural total ordering of vertices in I can be derived: for $x, y \in I$, set $x < y$ if x is assigned before y . Unfortunately, such an ordering does not always satisfy our requirement. However, we can use it as a basis for constructing our desired total ordering of vertices in I .

Before showing how to construct \prec , we define some useful notation. Let $f : I \rightarrow J$ be a target assignment produced from Algorithm 1. For a leg L of G and a vertex $x \in I_L \cup J_L$, we say that the leg L *contains* x , and

x is *inside* L . For each leg L of G , we define $I_L^1 = \{w \in I_L : f(w) \notin J_L\}$ and $I_L^2 = \{w \in I_L : f(w) \in J_L\}$. Roughly speaking, a token in I_L^1 (resp. I_L^2) must finally be moved to a target outside (resp. inside) the leg L . Given a total ordering \triangleleft on vertices of I and a vertex $x \in I$, we define $K(x, \triangleleft) = N_G[P_{xf(x)}] \cap \{y \in I : x \triangleleft y\}$. Intuitively, if $y \in K(x, \triangleleft)$, then in order to move the token on x to its final target $f(x)$, one should move the token on y beforehand. In some sense, the token on y is an “obstacle” that forbids moving the token on x to its final target $f(x)$. If $x \in I_L$ for some leg L of G , we define $K^1(x, \triangleleft) = K(x, \triangleleft) \cap I_L^1$ and $K^2(x, \triangleleft) = K(x, \triangleleft) \cap I_L^2$. As before, a token in $K^1(x, \triangleleft)$ (resp. $K^2(x, \triangleleft)$) must finally be moved to a target outside (resp. inside) the leg L containing x . By definition, it is not hard to see that I_L^1 and I_L^2 (resp. $K^1(x, \triangleleft)$ and $K^2(x, \triangleleft)$) form a partition of I_L (resp. $K(x, \triangleleft)$).

Ideally, in our desired total ordering \prec , for any $w \in I$, we must have $K(w, \prec) = \emptyset$. This enables us to move tokens in a way that any token placed at $w \in I$ is moved directly to its final target $f(w)$ through the (unique) shortest path $P_{wf(w)}$ between them; and once a token is moved to its final target, it will never be moved again. We note that this does not always hold for the natural total ordering $<$ defined from Algorithm 1 above. Therefore, a natural approach is to construct \prec from $<$ by looking at all $w \in I$ with $K(w, <) \neq \emptyset$ and reversing the ordering of any pair of vertices that makes our desired moving strategy impossible. A formal description of this procedure is in Algorithm 2 below.

To provide a better explanation of Algorithm 2, we briefly introduce the cases that require changing the ordering $<$. Assume that $w \in I$ is such that $K(w, <) \neq \emptyset$.

- **Ordering between w and vertices in $K(w, <)$.** For each $x \in K(w, <)$, originally $w < x$, but in the new ordering, $x \prec w$. That is, to move the token on w , one should move any “obstacle” (which belongs to $K(w, <)$) beforehand;
- **Ordering between vertices in $K^2(w, <)$.** If $K^2(w, <) \neq \emptyset$, the token on w and any token in $K^2(w, <)$ must be moved to targets inside the leg L containing w . (If $f(w) \notin J_L$ then any “obstacle” between w and $f(w)$ must be moved to targets outside w , which means $K^2(w, <)$ is empty.) Consequently, for $x, y \in K^2(w, <)$, if $x < y$, the token on x should move after the token on y , that is, we should define $x \succ y$.
- **Ordering of vertices between $K^1(w, <)$ and $K^2(w, <)$.** If both $K^1(w, <)$ and $K^2(w, <)$ are non-empty, then it is better (but not strictly required) if we move the tokens in $K^1(w, <)$ before moving any token in $K^2(w, <)$. Originally, vertices in $K^1(w, <)$ (whose targets is outside L) is assigned after those in $K^2(w, <)$ (whose targets is inside L) in Algorithm 1. Intuitively, this is because tokens in $K^1(w, <)$ is “closer” to the body vertex v than those in $K^2(w, <)$, and moving tokens in $K^1(w, <)$ creates “empty space” in L for moving tokens in $K^2(w, <)$ later.

Note that when changing the ordering of vertices between $K^1(w, <)$ and $K^2(w, <)$, we also affect the ordering between vertices in $I_L^1 \supseteq K^1(w, <)$. However, the ordering of vertices in I_L^1 should remain unchanged, since Algorithm 1 always assign vertices in I_L^1 whose distance is closest to the body vertex v first. Thus, for each $x \in I_L^1 \setminus K^1(w, <)$ and $y \in K^1(w, <) \cup K^2(w, <) \cup \{w\}$, we need to set $x \prec y$.

The next lemma (Lemma 5) says that Algorithm 2 correctly produces a total ordering \prec on vertices of I such that $K(w, \prec) = \emptyset$ for every $w \in I$. Intuitively, Lemma 5(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 $K(w_i, \prec) \cup I_L^1 \cup \{w_i\}$ of I_L contains “candidates” for “re-ordering”. That is, the process of changing the ordering of tokens in each iteration of Algorithm 2 will not affect the ordering between tokens inside and outside L . Lemma 5(iii) guarantees that after “re-ordering”, w_i will never be chosen again³, 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.

³ $K(w_i, \prec) = \emptyset$ 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.

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 .

```

1: while there exists  $w$  such that  $K(w, <) \neq \emptyset$  do
2:   Let  $w$  be the smallest element of  $I$  with respect to  $<$  such that  $K(w, <) \neq \emptyset$ .
3:   Let  $L$  be the leg of  $G$  such that  $w \in I_L$ .
4:   for  $x \in K(w, <)$  do
5:     Set  $x \prec w$ .
6:   end for
7:   if  $|K^2(w, <)| \geq 2$  then
8:     For  $x, y \in K^2(w, <)$ , if  $x < y$ , then set  $x \succ y$ .
9:   end if
10:  if  $\min\{|K^1(w, <)|, |K^2(w, <)|\} \geq 1$  then
11:    For  $x \in K^1(w, <)$  and  $y \in K^2(w, <)$ , set  $x \prec y$ .
12:  end if
13:  if  $\min\{|K^1(w, <)|, |I_L^1 \setminus K^1(w, <)|\} \geq 1$  then
14:    For  $x \in I_L^1 \setminus K^1(w, <)$  and  $y \in K^1(w, <) \cup K^2(w, <) \cup \{w\}$ , set  $x \prec y$ .
15:  end if
16:  For  $x, y \in I$  whose ordering has not been defined, if  $x < y$  then set  $x \prec y$ .
17:  Re-define  $<$  to use in the next iteration by setting  $x < y$  if  $x \prec y$  for every  $x, y \in I$ .
18: end while
19: return The total ordering  $\prec$  of vertices in  $I$ .

```

Lemma 5. 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 $f : I \rightarrow J$ be a target assignment produced from Algorithm 1, and $<$ be the corresponding natural total ordering on vertices of I . Assume that $I = \{w_1, w_2, \dots, w_{|I|}\}$ is such that $w_1 < w_2 < \dots < w_{|I|}$. 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 2–17 of Algorithm 2, where the corresponding vertex w is replaced by w_i . Then,

(ii-1) If $x \in K(w_i, <)$, then $x \succ w_i$ 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 $x \succ y$ 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 $K(w_j, \prec) = K(w_j, <)$.

Proof. First of all, note that $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$ is equivalent to saying that both $I \cap N_G(v)$ and $J \cap N_G(v)$ are empty.

(i) We first show that $K(w_i, <) \subseteq I_L$. If $f(w_i) \in J_L$, then since $I \cap N_G(v) = \emptyset$, it follows that $K(w_i, <) \subseteq I \cap V(P_{w_i f(w_i)}) \subseteq I_L$. Now, we consider the case $f(w_i) \in J_{L'} \neq J_L$ for some leg L' of G . Let $x \in N_G[P_{w_i f(w_i)}] \cap I_{L'}$. We claim that $x < w_i$, which then implies $x \notin K(w_i, <)$ and therefore $K(w_i, <) \subseteq I_L$. Since $w_i \in I_L$ and $f(w_i) \notin J_L$, it follows that for any $x \in I_{L'}$, $f(x) \in J_{L'}$, and hence by Algorithm 1, $x < w_i$.

Now, we show that $f(w_i) \in J_L$, which by definition means $w_i \in I_L^2$. Suppose to the contrary that $f(w_i) \notin J_L$. For a vertex $w \in I_L \cap N_G[P_{w_i f(w_i)}]$, we must have $\text{dist}_G(w, v) < \text{dist}_G(w_i, v)$, which means that $w < w_i$. Thus, $K(w_i, <) = \emptyset$, which contradicts the definition of w_i .

(ii) We prove (ii-4) and (ii-5). Other statements are followed immediately from Algorithm 2.

(ii-4) Let $x \in K^1(w_i, <)$ and $y \in K^2(w_i, <)$. Clearly, by Algorithm 2, $x \prec y$. It remains to show that $x > y$ holds. To see this, note that $K^1(w_i, <) \subseteq I_L^1$ and $K^2(w_i, <) \subseteq I_L^2$, and Algorithm 1 always assigns vertices in I_L^2 (lines 1–7) before those in I_L^1 (lines 8–12).

(ii-5) Let $x \in I_L^1 \setminus K^1(w_i, <)$ and $y \in K^1(w_i, <)$. From Algorithm 2, it suffices to show $w_i < x < y$. For every $x \in I_L^1 \setminus K^1(w_i, <)$, since $w_i \in I_L^2$, using a similar argument as in (ii-4), we have $w_i < x$. It remains to show that for every $x \in I_L^1 \setminus K^1(w_i, <)$ and $y \in K^1(w_i, <)$, $x < y$. To see this, it is sufficient to show that if $y \in K^1(w_i, <)$, for any $z \in I_L^1$ with $z > y$, we have $z \in K^1(w_i, <)$. (Recall that since $<$ is a total ordering on I , either $x < y$ or $y < x$ and here we show that the later case cannot happen.) Indeed, since $z \in I_L^1$ and $z > y$, Algorithm 1 implies $\text{dist}_G(y, v) < \text{dist}_G(z, v) < \text{dist}_G(w_i, v)$ (note that $w_i \in I_L^2$). Since $y \in K^1(w_i, <) \subseteq I_L$, we have $\text{dist}_G(f(w_i), v) \leq \text{dist}_G(y, v) < \text{dist}_G(w_i, v)$. Hence, $\text{dist}_G(f(w_i), v) \leq \text{dist}_G(z, v) < \text{dist}_G(w_i, v)$, which means $z \in K^1(w_i, <)$.

(iii) It follows immediately from Algorithm 2 that $K(w_i, \prec) = \emptyset$. It remains to show that if w_j is the smallest element in I (with respect to the ordering \prec) such that $K(w_j, \prec) \neq \emptyset$, then $K(w_j, \prec) = K(w_j, <)$.

Note that if w_{i-1} exists, then $w_j \succ w_{i-1}$; otherwise, it contradicts the assumption that w_i is the smallest member of I (with respect to the ordering $<$) such that $K_{w_i} \neq \emptyset$. On the other hand, by (ii), $w_j \succ w_i$ if and only if $w_j > w_i$. Thus, for any $w \in I$, $w \succ w_j$ if and only if $w > w_j$, which implies $K(w_j, \prec) = K(w_j, <)$.

It remains to consider the case when $w_j \in K(w_i, <) \cup I_L^1 \cup \{w_i\}$.

– We first show that if $w_j \in I_L^1 \cup \{w_i\}$, then $K(w_j, \prec) = \emptyset$. If $w_j = w_i$, we are done. Let consider the case $w_j \in I_L^1$. We claim that for every $x \in I \cap N_G[P_{w_j f(w_j)}] = (I_L \cup I_{L'} \cup (I \cap \{v\})) \cap N_G[P_{w_j f(w_j)}]$, we have $x \prec w_j$, which means $x \notin K(w_j, \prec)$. Here $L' \neq L$ is the leg of G such that $f(w_j) \in J_{L'}$.

* By Algorithm 1, if $v \in I$, for every $w \in \bigcup_L I_L^1$, we have $v < w$. By (ii), $v \prec w$. Since $w_j \in I_L^1$ for some leg L , the above arguments hold for w_j .

* If $x \in I_{L'} \cap N_G[P_{w_j f(w_j)}]$ and $x \neq v$, then by (ii), Algorithm 1, and the assumption $I \cap N_G(v) = \emptyset$, it follows that $x < w_j$ and $x \prec w_j$.

* If $x \in I_L \cap N_G[P_{w_j f(w_j)}]$ and $x \neq v$, then $\text{dist}_G(x, v) < \text{dist}_G(w_j, v)$ (because $f(w_j) \notin J_L$), and therefore $x \in I_L^1$ and $x < w_j$. By (ii), we have $x \prec w_j$.

– Since $K^1(w_i, <) \subseteq I_L^1$, it suffices to consider $w_j \in K^2(w_i, <)$. In this case, we claim that $K(w_j, \prec) = K(w_j, <)$. Let $K^2(w_i, <) = \{x_1, x_2, \dots, x_{|K^2(w_i, <)|}\}$ be such that $x_1 \prec x_2 \prec \dots \prec x_{|K^2(w_i, <)|}$. Since $w_i \in I_L$ and $f(w_i) \in J_L$, it follows that $x_1 > x_2 > \dots > x_{|K^2(w_i, <)|}$ and $\text{dist}_G(f(x_1), v) < \dots < \text{dist}_G(f(x_{|K^2(w_i, <)|}), v) < \text{dist}_G(f(w_i), v) < \text{dist}_G(x_1, v) < \dots < \text{dist}_G(x_{|K^2(w_i, <)|}, v) < \text{dist}_G(w_i, v)$. Since for every $p \in \{2, 3, \dots, |K^2(w_i, <)|\}$, $\text{dist}_G(f(x_1), v) < \text{dist}_G(f(x_p), v) < \text{dist}_G(x_1, v) < \text{dist}_G(x_p, v)$, it follows that if $K(x_1, \prec) = \emptyset$, for every $p \in \{2, 3, \dots, |K^2(w_i, <)|\}$, we have $K(x_p, \prec) = \emptyset$. Since $w_j \in K^2(w_i, <)$ and $K(w_j, \prec) \neq \emptyset$, one of the $K(x_p, \prec)$ ($p \in \{1, 2, 3, \dots, |K^2(w_i, <)|\}$) must be non-empty. Hence, $K(x_1, \prec) \neq \emptyset$, and therefore $w_j = x_1$. On the other hand, note that for every $x \in K^2(w_i, <) \setminus \{x_1\}$, we have $x \succ x_1$ and $x \notin N_G[P_{x_1 f(x_1)}]$. Thus, for every $w \in K(w_j, \prec) = K(x_1, \prec)$ (and then $w \succ w_j$), it must happen that $w > w_j$.

Moreover, $w_j = x_1$ is the maximum element in $K^2(w_i, \prec)$ with respect to the ordering \prec . Thus, for every $w > w_j$, $w \notin K^2(w_i, \prec)$, and therefore $w \succ w_j$. Hence, $K(w_j, \prec) = K(w_j, \prec)$.

□

Now, we are ready to prove the following lemma.

Lemma 6. *Let (G, I, J) be an instance of SHORTEST SLIDING TOKEN for spiders where the body v of G satisfies $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. 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 $\text{len}(S) = M^*(G, I, J)$.*

Proof. Let f be a target assignment produced from Algorithm 1 and \prec be a corresponding total ordering defined in Algorithm 2. For convenience, for $x, y \in I$, if $x < y$ and $x \prec y$, we say that Algorithm 2 *preserves* the ordering between x and y .

Assume that $I = \{w_1, \dots, w_{|I|}\}$ is such that $w_1 \prec \dots \prec w_{|I|}$. Let S be a sequence of token-slides constructed as follows: for each $w_i \in I$ ($i \in \{1, 2, \dots, |I|\}$), slide the token t_i on w_i to $f(w_i)$ along the path $P_{w_i f(w_i)}$ (using exactly $\text{dist}_G(w_i, f(w_i))$ token-slides). Clearly, $\text{len}(S) = M^*(G, I, J)$ (Lemma 3).

Since Algorithm 1 takes $O(n)$ time, Algorithm 2 takes $O(n^2)$ time, and S uses $O(n)$ token-slides for each token in I , it follows that the construction of S takes $O(n^2)$ time.

To conclude this proof, we show that S is actually a TS-sequence in G by induction on $i \in \{1, 2, \dots, |I|\}$.

Base case: $i = 1$. Since for every $j > 1$, $w_j \notin N_G[P_{w_1 f(w_1)}]$, t_1 clearly can be slid from w_1 to $f(w_1)$ along $P_{w_1 f(w_1)}$.

Inductive step: Assume that for $j \leq i - 1$ ($i \in \{2, 3, \dots, |I|\}$), t_j can be slid from w_j to $f(w_j)$ along $P_{w_j f(w_j)}$. We show that t_i can be slid from w_i to $f(w_i)$ along $P_{w_i f(w_i)}$. Suppose to the contrary that it is not. Note that by Algorithm 2, for every $j > i$, $w_j \notin N_G[P_{w_i f(w_i)}]$. Thus, there must be some $j < i$ such that $f(w_j) \in N_G[P_{w_i f(w_i)}]$. In other words, after t_j is moved from w_j to $f(w_j)$, it becomes an ‘‘obstacle’’ that forbids sliding t_i from w_i to $f(w_i)$. We consider the following cases.

- **Case 1:** $w_i \in I_L$ and $f(w_i) \in J_L$ for some leg L of G . Since $f(w_j) \in N_G[P_{w_i f(w_i)}]$, we have $f(w_j) \in J_L$. Since t_j is moved before t_i from w_j to $f(w_j) \in J_L$, it follows that $w_j \in I_L$. Indeed, if $w_j \notin I_L$, the vertex w_i must be assigned before w_j in Algorithm 1, and Algorithm 2 preserves that ordering (see Lemma 5(ii)), which means $w_i \prec w_j$, a contradiction. Now, if $\text{dist}_G(w_j, v) > \text{dist}_G(w_i, v)$, we must have $\text{dist}_G(f(w_j), v) > \text{dist}_G(w_i, v)$; otherwise t_i is an obstacle that forbids sliding t_j to $f(w_j)$, which contradicts our inductive hypothesis. We note that the existence of $f(w_j)$ implies that $w_i \neq f(w_i)$. If $\text{dist}_G(f(w_i), v) < \text{dist}_G(w_i, v)$, then $f(w_j) \in N_G[w_i]$, which also means $w_i \in N_G[P_{w_j f(w_j)}]$. This contradicts the assumption $w_j \prec w_i$. Therefore, $\text{dist}_G(f(w_i), v) > \text{dist}_G(w_i, v)$. Since $f(w_j) \in N_G[w_i]$ and $\text{dist}_G(f(w_j), v) > \text{dist}_G(w_i, v)$, we must have $\text{dist}_G(w_i, v) < \text{dist}_G(f(w_j), v) < \text{dist}_G(f(w_i), v)$. On the other hand, since $w_i, w_j \in I_L$, $f(w_i), f(w_j) \in J_L$, and $\text{dist}_G(w_j, v) > \text{dist}_G(w_i, v)$, by Algorithm 1, we must have $\text{dist}_G(f(w_j), v) > \text{dist}_G(f(w_i), v)$, which is a contradiction. Using a similar argument, one can show that the case $\text{dist}_G(w_j, v) < \text{dist}_G(w_i, v)$ also leads to a contradiction.
- **Case 2:** $w_i \in I_L$ and $f(w_i) \in J_{L'}$, where L and L' are two distinct legs of G . First of all, recall that the legs L_i 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 $1 \leq i \leq j \leq \text{deg}_G(v)$. Therefore, if $|I_L| \neq |J_L|$ for some leg L of G , we have $|I_{L_1}| \leq |J_{L_1}|$. To see this, note that if $|I_{L_1}| > |J_{L_1}|$, there must be some $i \in \{2, 3, \dots, \text{deg}_G(v)\}$ such that $|I \cap V(L_i)| = |I_{L_i}| < |J_{L_i}| = |J \cap V(L_i)|$ (because $\sum_{i=1}^{\text{deg}_G(v)} I_{L_i} = |I| = |J| = \sum_{i=1}^{\text{deg}_G(v)} J_{L_i}$), which means $|I \cap V(L_i)| - |J \cap V(L_i)| \leq -1$. On the other hand, one can verify that $|I \cap V(L_1)| - |J \cap V(L_1)| > -1$. This contradicts our assumption that $|I \cap V(L_1)| - |J \cap V(L_1)| \leq |I \cap V(L_i)| - |J \cap V(L_i)|$. Thus, $|I_{L_1}| \leq |J_{L_1}|$. Note that if $v \in J$ and $|I_{L_1}| = |J_{L_1}|$ then one can verify that $|I \cap V(L_1)| - |J \cap V(L_1)| \geq 0$. As before, there must be some $i \in \{2, 3, \dots, \text{deg}_G(v)\}$ such that $|I \cap V(L_i)| = |I_{L_i}| < |J_{L_i}| = |J \cap V(L_i)|$, which means $|I \cap V(L_i)| - |J \cap V(L_i)| < 0 \leq |I \cap V(L_1)| - |J \cap V(L_1)|$, a contradiction. Therefore, if $v \in J$, we have $|I_{L_1}| < |J_{L_1}|$. It follows that if $v \in I$ (resp. $v \in J$), then $v \in I_{L_1}^2$ (resp. $f^{-1}(v) \in I_{L_1}^1$). Moreover, Algorithm 1 and Lemma 5(ii) implies that if $v \in J$, then $f^{-1}(v) = \max_{\prec}\{w : w \in \bigcup_L I_L^1\}$. Since $w_j \prec w_i \in \bigcup_L I_L^1$, we have $w_j \neq f^{-1}(v)$, and hence $f(w_j) \neq v$. Since $f(w_j) \in N_G[P_{w_i f(w_i)}]$,

$I \cap N_G(v) = J \cap N_G(v) = \emptyset$, and $f(w_j) \neq v$, $f(w_j)$ must belong to either J_L or $J_{L'}$. If $f(w_j)$ belongs to $J_{L'}$, then since $f(w_j) \in N_G[P_{w_i f(w_i)}]$, it follows that $\text{dist}_G(f(w_j), v) < \text{dist}_G(f(w_i), v)$, which means that $w_j \notin J_{L'}$. It follows that w_j is assigned after w_i , and since Algorithm 2 preserves this ordering, $w_j \succ w_i$, which is a contradiction. Hence, $f(w_j)$ belongs to J_L . Additionally, from Algorithm 1, since $f(w_i) \in J_{L'} \neq J_L$, w_j must belong to I_L and $\text{dist}_G(w_j, v) > \text{dist}_G(w_i, v)$. As t_i is not an obstacle that forbid sliding t_j from w_j to $f(w_j)$, it follows that $f(w_j) \in N_G[w_i]$, which means $w_i \in N_G[P_{w_j f(w_j)}]$. This contradicts our assumption that $w_j \prec w_i$.

Hence, t_i can be slid from w_i to $f(w_i)$ along $P_{w_i f(w_i)}$. Our proof is now complete. \square

In Lemma 6, 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 7. *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 $\text{len}(S) = M^*(G, I, J)$.*

Proof. Let f be a target assignment produced from Algorithm 1. From the assumption, we have $\bigcup_L I_L^1 = \emptyset$, i.e., for every leg L , if $w \in I_L$, then $f(w) \in J_L$. Now, for a leg L , let $I_L = \{w_1, w_2, \dots, w_{|I_L|}\}$ be such that $w_1 \prec w_2 \prec \dots \prec w_{|I_L|}$, where \prec is the ordering produced from Algorithm 2. Let S_L be a sequence of token-slides constructed as follows: for each $w_i \in I_L$ ($i \in \{1, 2, \dots, |I_L|\}$), slide the token on w_i to $f(w_i) \in J_L$ along the path $P_{w_i f(w_i)}$ (using exactly $\text{dist}_G(w_i, f(w_i))$ token-slides). From the proof of Lemma 6 (see **Case 1**), S_L is indeed a TS-sequence in G of length $\text{len}(S_L) = \sum_{w \in I_L} \text{dist}_G(w, f(w))$ that reconfigures I_L to J_L , and S_L can be constructed in $O(n^2)$ time. Moreover, since $\bigcup_L I_L^1 = \emptyset$, for two distinct legs L, L' , the concatenation $S_L \oplus S_{L'}$ is also a TS-sequence in G . Thus, a TS-sequence S in G between I and J can be constructed by taking the concatenation of all S_L . Clearly, $\text{len}(S) = M^*(G, I, J)$, and S can be constructed in $O(n^2)$ time. \square

Using Lemma 7, from this point forward, 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 $|I_L| \neq |J_L|$.

3.3. When $0 < \max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$

This section is devoted to showing the following lemma.

Lemma 8. *Let (G, I, J) be an instance of SHORTEST SLIDING TOKEN for spiders where the body v of G satisfies $0 < \max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. 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 $M^*(G, I, J)$.*

Proof. (i) By assumption, we have $N_G(v) \cap V(L) = \{x\} = \{y\}$. If $N_G(x) \setminus \{v\} \neq \emptyset$, then let v' be such that $N_G(x) \setminus \{v\} = \{v'\}$. (Note that since G is spider, $|N_G(x)| \leq 2$.) We claim that for any TS-sequence S in G between I and J , S must make detour over either $e_1 = xv = yv$ or $e_2 = xv' = yv'$. (See Figure 12.) By assumption, note that there must be some leg K such that $|I_K| > |J_K|$. Let t_w be the token placed at some vertex $w \in I_K$. Since $|I_K| > |J_K|$, at some point, S must slide t_w to some vertex not in K . As G is a spider, at some point, S must slide t_w to v , which means it must slide the token t_x on $x \in I_L$ to some other vertex not in $N_G(v)$ beforehand. There are only two possible movements: at some point, S slides t_x either to v or to v' (and then maybe to some other vertex not in $N_G(v)$).

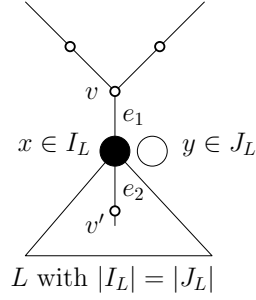


Figure 12: Illustration of Lemma 8(i). Tokens in I (resp. J) are of black (resp. white) color.

- **Case 1: S slides t_x to v and then to some other vertex not in $N_G(v)$.** Let I' be the resulting independent set at this point. Let S_1 and S_2 be the subsequences of S that reconfigure I to I' and I' to J , respectively. Clearly, S_1 slides t_x from x to v at some point. Since $|I'_L| < |I_L|$ (t_x is already moved) and $y \in J_L$, it follows that at some point S_2 must slide some token t_z on some vertex $z \notin I'_L$ to $y = x$. As G is a spider, S_1 must slide t_z to v beforehand, and then moves t_z from v to $y = x$. In summary, $x \rightarrow v$ and $v \rightarrow y = x$ are members of S_1 and S_2 , respectively. Hence, S makes detour over $e_1 = xv$.
- **Case 2: S slides t_x to v' (and then maybe to some other vertex not in $N_G(v)$).** Let I'' be the resulting independent set at this point. Let S_3 and S_4 be the subsequences of S that reconfigure I to I'' and I'' to J , respectively. Clearly, S_3 slides t_x from x to v' at some point. Since $|I''_L| = |I_L|$ and $y \in J_L$, at some point, S_4 must slide t_x from v' to $y = x$ (regardless of which token is finally moved to y). In summary, $x \rightarrow v'$ and $v' \rightarrow y = x$ are members of S_3 and S_4 , respectively. Hence, S makes detour over $e_2 = xv'$.

Since for any TS-sequence S , one of the above cases must happen, we always have $D_G(S) \geq 2$, which means $D^*(G, I, J) \geq 2$.

Now, we describe how to construct a TS-sequence S whose length $\text{len}(S) = M^*(G, I, J) + 2$. Let $I' = I \setminus \{x\} \cup \{v\}$ and $J' = J \setminus \{y\} \cup \{v\}$. Intuitively, I' (resp. J') is obtained from I (resp. J) via a single token-slide that moves the token on $x \in I$ (resp. $y \in J$) to v . Note that this can be done because $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. Recall that $x = y \in I \cap J$. Clearly, $\max\{|I' \cap N_G(v)|, |J' \cap N_G(v)|\} = 0$. Moreover, by Lemma 3, $M^*(G, I, J) = M^*(G, I', J')$. To see this, note that if $f : I \rightarrow J$ (resp. $g : I' \rightarrow J'$) is a target assignment produced from Algorithm 1, then $x = f(x) \in I_L \cap J_L$ and $v = g(v) \in I_{L_1} \cap J_{L_1}$. Let S' be a TS-sequence in G that reconfigures I' to J' as described in Lemma 6. Let $S = \langle x \rightarrow v \rangle \oplus S' \oplus \langle v \rightarrow x \rangle$. Clearly, S is a TS-sequence in G that reconfigures I to J of length $\text{len}(S) = M^*(G, I, J) + 2$, and it can be constructed in $O(n^2)$ time.

- (ii) Let $f : I \rightarrow J$ be a target assignment produced from Algorithm 1, and \prec be a corresponding total ordering on vertices of I produced from Algorithm 2. In each of the following cases, we describe how to construct a TS-sequence S in G between I and J whose length is $M^*(G, I, J)$.

- **Case 1: Only x exists.** First of all, we consider the case $v \in J$ and $f(x) = v$. Let $I' = I \setminus \{x\} \cup \{v\}$. Intuitively, the independent set I' is obtained from I by sliding the token on x to v . Moreover, by Lemma 3, $M^*(G, I, J) = M^*(G, I', J) + 1$. To see this, note that if $g : I' \rightarrow J$ is a target assignment produced from Algorithm 1, then $g(v) = v = f(x)$. Since only x exists, we must have $\max\{|I' \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. Let S' be a TS-sequence in G that reconfigures I' to J as described in Lemma 6. Clearly, the sequence $S = \langle x \rightarrow v \rangle \oplus S'$ is our desired TS-sequence. Without loss of generality, we can now assume that if $v \in J$, then we have $f(x) \neq v$. Suppose that $x \in I_L$ for some leg L of G . We consider the following cases.

- * **Case 1.1:** $f(x) \in J_L$. Let $I_x = \{x\} \cup \{w \in I_L : w \prec x\}$. Let S_1 be a sequence of token-slides constructed as follows: (a) Take the minimum element w of I_x (with respect to \prec) and slide

the token on w to $f(w)$; and (b) Repeat (a) with $I_x \setminus \{w\}$ instead of I_x . From the proof of Lemma 6, it follows that S_1 is indeed a TS-sequence in G of length $\sum_{w \in I_x} \text{dist}_G(w, f(w))$ that moves the token on x to $f(x)$. Intuitively, S_1 only moves tokens “inside” the leg L . Additionally, note that if I' is the resulting independent set obtained from I by performing S_1 , then $M^*(G, I, J) = M^*(G, I', J) + \text{len}(S_1)$ and $\max\{|I' \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. Thus, if S_2 is the TS-sequence in G that reconfigures I' to J , as described in Lemma 6, then $S = S_1 \oplus S_2$ is our desired TS-sequence.

- * **Case 1.2:** $f(x) \in J_{L'}$ for some leg $L' \neq L$ of G . Let $I_x = \{x\} \cup \{w : w \in I_{L'}\}$. From Algorithm 1 and Lemma 5(ii), note that $x = \max_{\prec} \{w : w \in I_x\}$. Let S_1 be the sequence of token-slides constructed as follows: (a) Take the minimum element w of I_x (with respect to \prec) and slide the token on w to $f(w)$; and (b) Repeat (a) with $I_x \setminus \{w\}$ instead of I_x . From the proof of Lemma 6 and the assumption $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$, it follows that S_1 is a TS-sequence in G of length $\sum_{w \in I_x} \text{dist}_G(w, f(w))$ that moves the token on x to $f(x)$. Intuitively, S_1 first moves tokens “inside” the leg L' to their final target vertices in order to “clear the path” for moving the token on x to $f(x)$. Additionally, note that if I' is the resulting independent set obtained from I by performing S_1 , then $M^*(G, I, J) = M^*(G, I', J) + \text{len}(S_1)$ and $\max\{|I' \cap N_G(v)|, |J \cap N_G(v)|\} = 0$. As before, if S_2 is the TS-sequence in G that reconfigures I' to J , as described in Lemma 6, then $S = S_1 \oplus S_2$ is our desired TS-sequence.
- **Case 2: Only y exists.** First of all, we consider the case $v \in I$ and $f(v) = y$. Let $J' = J \setminus \{y\} \cup \{v\}$. Analogously to **Case 1**, we have $M^*(G, I, J) = M^*(G, I, J') + 1$, and $\max\{|I \cap N_G(v)|, |J' \cap N_G(v)|\} = 0$. Then, if S' is the TS-sequence that reconfigures I to J' as described in Lemma 6 then $S = S' \oplus \langle v \rightarrow y \rangle$ is our desired TS-sequence.

Without loss of generality, we can now assume that if $v \in I$ then $f(v) \neq y$. Suppose that $y \in J_L$ for some leg L of G . We consider the following cases.

- * **Case 2.1:** $f^{-1}(y) = z \in I_L$. Let $I_z = \{z\} \cup \{w \in I_L : w \prec z\}$. From Algorithm 1, Lemma 5(ii), and the assumption $y \in N_G(v)$, we must have $I_z \setminus \{z\} \subseteq I_L^1$. Intuitively, the target of any token “inside” the path P_{yz} must be some vertex “outside” L . Then, for each $w \in I_z \setminus \{z\}$, one can use a similar idea as in **Case 1.2** for constructing a TS-sequence S_w that moves the token on $w \in I_L$ to $f(w) \notin J_L$. For notational convention, let S_z be the TS-sequence of $\text{dist}_G(z, f(z))$ token-slides that moves the token on $z \in I_L$ to $f(z) = y \in J_L$ along $P_{zf(z)}$. Let S_1 be a TS-sequence of token-slides constructed as follows: (a) Take the minimum element w of I_z (with respect to \prec) and perform S_w ; and (b) Repeat (a) with $I_z \setminus \{w\}$ instead of I_z . Intuitively, S_1 moves every token “inside” the path P_{zy} (which, by Algorithm 1, is also in I_L^1) “out of” the leg L , and then moves the token on z to y . Let I' be the resulting independent set obtained from I by performing S_1 . Then, note that $I' \cap J = f(I_z) = \bigcup_{w \in I_z} \{f(w)\}$. It follows that the reverse TS-sequence $\text{rev}(S_1)$ of S_1 can be performed with the initial independent set J . Intuitively, $\text{rev}(S_1)$ moves any token on $w \in f(I_z) \subseteq J$ to $f^{-1}(w) \in I$, and tokens not in $f(I_z)$ remains at their original position. Now, let J' be the resulting independent set obtained from J by performing $\text{rev}(S_1)$. Clearly, $J' \cap I = I_z$, and therefore S_1 can be performed with the initial independent set J' . Intuitively, only tokens in I_z are not placed at their final positions. Note that $M^*(G, I, J) = M^*(G, I, J') + \sum_{w \in I_z} \text{len}(S_w)$ and $\max\{|I \cap N_G(v)|, |J' \cap N_G(v)|\} = 0$. Then, if S_2 is the TS-sequence that reconfigures I to J' as described in Lemma 6 then $S = S_2 \oplus S_1$ is our desired TS-sequence.
- * **Case 2.2:** $f^{-1}(y) = z \in I_{L'}$ for some leg $L' \neq L$ of G . Let $I_z = \{z\} \cup \{w \in I_{L'} : w \prec z\}$. From Algorithm 1 and Lemma 5(ii), we must have $I_z \subseteq I_{L'}^1$. Now, the construction of our desired TS-sequence S can be done in a similar manner as in **Case 2.1**.
- **Case 3: Both x and y exist, and $f(x) \neq y$.** We note that in this case $v \notin I \cup J$. Combining the techniques in **Case 1** and **Case 2**, one can construct:
 - * a TS-sequence S_1 that moves the token on x to $f(x)$, and the resulting independent set I' satisfies $M^*(G, I, J) = M^*(G, I', J) + \text{len}(S_1)$; and

* a TS-sequence S_2 whose reverse $\text{rev}(S_2)$ moves the token on y to $f^{-1}(y)$, and the resulting independent set J' after performing $\text{rev}(S_2)$ satisfies $M^*(G, I', J) = M^*(G, I', J') + \text{len}(S_2)$.

Note that $\max\{|I' \cap N_G(v)|, |J' \cap N_G(v)|\} = 0$. Thus, if S_3 is the TS-sequence that reconfigures I' to J' as described in Lemma 6 then $S = S_1 \cup S_3 \cup S_2$ is our desired TS-sequence.

- **Case 4: Both x and y exist, and $f(x) = y$.** We note that in this case $v \notin I \cup J$. From the assumption, it must happen that $x \in I_L$ and $y \in J_{L'}$ for two distinct legs L, L' of G . (If $L = L'$ the Algorithm 1 implies that $|I_L| = |J_L|$, which contradicts our assumption.) Moreover, Algorithm 1 implies that $\bigcup_L I_L^1 = \{x\}$. To see this, note that if there exists $z \in \bigcup_L I_L^1 \setminus \{x\}$ then we must have $1 = \text{dist}_G(x, v) \leq \text{dist}_G(z, v)$ and $1 = \text{dist}_G(f(x), v) \leq \text{dist}_G(f(z), v)$; otherwise, either $z \neq x$ or $f(z) \neq y$ belongs to $N_G(v)$, which contradicts the assumption $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. However, this contradicts Algorithm 1. Thus, we must have $\bigcup_L I_L^1 = \{x\}$. Let $S_1 = \langle x \rightarrow v, v \rightarrow y \rangle$ be the TS-sequence of length $\text{dist}_G(x, y) = 2$ that moves the token on $x \in I_L$ to $y \in J_{L'}$, and let I' be the resulting independent set. This can be done simply because $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. Note that $M^*(G, I, J) = M^*(G, I', J) + 2$, and every leg L of G satisfies $|I'_L| = |J_L|$ (otherwise, $\bigcup_L I'_L \neq \emptyset$, and hence $\bigcup_L I_L^1 \neq \{x\}$, which is a contradiction). Then, if S_2 is the TS-sequence that reconfigures I' to J as described in Lemma 7 then $S = S_1 \oplus S_2$ is our desired TS-sequence.

We have shown how to construct a TS-sequence S in G between I and J whose length is $M^*(G, I, J)$. From the above cases, it is clear that the construction of S takes $O(n^2)$ time. \square

3.4. When $\max\{|I \cap N_G(v)|, |J \cap N_G(v)|\} \geq 2$

In this section, we claim that

Lemma 9. *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 9, we prove the following useful lemma.

Lemma 10. *The value of $\text{cost}(T, I, xy)$ can be calculated in $O(n)$ time for $x \in I$ and $y \in N_T(x)$, where I is an independent set of a tree T on n vertices. Moreover, if $\text{cost}(T, I, xy) < \infty$, then in $O(n)$ time, one can output a TS-sequence $S(T, I, xy)$ in T of length $\text{cost}(T, I, xy)$ such that $S(T, I, xy)$ moves the token on x to y .*

Proof. We modify the algorithm described in [5, Lemma 2]. First, regard x as the root of T . Then, we define $\phi(y)$ for each vertex $y \in V(T)$ from leaves of T to the root u as follows. For each leaf y of T , we set $\phi(y) = \infty$ if $y \in I$; otherwise, $\phi(y) = 1$. For each internal vertex y of T with $y \notin I$, if no children of y is in I , we set $\phi(y) = 1$; otherwise, we set $\phi(y) = 1 + \sum_{w \in I \text{ and } w \text{ is a child of } y} \phi(w)$. For each internal vertex y of T with $y \in I$, we set $\phi(y) = \min_{\text{child } w \text{ of } y} \phi(w)$. Finally, we set $\text{cost}(T, I, xy) = \phi(y)$. By definition, it is not hard to see that the above algorithm correctly computes $\text{cost}(T, I, xy)$. For each $y \in V(T)$, the value $\phi(y)$ is computed exactly once. Thus, $\text{cost}(T, I, xy)$ can be calculated in $O(n)$ time.

Assume that $\text{cost}(T, I, xy) < \infty$. We now show how to construct $S(T, I, xy)$ using the described algorithm. For each $z \in I \setminus \{x\}$ with $\phi(z) < \infty$, we define $c(z)$ to be a child of z such that $\phi(c(z)) = \min_{\text{child } w \text{ of } z} \phi(w)$. For $z \in I$ with $\phi(z) = 1$, clearly $S(T, I, zc(z)) = \langle z \rightarrow c(z) \rangle$. For every $z \in I$ with $1 < \phi(z) < \infty$, set $S(T, I, zc(z)) = \bigoplus_{z' \in I \cap N_{T_{c(z)}}(c(z))} S(T, I, z'c(z')) \oplus \langle z \rightarrow c(z) \rangle$. One can verify that the sequence $S(T, I, zc(z))$ of token-slides is indeed a TS-sequence in T . The TS-sequence $S(T, I, xy)$ is indeed $\bigoplus_{w \in I \cap N_{T_x}(y)} S(T, I, wc(w)) \oplus \langle x \rightarrow y \rangle$. Clearly, we can use this recursive relation to construct $S(T, I, xy)$ in $O(n)$ time. \square

Next, we define a 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 $E(A(T, I, J)) = \{(x, y) : xy \in E(T) \text{ and } |I \cap T_y^x| \leq |J \cap T_y^x|\}$. 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. Figure 13 illustrates an example of the auxiliary graph $A(G, I, J)$ for an instance (G, I, J) of the problem for spiders.

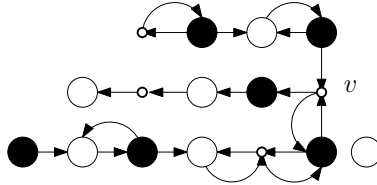


Figure 13: An example of the auxiliary graph $A(G, I, J)$ for an instance of SHORTEST SLIDING TOKEN for spiders. Tokens in I (resp. J) are of black (resp. white) color.

We are now ready to prove Lemma 9.

Proof of Lemma 9. We consider the following cases.

- **Case 1:** $|I \cap N_G(v)| \geq 2$ and $|J \cap N_G(v)| \leq 1$. (See Figure 14.)

From the assumption, note that $v \notin I$. Let $I \cap N_G(v) = \{w_1, w_2, \dots, w_k\}$ ($2 \leq k \leq \deg_G(v)$). For $i \in \{1, 2, \dots, k\}$, let t_i be the token placed at w_i , and L_{w_i} be the leg of G containing w_i . If $N_G(w_i) \setminus \{v\} \neq \emptyset$ then let x_i be such that $N_G(w_i) \setminus \{v\} = \{x_i\}$. (Since G is a spider, w_i has at most two neighbors.)

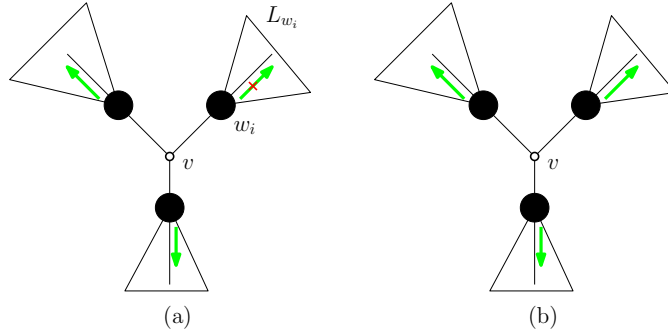


Figure 14: Illustration of **Case 1** of Lemma 9: (a) **Case 1.1**, and (b) **Case 1.2**. Here $k = |I \cap N_G(v)| = 3$, and tokens in I are of black color.

- **Case 1.1:** There exists $i \in \{1, 2, \dots, k\}$ such that t_i is $(L_{w_i}, I_{L_{w_i}})$ -rigid. From [6, Lemma 2], the token t_i is unique, i.e., there is no $j \neq i$ such that t_j is $(L_{w_j}, I_{L_{w_j}})$ -rigid; otherwise, t_i and t_j are both (G, I) -rigid, which contradicts our assumption that there are no (G, I) -rigid tokens. Note that for $j \neq i$, $N_G(w_j) \setminus \{v\} \neq \emptyset$; otherwise, t_j is clearly $(L_{w_j}, I_{L_{w_j}})$ -rigid, which is a contradiction. For each $j \in \{1, 2, \dots, k\}$ with $j \neq i$, let $S_{w_j x_j}$ be the TS-sequence of length $\text{cost}(G, I, w_j x_j)$ that moves t_j from w_j to x_j , as described in Lemma 10. (Since G is a spider, such $S_{w_j x_j}$ is uniquely determined.) Let $S_1^i = \bigoplus_j S_{w_j x_j}$. From the proof of Lemma 6, S_1^i is indeed a TS-sequence that moves t_j from w_j to x_j , for every $j \neq i$. Intuitively, each $S_{w_j x_j}$ only moves tokens “inside” the leg L_{w_j} . Let I' be the resulting independent set (of performing S_1^i). Clearly, $\max\{|I' \cap N_G(v)|, |J \cap N_G(v)|\} \leq 1$. Let S_2^i be the TS-sequence that reconfigures I' to J as described in Lemma 8. We claim that $S = S_1^i \oplus S_2^i$ is a TS-sequence between I and J of

shortest length. It is trivial that S is a TS-sequence between I and J , as it reconfigures I to I' , and then I' to J . To see that S is indeed shortest, note that since $|I_L| \neq |J_L|$ for some leg L of G , any TS-sequence must move t_i to some vertex not in $N_G(v)$; otherwise, some token in I_L where $|I_L| > |J_L|$ cannot be moved to its final target vertex. Since t_i is $(L_{w_i}, I_{L_{w_i}})$ -rigid, the only way is to move t_i “out of” L_{w_i} . Roughly speaking, the token-slides in S_1^i is unavoidable, i.e., any TS-sequence S between I and J contains S_1^i as a subsequence. Since any token-slide in S before S_1^i can only be performed “inside” a particular leg of G , one can assume without loss of generality that S_1^i is performed before any other token-slide in S . Additionally, from Lemma 8, S_2^i must be a TS-sequence of shortest length between I' and J . Hence, S is indeed a TS-sequence of shortest length between I and J .

- **Case 1.2: For every $i \in \{1, 2, \dots, k\}$, t_i is not $(L_{w_i}, I_{L_{w_i}})$ -rigid.** As before, note that x_i exists for every i . For each t_i , using the same technique as in **Case 1.1**, one can indeed construct a TS-sequence S_1^i that moves all t_j ($j \neq i$) from w_j to x_j of length $\text{len}(S_1^i) = \sum_{j \neq i} \text{cost}(G, I, w_j x_j)$, and a TS-sequence S_2^i that reconfigures the resulting independent set (after performing S_1^i) to J . Let $S^i = S_1^i \oplus S_2^i$. Then, S^i is indeed a TS-sequence that reconfigures I to J . Let S be a TS-sequence whose length is smallest among all S^i . We claim that S is indeed our desired TS-sequence. Trivially, S reconfigures I to J . To see that it is indeed shortest, note that since there exists some leg L with $|I_L| \neq |J_L|$, any TS-sequence between I and J must perform one of S_1^i . As before, we can also assume without loss of generality that for any TS-sequence S' between I and J , the sequence S_1^i , if in S' , is performed before any other token-slide in S' . Then, each S^i is of smallest length among all TS-sequence S' between I and J containing S_1^i . Moreover, it is clear from the construction that if S' contains S_1^i as a subsequence then it does not contain any S_1^j for $j \neq i$. Therefore, a TS-sequence S of smallest length among all S^i is indeed our desired TS-sequence.

- **Case 2: $|I \cap N_G(v)| \leq 1$ and $|J \cap N_G(v)| \geq 2$.**

Analogously to **Case 1**, one can also construct a TS-sequence of shortest length between I and J . Intuitively, instead of moving tokens in $I \cap N_G(v)$ (as in **Case 1**), we now move tokens in $J \cap N_G(v)$: keep one token fixed, and move all other tokens to their corresponding neighbors (different from v). Once we have the resulting independent set J' , the reverse of the above TS-sequence can be used to reconfigure J' to J , and by Lemma 8 we already know how to reconfigure I to J' using a smallest possible number of tokens. Combining these two reconfigurations, we now have a TS-sequence that reconfigures I to J . Our desired TS-sequence is the shortest among all (in particular, there are at most $\text{deg}_G(v)$ of them) such TS-sequences between I and J above.

- **Case 3: $|I \cap N_G(v)| \geq 2$ and $|J \cap N_G(v)| \geq 2$.**

A shortest TS-sequence between I and J can be constructed by simply combining the techniques in **Case 1** and **Case 2**.

In all above cases, the construction of our desired TS-sequence S obviously takes $O(n^2)$ time.

We remark that in the described algorithm, $D_G(S)$ was not explicitly calculated. In the remaining part of this proof, we show how to calculate $D_G(S)$. It is sufficient to show how to calculate $D_G(S)$ in **Case 1.1**; other cases can be done in similar manner. From **Case 1.1**, $S = S_1^i \oplus S_2^i$. We note that from Lemma 10 S_1^i itself does not make detour over any edge of G . On the other hand, Lemma 8 implies that S_2^i itself makes detour over at most one edge of G (due to whether Lemma 8(i) holds). From Lemma 2, it remains to calculate the number of detours made by S_1^i and S_2^i together. From the construction of S_1^i , note that each move $x \rightarrow y$ in S_1^i appears exactly once. Consider a move $x \rightarrow y$ in S_1^i such that (y, x) is a directed edge of the corresponding auxiliary graph $A(G, I, J)$. By definition, $|I \cap G_y^x| \geq |J \cap G_y^x|$. Let I' be the resulting independent set after the move $x \rightarrow y$. Then, it can be shown by induction on the number of such moves in S_1^i that $|I' \cap G_y^x| > |J \cap G_y^x|$. It follows that at some point, S_2^i will have to make a move $y \rightarrow x$. Together, these moves form detour over $e = xy \in E(G)$. Since each move $x \rightarrow y$ in S_1^i appears exactly once, we must have $D_G(S) = D_G(S_2^i) + 2|\{(x \rightarrow y) \in S_1^i : (y, x) \in E(A(G, I, J))\}|$. In a similar manner, in

Case 1.2, $D_G(S)$ can be calculated. In **Case 2**, we argue with tokens in J (instead of I) and the auxiliary graph $A(G, J, I)$ (instead of $A(G, I, J)$). Finally, in **Case 3**, we simply combine the arguments in **Cases 1** and **Case 2**. \square

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). We hope that our ideas and approaches described here will provide a useful framework for improving the polynomial-time algorithm for SHORTEST SLIDING TOKEN for trees [23].

References

- [1] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *arXiv preprint*, 2018. [arXiv:1807.05322](https://arxiv.org/abs/1807.05322).
- [2] Marthe Bonamy and Nicolas Bousquet. Token sliding on chordal graphs. In *Proceedings of WG 2017*, volume 10520 of *LNCS*, pages 127–139. Springer, 2017. [doi:10.1007/978-3-319-68705-6_10](https://doi.org/10.1007/978-3-319-68705-6_10).
- [3] Paul S. Bonsma, Marcin Kamiński, and Marcin Wrochna. Reconfiguring independent sets in claw-free graphs. In *Proceedings of SWAT 2014*, volume 8503 of *LNCS*, pages 86–97. Springer, 2014. [doi:10.1007/978-3-319-08404-6_8](https://doi.org/10.1007/978-3-319-08404-6_8).
- [4] Erik D. Demaine and Mikhail Rudoy. A simple proof that the $(n^2 - 1)$ -puzzle is hard. *Theoretical Computer Science*, 732:80–84, 2018. [doi:10.1016/j.tcs.2018.04.031](https://doi.org/10.1016/j.tcs.2018.04.031).
- [5] Erik D. Demaine, Martin L. Demaine, Eli Fox-Epstein, Duc A. Hoang, Takehiro Ito, Hirotaka Ono, Yota Otachi, Ryuhei Uehara, and Takeshi Yamada. Polynomial-time algorithm for sliding tokens on trees. In *Proceedings of ISAAC 2014*, volume 8889 of *LNCS*, pages 389–400. Springer, 2014. [doi:10.1007/978-3-319-13075-0_31](https://doi.org/10.1007/978-3-319-13075-0_31).
- [6] 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. *Theoretical Computer Science*, 600:132–142, 2015. [doi:10.1016/j.tcs.2015.07.037](https://doi.org/10.1016/j.tcs.2015.07.037).
- [7] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.
- [8] Eli Fox-Epstein, Duc A. Hoang, Yota Otachi, and Ryuhei Uehara. Sliding token on bipartite permutation graphs. In *Proceedings of ISAAC 2015*, volume 9472 of *LNCS*, pages 237–247. Springer, 2015. [doi:10.1007/978-3-662-48971-0_21](https://doi.org/10.1007/978-3-662-48971-0_21).
- [9] Martin Gardner. The hypnotic fascination of sliding-block puzzles. *Scientific American*, 210:122–130, 1964.
- [10] Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009. [doi:10.1137/07070440X](https://doi.org/10.1137/07070440X).
- [11] Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005. [doi:10.1016/j.tcs.2005.05.008](https://doi.org/10.1016/j.tcs.2005.05.008).
- [12] Robert A. Hearn and Erik D. Demaine. *Games, puzzles, and computation*. A K Peters, 2009.
- [13] Duc A. Hoang and Ryuhei Uehara. Sliding tokens on a cactus. In *Proceedings of ISAAC 2016*, volume 64 of *LIPICs*, pages 37:1–37:26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. [doi:10.4230/LIPICs.ISAAC.2016.37](https://doi.org/10.4230/LIPICs.ISAAC.2016.37).
- [14] Duc A. Hoang, Eli Fox-Epstein, and Ryuhei Uehara. Sliding tokens on block graphs. In *Proceedings of WALCOM 2017*, volume 10167 of *LNCS*, pages 460–471. Springer, 2017. [doi:10.1007/978-3-319-53925-6_36](https://doi.org/10.1007/978-3-319-53925-6_36).
- [15] Takehiro Ito, Erik D. Demaine, Nicholas J. A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011. [doi:10.1016/j.tcs.2010.12.005](https://doi.org/10.1016/j.tcs.2010.12.005).
- [16] Marcin Kamiński, Paul Medvedev, and Martin Milanić. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012. [doi:10.1016/j.tcs.2012.03.004](https://doi.org/10.1016/j.tcs.2012.03.004).
- [17] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. An exact algorithm for the boolean connectivity problem for k -CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011. [doi:10.1016/j.tcs.2011.04.041](https://doi.org/10.1016/j.tcs.2011.04.041).
- [18] Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, Narges Simjour, and Akira Suzuki. On the parameterized complexity of reconfiguration problems. In *Proceedings of IPEC 2013*, volume 8246 of *LNCS*, pages 281–294. Springer, 2013. [doi:10.1007/978-3-319-03898-8_24](https://doi.org/10.1007/978-3-319-03898-8_24).
- [19] Amer E. Mouawad, Naomi Nishimura, Venkatesh Raman, and Marcin Wrochna. Reconfiguration over tree decompositions. In *Proceedings of IPEC 2014*, volume 8894 of *LNCS*, pages 246–257. Springer, 2014. [doi:10.1007/978-3-319-13524-3_21](https://doi.org/10.1007/978-3-319-13524-3_21).
- [20] D. Ratner and M. Warmuth. Finding a shortest solution for the $N \times N$ -extension of the 15-puzzle is intractable. *Journal of Symbolic Computation*, 10:111–137, 1990.
- [21] Mariana Teatini Ribeiro and Vinícius Fernandes dos Santos. Personal communications, March, 2018.
- [22] Jerry Slocum and Dic Sonneveld. *The 15 puzzle book: how it drove the world crazy*. Socum Puzzle Foundations, 2006.
- [23] Ken Sugimori. Shortest reconfiguration of sliding tokens on a tree, 2018. AAAC 2018, May, 2018.
- [24] Ken Sugimori. Personal communications, May, 2018.
- [25] Tom C. van der Zanden. Parameterized complexity of graph constraint logic. In *Proceedings of IPEC 2015*, volume 43 of *LIPICs*, pages 282–293. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. [doi:10.4230/LIPICs.IPEC.2015.282](https://doi.org/10.4230/LIPICs.IPEC.2015.282).

- [26] Takeshi Yamada and Ryuhei Uehara. Shortest reconfiguration of sliding tokens on a caterpillar. In Mohammad Kaykobad and Rossella Petreschi, editors, *Proceedings of WALCOM 2016*, volume 9627 of *LNCS*, pages 236–248. Springer, 2016. doi:10.1007/978-3-319-30139-6_19.