Title	The independent set reconfiguration problem on some restricted graphs
Author(s)	HOANG, Duc Anh
Citation	
Issue Date	2015-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/12643
Rights	
Description	Supervisor:Ryuhei Uehara, 情報科学研究科, 修士

The independent set reconfiguration problem on some restricted graphs

By HOANG, Duc Anh

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Ryuhei Uehara

March, 2015

The independent set reconfiguration problem on some restricted graphs

By HOANG, Duc Anh (1310064)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Ryuhei Uehara

and approved by
Professor Kunihiko Hiraishi
Professor Atsuko Miyaji

February, 2015 (Submitted)

Abstract

Title: The independent set reconfiguration problem on some restricted graphs.
Author: HOANG, Duc Anh.
Student number: 1310064.
School: School of Information Science, JAIST.
Date of submission: February 2015.
Key words: reconfiguration problem, independent set, token sliding, graph.

Recently, *reconfiguration problems* attract the attention in the field of theoretical computer science. The problem arises when we wish 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. A well-known example is that given two specified satisfiable assignments (assignments which return the TRUE value) A and B to a Boolean formula, one might ask whether A can be transformed into B by changing the assignment of one variable at a time such that each intermediate assignment is also satisfiable. Readers may also remember Rubik's cube and its relatives as examples of reconfiguration puzzles. This kind of reconfiguration problems has been studied extensively for several well-known problems, including the so-called *independent set reconfiguration problem* (ISRECONF).

Recall that an *independent set* in a graph G is a set of pairwise non-adjacent vertices. Given a graph G, and two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of G, imagine that a token (coin) is put at each vertex of $\mathbf{I}_b$, the ISRECONF problem asks whether we can transform $\mathbf{I}_b$ to $\mathbf{I}_r$ via a sequence of independent sets of G, each of which results from the previous one by moving a token under some given *reconfiguration rules*, namely token sliding (TS), token jumping (TJ), and token addition and removal (TAR).

- *Token Sliding* (TS rule): A single token can be slid only along an edge of a graph. The ISRECONF problem under TS rule is also known as the SLIDING TOKEN problem.
- *Token Jumping* (TJ rule): A single token can "jump" to any vertex (including non-adjacent one).
- *Token Addition and Removal* (TAR rule): We can either add or remove a single token at a time if it results in an independent set of cardinality at least a given threshold.

The ISRECONF problem is PSPACE-complete under any of the three reconfiguration rules for general graphs, for planar graphs, for perfect graphs, and even for bounded bandwidth graphs.

The ISRECONF problem under TS rule, in which tokens may only be moved to adjacent vertices, is called the SLIDING TOKEN problem and is of particular theoretical interest. Given two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a graph $G = (V, E)$ such that $|\mathbf{I}_b| = |\mathbf{I}_r|$, and imagine that a token (coin) is placed on each vertex in $\mathbf{I}_b$, the SLIDING TOKEN problem asks whether there exists a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of G such that:

(a) $\mathbf{I}_1 = \mathbf{I}_b$, $\mathbf{I}_\ell = \mathbf{I}_r$, and $|\mathbf{I}_i| = |\mathbf{I}_b| = |\mathbf{I}_r|$ for all i, $1 \leq i \leq \ell$; and

(b) for each i, $2 \leq i \leq \ell$, there is an edge $\{u, v\}$ in G such that $\mathbf{I}_{i-1} \setminus \mathbf{I}_i = \{u\}$ and $\mathbf{I}_i \setminus \mathbf{I}_{i-1} = \{v\}$, that is, $\mathbf{I}_i$ can be obtained from $\mathbf{I}_{i-1}$ by sliding exactly one token on a vertex $u \in \mathbf{I}_{i-1}$ to its adjacent vertex v along $\{u, v\} \in E$.

Such a sequence is called a *reconfiguration sequence* between $\mathbf{I}_b$ and $\mathbf{I}_r$. In computational complexity theory, several PSPACE-hardness results have been proved using reduction from the SLIDING TOKEN problem. SLIDING TOKEN is known to be PSPACE-complete even for planar graphs, and also for bounded treewidth graphs.

In this thesis, we mainly focus on the SLIDING TOKEN problem (i.e. ISRECONF under TS rule) restricted to trees. In 2012, Kamiński et al. gave a linear-time algorithm for solving ISRECONF for even-hole-free graphs (which include trees) under TJ and TAR rules. Indeed, the answer is always YES under the two rules when restricted to even-hole-free graphs (as long as two given independent sets have the same cardinality for the TJ rule.) Furthermore, tokens never make detours in even-hole-free graphs under the TJ and TAR rules. On the other hand, under TS rule, tokens are required to make detours even in trees. In addition, there are NO-instances for trees under TS rule. These are the reasons why the problem for trees under TS rule is much more complicated and was still open, despite the intensive algorithmic research on ISRECONF. In this thesis, we show that SLIDING TOKEN for trees can be solved in linear time. This result was also presented at the 25^{th} International Symposium on Algorithms and Computation (ISAAC 2014, Jeonju, Korea).

Contents

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. To the best of my knowledge, all sources, references and literature used or excerpted during the presentation of this work are properly cited and listed in complete reference to the due source. For the purpose of easy understanding, some small parts of this thesis are quoted with proper citations.

HOANG, Duc Anh.
March 2015.
JAIST, Japan.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor Professor Ryuhei Uehara of Japan Advanced Institute of Science and Technology (JAIST) for the continuous support of my master's degree study and research, for his patience, motivation, enthusiasm, and immense knowledge. Professor Uehara has supported me not only by providing a research assistantship, but also academically and emotionally through the rough road to finish this thesis.

Besides my supervisor, I would like to thank the rest of my thesis committee: Professor Kunihiko Hiraishi (JAIST), and Professor Atsuko Miyaji (JAIST), for their encouragement, insightful feedbacks, and hard questions.

Also, I would like to thank Professor Yota Otachi (JAIST) and Eli Fox-Epstein (Brown University, USA) for their useful comments and discussion during the time of developing the ideas of this thesis.

Especially, I would like to express my sincerest thanks and appreciation to Professor Tetsuo Asano (JAIST) for his support and guidance not only in my research but also in my personal life in Japan.

To the staff and students at Asano and Uehara laboratories, I am grateful for the chance to study in Japan and be a part of the lab. Thank you for welcoming me as a friend and helping to improve my basic knowledge about algorithms and graph theory.

Additionally, I would like to thank the Japan Advanced Institute of Science and Technology (JAIST) for providing me the financial support and good environment for my study.

Lastly, I would like to thank my family for all their love, support, understanding and encouragement.

HOANG, Duc Anh.
March 2015.
JAIST, Japan.

List of Figures

Chapter 1

Introduction

In real-world situations, there may exist many feasible solutions which can be used to solve a single problem. Usually, for saving time, money, etc., it is required that one need to find a way to transform (reconfigure) one solution to another. This gives rise to the study of a collection of combinatorial problems which is known as *reconfiguration problems*. In this chapter, we will present a brief introduction to *reconfiguration problems* and some of its variants, especially the *independent set reconfiguration problem*. We also describe shortly the main result of this thesis [8], which was presented at the 25^{th} International Symposium on Algorithms and Computation (ISAAC 2014).

Reconfiguration problems are the set of problems in which we are given a set of feasible solutions of a problem, together with some reconfiguration rule(s). The question is, using a reconfiguration rule, can we find a step-by-step transformation which transform one solution to another? A well-known example is that given two specified satisfiable assignments (assignments which return the TRUE value) A and B to a Boolean formula, one might ask whether A can be transformed into B by changing the assignment of one variable at a time such that each intermediate assignment is also satisfiable. Readers may also remember Rubik's cube and its relatives as examples of reconfiguration puzzles. Recently, many kind of *reconfiguration problems* have been studied extensively for several well-known problems, including INDEPENDENT SET [1, 3, 5, 7, 11, 12, 14, 16, 17, 19, 20, 23], SATISFIABILITY [10, 18], SET COVER, CLIQUE, MATCHING [14], VERTEX-COLOURING [2, 4, 6, 23], LIST EDGE-COLOURING [13, 15], etc. A recent survey by van den Heuvel [21] gave a very good introduction to this research area.

Among many variants of *reconfiguration problems*, the *independent set reconfiguration problem* (ISRECONF) is of particular theoretical interest. Recall that an *independent set* in a graph G is a set of pairwise non-adjacent vertices. Given a graph G and two independent sets $\mathbf{I}_b, \mathbf{I}_r$, the ISRECONF problem asks if one can transform $\mathbf{I}_b$ to $\mathbf{I}_r$ using a given reconfiguration rule such that all intermediate sets are also independent. Intuitively, imagine that a token (coin) is placed at each vertex of $\mathbf{I}_b$. We want to know if there is a way to transform the set of tokens using a given rule so that after transforming, each vertex of $\mathbf{I}_r$ contains a token and all intermediate sets of tokens are independent. [1] The following reconfiguration rules are mainly studied:

- *Token Sliding* (TS rule) [4, 5, 7, 11, 12, 17, 23]: A single token can be slid only along an edge of a graph. The ISRECONF problem under TS rule is also known as the SLIDING TOKEN problem.

- *Token Jumping* (TJ rule) [5, 16, 17, 23]: A single token can "jump" to any vertex (including non-adjacent one).

- *Token Addition and Removal* (TAR rule) [1, 3, 14, 17, 19, 20, 23]: We can either add or remove a single token at a time if it results in an independent set of cardinality at least a given threshold.

As the (ordinary) INDEPENDENT SET problem plays an important role in computational complexity theory, the ISRECONF problem is also one of the most well-studied reconfiguration problems. ISRECONF is PSPACE-complete under any of the three reconfiguration rules for general graphs [14], for planar graphs [4, 11, 12], for perfect graphs [17], and even for bounded bandwidth graphs [23]. Recall that a decision problem (a problem which has as answer either YES or NO) is in PSPACE, or *can be solved in polynomial space*, if there exists an algorithm that solves the problem using an amount of memory that is polynomial in the size of the input, and the *complete* problems are the "most difficult" problems in their complexity class.

In computational complexity theory, SLIDING TOKEN problem, or ISRECONF problem under TS rule, plays an important role since several PSPACE-hardness results have been proved using reduction from it. Suppose that we are given two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a graph $G = (V, E)$ such that $|\mathbf{I}_b| = |\mathbf{I}_r|$, and imagine that a token (coin) is placed on each vertex in $\mathbf{I}_b$. The SLIDING TOKEN problem asks whether there exists a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of G such that:

(a) $\mathbf{I}_1 = \mathbf{I}_b$, $\mathbf{I}_\ell = \mathbf{I}_r$, and $|\mathbf{I}_i| = |\mathbf{I}_b| = |\mathbf{I}_r|$ for all i, $1 \leq i \leq \ell$; and

(b) for each i, $2 \leq i \leq \ell$, there is an edge $\{u, v\}$ in G such that $\mathbf{I}_{i-1} \setminus \mathbf{I}_i = \{u\}$ and $\mathbf{I}_i \setminus \mathbf{I}_{i-1} = \{v\}$, that is, $\mathbf{I}_i$ can be obtained from $\mathbf{I}_{i-1}$ by sliding exactly one token on a vertex $u \in \mathbf{I}_{i-1}$ to its adjacent vertex v along $\{u, v\} \in E$.

Such a sequence is called a *reconfiguration sequence* between $\mathbf{I}_b$ and $\mathbf{I}_r$. Without loss of generality, one can assume that G is simple and connected. Note that the tokens are unlabelled, while the vertices in a graph are labelled. We sometimes omit to say the vertex on which a token is placed, and simply say a token in an independent set $\mathbf{I}$. The PSPACE-hardness implies that an instance of SLIDING TOKEN may require an exponential number of token-slides even in a minimum-length reconfiguration sequence. In such a case, tokens should make "detours" to avoid violating to be independent. Figure 1.1 illustrates the reconfiguration sequence which transforms $\mathbf{I}_b = \mathbf{I}_1$ into $\mathbf{I}_r = \mathbf{I}_5$ using TS rule where the token on vertex w has to make detour to ensure that all intermediate sets $\mathbf{I}_2, \mathbf{I}_3, \mathbf{I}_4$ are independent.

[1]By saying "an intermediate set of tokens is independent", we actually mean that "the set of vertices where tokens are placed is independent".

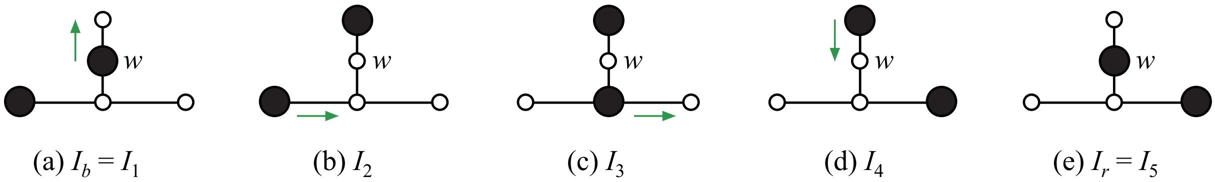

Figure 1.1: Transform $\mathbf{I}_b = I_1$ to $\mathbf{I}_r = I_5$ using TS rule. The tokens are marked by large black circles.

In this thesis, we show that SLIDING TOKEN problem for trees can be solved in linear time. Recently, Kamiński et al. [17] gave a linear-time algorithm to solve SLIDING TOKEN problem for cographs (also known as P_4-free graphs). They also showed that for any YES-instance on cographs, there exists a reconfiguration sequence from $\mathbf{I}_b$ to $\mathbf{I}_r$ such that no token makes detour. Bonsma et al. [5] proved that SLIDING TOKEN can be solved in polynomial time for claw-free graphs. Note that neither cographs nor claw-free graphs contain trees as a subclass. Also, Kamiński et al. [17] gave a linear-time algorithm for solving ISRECONF for even-hole-free graphs (which include trees) under TJ and TAR rules. Indeed, the answer is always YES under the two rules when restricted to even-hole-free graphs (as long as two given independent sets have the same cardinality for the TJ rule.) Furthermore, tokens never make detours in even-hole-free graphs under the TJ and TAR rules. On the other hand, under the TS rule, tokens are required to make detours even in trees (See Figure 1.1). In addition, there are NO-instances for trees under TS rule (See Figure 1.2). These make the problem much more complicated, and we think they are the main reasons why SLIDING TOKEN for trees was open.

Figure 1.2: A YES-instance for ISRECONF under the TJ rule, which is a NO-instance for the SLIDING TOKEN problem.

In the next chapters, we are going to present the followings:

- **Chapter 2: Preliminaries**: In this chapter, we introduce some concepts and notation which will be used to present our algorithm.
- **Chapter 3: Sliding Tokens on Trees**: In this chapter, we present a polynomial-time algorithm for solving SLIDING TOKEN problem for trees [8]. We also show that its running time can be improved to linear-time [7], which implies that the ISRECONF problem for trees can be solved in linear time under any of three reconfiguration rules.

In the remainder of this chapter, we briefly explain our idea for solving SLIDING TOKEN for trees. Let T be a tree and let $\mathbf{I}$ be an independent set of T. Imagine that a token is placed at each vertex of $\mathbf{I}$. Intuitively, we say that a token in vertex v is "rigid" if it cannot be slid at all. More precisely, $v \in \mathbf{I}'$ for any independent set $\mathbf{I}'$ which can be reconfigured from $\mathbf{I}$. Our algorithm is based on the following claims:

1. Given an independent set $\mathbf{I}$ of T, one can find all rigid tokens of $\mathbf{I}$ in linear time. If two sets $\mathbf{I}_b$ and $\mathbf{I}_r$ have different placements of rigid tokens, then it is a NO-instance.

2. Otherwise, we obtain a forest by deleting all rigid tokens and its neighbors. If for each tree in this forest, the number of tokens in $\mathbf{I}_b$ and $\mathbf{I}_r$ are the same, then it is a YES-instance. Otherwise, it is a NO-instance.

Chapter 2

Preliminaries

In this chapter, we introduce some basic definitions and notation. The contents of this chapter are referenced from the original paper [8]. For more details on graph concepts, refer to some textbooks such as West [22], Diestel [9], etc.

For SLIDING TOKEN problem, we can assume without loss of generality that graphs are simple and connected. We now define some commonly used graph notation.

Notation 2.1. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$.

For a vertex $v \in V(G)$, let $N(G,v) = \{w \in V(G) \mid \{v,w\} \in E(G)\}$ and $N[G,v] = N(G,v) \cup \{v\}$.

Similarly, for an arbitrary subset $S \subseteq V(G)$, we write $N[G,S] = \bigcup_{v \in S} N[G,v]$.

For a subgraph G' of G, denote by $G \setminus G'$ the subgraph of G induced by $V(G) \setminus V(G')$.

Notation 2.2. Let T be a tree.

Denote by $\mathsf{dist}(v,w)$ the length of the unique (shortest) path between v and w in T. We call it the *distance* between v and w. The path between v and w is simply called the *vw-path*.

For two vertices u and v of a tree T, let T^u_v be the subtree of T obtained by regarding u as the root of T and then taking the subtree rooted at v which consists of v and all descendants of v. (See Figure 2.1) It should be noted that u is not contained in the subtree T^u_v.

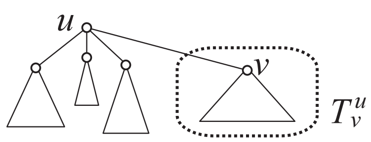

Figure 2.1: Subtree T^u_v in the whole tree T.

We define some concepts and notation which will be used for solving SLIDING TOKEN problem for trees.

Notation 2.3. Let $\mathbf{I}$ and $\mathbf{I}'$ be two independent sets of a graph G such that $|\mathbf{I}| = |\mathbf{I}'|$. If there exists exactly one edge $\{u,v\}$ in G such that $\mathbf{I} \setminus \mathbf{I}' = \{u\}$ and $\mathbf{I}' \setminus \mathbf{I} = \{v\}$, then we say that $\mathbf{I}'$ can be obtained from $\mathbf{I}$ by *sliding* the token on $u \in \mathbf{I}$ to its adjacent vertex v along the edge $\{u, v\}$, and denote it by $\mathbf{I} \leftrightarrow \mathbf{I}'$, or sometimes by $\mathbf{I} \overset{G}{\leftrightarrow} \mathbf{I}'$. Note that "sliding a token" can be reversed, i.e. $\mathbf{I} \leftrightarrow \mathbf{I}'$ if and only if $\mathbf{I}' \leftrightarrow \mathbf{I}$.

Definition 2.1. A *reconfiguration sequence* between two independent sets $\mathbf{I}_1$ and $\mathbf{I}_\ell$ of G is a sequence $\langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ of independent sets of G such that $\mathbf{I}_{i-1} \leftrightarrow \mathbf{I}_i$ for $i = 2, 3, \ldots, \ell$. We sometimes write $\mathbf{I} \in \mathcal{S}$ if an independent set $\mathbf{I}$ of G appears in the reconfiguration sequence $\mathcal{S}$. We write $\mathbf{I}_1 \overset{G}{\leftrightsquigarrow} \mathbf{I}_\ell$ if there exists a reconfiguration sequence $\mathcal{S}$ between $\mathbf{I}_1$ and $\mathbf{I}_\ell$ such that all independent sets $\mathbf{I} \in \mathcal{S}$ satisfy $\mathbf{I} \subseteq V(G)$. Sometimes, to emphasize the existence of a reconfiguration sequence, we also write $\mathbf{I}_1 \underset{\mathcal{S}}{\overset{G}{\leftrightsquigarrow}} \mathbf{I}_\ell$. Moreover, a reconfiguration sequence is *reversible*, i.e. $\mathbf{I}_1 \overset{G}{\leftrightsquigarrow} \mathbf{I}_\ell$ if and only if $\mathbf{I}_\ell \overset{G}{\leftrightsquigarrow} \mathbf{I}_1$. [1] The *length* of a reconfiguration sequence $\mathcal{S}$ is defined as the number of independent sets contained in $\mathcal{S}$. For example, the length of the reconfiguration sequence in Figure 1.1 is 5.

Definition 2.2. We say that a degree-1 vertex v of T is *safe* if its unique neighbor u has at most one neighbor w of degree more than one. (See Figure 2.2.) Note that any tree has at least one safe degree-1 vertex.

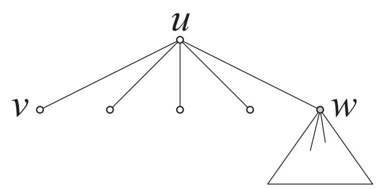

Figure 2.2: A degree-1 vertex v of a tree T which is safe.

[1]This is clear because $\mathbf{I} \overset{G}{\leftrightarrow} \mathbf{I}'$ if and only if $\mathbf{I}' \overset{G}{\leftrightarrow} \mathbf{I}$.

Chapter 3

Sliding Tokens on Trees

In this chapter, we present the main result of this thesis. We present our polynomial-time algorithm for solving SLIDING TOKEN problem for trees [8]. We also show that our algorithm can be improved to execute in linear time [7]. Given two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a graph G, the SLIDING TOKEN problem asks whether $\mathbf{I}_b \overset{G}{\leftrightsquigarrow} \mathbf{I}_r$ or not. We may assume without loss of generality that $|\mathbf{I}_b| = |\mathbf{I}_r|$; otherwise the answer is clearly NO. Note that SLIDING TOKEN is a decision problem asking for the existence of a reconfiguration sequence between $\mathbf{I}_b$ and $\mathbf{I}_r$, and hence it does not ask an actual reconfiguration sequence. We always denote by $\mathbf{I}_b$ and $\mathbf{I}_r$ the *initial* and *target* independent sets of G, respectively.

Theorem 3.1. *The* SLIDING TOKEN *problem can be solved in polynomial time for trees.*

To prove this theorem, we simply describe an algorithm which solves the SLIDING TOKEN problem for tree in polynomial time. The concept of "rigid tokens" is the key concept for our algorithm.

Definition 3.1. Let T be a tree and let $\mathbf{I}$ is an independent set of T. A token $v \in \mathbf{I}$ is $(T, \mathbf{I})$-*rigid* if $v \in \mathbf{I}'$ for any independent set $\mathbf{I}'$ of T such that $\mathbf{I} \overset{T}{\leftrightsquigarrow} \mathbf{I}'$. If $v \in \mathbf{I}$ is not $(T, \mathbf{I})$-rigid, then it is $(T, \mathbf{I})$-*movable*, in other words, there exists an independent set $\mathbf{I}'$ such that $\mathbf{I} \overset{T}{\leftrightsquigarrow} \mathbf{I}'$ but $v \notin \mathbf{I}'$.

The concept of rigid/movable tokens can be extended to subtrees of T. Let T' be a subtree of T. Let $\mathbf{I} \cap T'$ denote the set $\mathbf{I} \cap V(T')$. A token $v \in \mathbf{I} \cap T'$ is $(T', \mathbf{I} \cap T')$-*rigid* if $v \in \mathbf{J}$ for any independent set $\mathbf{J}$ of T' such that $\mathbf{I} \cap T' \overset{T'}{\leftrightsquigarrow} \mathbf{J}$. Note that, since independent sets are restricted only to the subtree T', we cannot use any vertex (and hence any edge) in $T \setminus T'$ during the reconfiguration. Furthermore, the vertex-subset $\mathbf{J} \cup \big(\mathbf{I} \cap (T \setminus T')\big)$ does not necessarily form an independent set of the whole tree T.

For example, in Figure 3.1, the tokens t_1, t_2, t_3, t_4 are $(T, \mathbf{I})$-rigid, while the tokens t_5, t_6, t_7 are $(T, \mathbf{I})$-movable. On the other hand, tokens t_6 and t_7 are $(T', \mathbf{I} \cap T')$-rigid even though they are $(T, \mathbf{I})$-movable in the whole tree T.

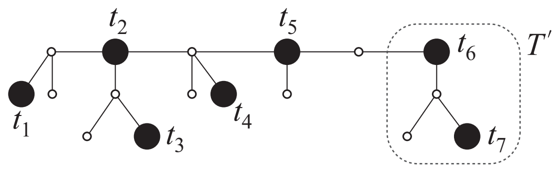

Figure 3.1: An independent set $\mathbf{I}$ of a tree T, where t_1, t_2, t_3, t_4 are $(T, \mathbf{I})$-rigid tokens and t_5, t_6, t_7 are $(T, \mathbf{I})$-movable tokens. For the subtree T', tokens t_6, t_7 are $(T', \mathbf{I} \cap T')$-rigid.

Note that, even though t_6 and t_7 cannot be slid to any neighbor in $\mathbf{I}$, we can slide them after sliding t_5 downward. Rigid tokens have the following important recursive characterization.

Lemma 3.1. *Let $\mathbf{I}$ be an independent set of a tree T, and let u be a vertex in $\mathbf{I}$.*

- *(a) Suppose that $|V(T)| = |\{u\}| = 1$. Then, the token on u is $(T, \mathbf{I})$-rigid.*
- *(b) Suppose that $|V(T)| \geq 2$. Then, a token on u is $(T, \mathbf{I})$-rigid if and only if, for all neighbors $v \in N(T, u)$, there exists a vertex $w \in \mathbf{I} \cap N(T^u_v, v)$ such that the token on w is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid.*

Proof. Part (a) is trivial by definition of rigid tokens. Assume that $|V(T)| \geq 2$, we show part (b).

We first show the if-part of (b). Suppose that

$$\forall v \in N(T, u)\ \exists w \in \mathbf{I} \cap N(T^u_v, v) \text{ s.t the token on } w \text{ is } (T^v_w, \mathbf{I} \cap T^v_w)\text{-rigid.} \tag{3.1}$$

We want to show that the token t on u is $(T, \mathbf{I})$-rigid. Suppose for a contradiction that t is $(T, \mathbf{I})$-movable, which means that t can be slid to a vertex $v \in N(T, u)$. By assumption 3.1, in order to slide t to v, we first need to slide the token t' on w to one of its neighbors other than v. But this contradicts the assumption that t' is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid. Hence, u is $(T, \mathbf{I})$-rigid.

(a) (b)

Figure 3.2: (a) A $(T, \mathbf{I})$-rigid token on u, and (b) a $(T, \mathbf{I})$-movable token on u.

Next, we show the only-if-part of (b). Suppose that u is $(T, \mathbf{I})$-rigid, we want to show that for all neighbors $v \in N(T, u)$, there exists a vertex $w \in \mathbf{I} \cap N(T^u_v, v)$ such that the token on w is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid. We will prove the contrapositive that if either

$$\exists v \in N(T, u)\ \mathbf{I} \cap N(T^u_v, v) = \emptyset \tag{3.2}$$

or

$$\exists v \in N(T, u)\ \forall w \in \mathbf{I} \cap N(T^u_v, v) \text{ the token on } w \text{ is } (T^v_w, \mathbf{I} \cap T^v_w)\text{-movable.} \tag{3.3}$$

then u is $(T, \mathbf{I})$-movable. Assumption (3.2) is trivial since u can be directly slid to v. We now consider assumption (3.3). For each $w \in \mathbf{I} \cap N(T^u_v, v)$, there exists a reconfiguration sequence $\mathcal{S}_w$ such that $\mathbf{I} \cap T^v_w \overset{T^v_w}{\underset{\mathcal{S}_w}{\leftrightsquigarrow}} \mathbf{J}$ where $\mathbf{J} \subseteq V(T^v_w)$ is independent and $w \notin \mathbf{J}$. Since $v \notin \mathbf{I}$ is the only vertex not in $V(T^v_w)$ and adjacent to a vertex in $V(T^v_w)$, for any independent set $\mathbf{J}' \in \mathcal{S}_w$, $\mathbf{J}' \cup (\mathbf{I} \setminus V(T^v_w))$ is independent. In other words, the reconfiguration sequence $\mathcal{S}_w$ on T^v_w can be extended to the whole tree T, which implies that, for each w, the token on w is $(T, \mathbf{I})$-movable and can be slid to one of w's neighbors other than v. Hence, the token on u can finally be slid to v, which means that u is $(T, \mathbf{I})$-movable. $\square$

Lemma 3.1 implies that we can check whether one token in an independent set $\mathbf{I}$ of a tree T is $(T, \mathbf{I})$-rigid or not in linear time.

Lemma 3.2. *Given a tree T with n vertices, an independent set $\mathbf{I}$ of T, and a vertex $u \in \mathbf{I}$, it can be decided in $O(n)$ time whether the token on u is $(T, \mathbf{I})$-rigid.*

Proof. We regard T as a rooted tree with the root u, and compute a $\{0, 1\}$-parity $\phi(v)$ for each vertex $v \in V(T)$ from the leaves of T to the root u, as follows.

- For each leaf v of T, we set $\phi(v) = 1$ if $v \in \mathbf{I}$, otherwise $\phi(v) = 0$.
- For each internal vertex v of T such that $v \notin \mathbf{I}$, we set $\phi(v) = 1$ if there exists a child w of v such that $w \in \mathbf{I}$ and $\phi(w) = 1$; otherwise $\phi(v) = 0$.
- For each internal vertex v of T such that $v \in \mathbf{I}$, we set $\phi(v) = 1$ if $\phi(w) = 1$ hold for *all* children w of v; otherwise $\phi(v) = 0$. (Note that $w \notin \mathbf{I}$ for all children w of v since $v \in \mathbf{I}$.)

By Lemma 3.1 the token on u is $(T, \mathbf{I})$-rigid if and only if $\phi(u) = 1$. Clearly, the parity $\phi(u)$ for the root u can be computed in $O(n)$ time. $\square$

For an independent set $\mathbf{I}$ of T, let $\mathsf{R}(\mathbf{I})$ be the set of all vertices in $\mathbf{I}$ on which $(T, \mathbf{I})$-rigid tokens are placed. The following algorithm determines whether $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$ or not.

Algorithm 1 Algorithm for solving the SLIDING TOKEN problem on trees.

Input: Two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a tree T with n vertices.

Output: Return YES if $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$; otherwise return NO.

1: Compute $\mathsf{R}(\mathbf{I}_b)$ and $\mathsf{R}(\mathbf{I}_r)$ using Lemma 3.2. If $\mathsf{R}(\mathbf{I}_b) \neq \mathsf{R}(\mathbf{I}_r)$, then return NO; otherwise go to **Step 2**.

2: Delete the vertices in $N[T, \mathsf{R}(\mathbf{I}_b)] = N[T, \mathsf{R}(\mathbf{I}_r)]$ from T, and obtain a forest F consisting of q trees $T_1, T_2, \ldots, T_q$. If $|\mathbf{I}_b \cap T_j| = |\mathbf{I}_r \cap T_j|$ holds for every $j \in \{1, 2, \ldots, q\}$, then return YES; otherwise return NO.

By Lemma 3.2 we can determine whether one token in an independent set $\mathbf{I}$ of T is $(T, \mathbf{I})$-rigid or not in $O(n)$ time, and hence **Step 1** can be done in time $O(n) \times (|\mathbf{I}_b| + |\mathbf{I}_r|) = O(n^2)$. Clearly, **Step 2** can be done in $O(n)$ time. Therefore, Algorithm 1 runs in $O(n^2)$ time in total.

We next prove the correctness of Algorithm 1. The following lemma is useful for our proofs later.

Lemma 3.3. *Let $\mathbf{I}$ be an independent set of a tree T such that all tokens are $(T, \mathbf{I})$-movable, and let v be a vertex such that $v \notin \mathbf{I}$. Then, there exists at most one neighbor $w \in \mathbf{I} \cap N(T, v)$ such that the token on w is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid.*

Proof. Suppose that there are two neighbors $w, w' \in \mathbf{I} \cap N(T, v)$ such that the tokens on both w and w' are respectively $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid and $(T^v_{w'}, \mathbf{I} \cap T^v_{w'})$-rigid. We claim that the token t on w is $(T, \mathbf{I})$-rigid.

Suppose that t is $(T, \mathbf{I})$-movable. Since t is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid, the only way to move t is sliding it to v. But, to slide t to v, we need to slide the token t' on w' to a vertex of $T^v_{w'}$. This contradicts our assumption that w' is $(T^v_{w'}, \mathbf{I} \cap T^v_{w'})$-rigid. □

We show the correctness of **Step 1**.

Lemma 3.4. *Suppose that $\mathsf{R}(\mathbf{I}_b) \neq \mathsf{R}(\mathbf{I}_r)$ for two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a tree T. Then, it is a* NO*-instance.*

Proof. We prove this lemma by contrapositive. Recall that if the token on $v \in \mathbf{I}$ is $(T, \mathbf{I})$-rigid then $v \in \mathbf{I}'$ for any $\mathbf{I}'$ such that $\mathbf{I} \overset{T}{\leftrightsquigarrow} \mathbf{I}'$. It follows that if $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$ then $\mathsf{R}(\mathbf{I}_b) = \mathsf{R}(\mathbf{I}_r)$. □

We then show the correctness of **Step 2**. First of all, we show that deleting the vertices with rigid tokens together with their neighbors does not affect the reconfigurability.

Lemma 3.5. *Suppose that $\mathsf{R}(\mathbf{I}_b) = \mathsf{R}(\mathbf{I}_r)$ for two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a tree T, and let F be the forest obtained by deleting the vertices in $N[T, \mathsf{R}(\mathbf{I}_b)] = N[T, \mathsf{R}(\mathbf{I}_r)]$ from T. Then, $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$ if and only if $\mathbf{I}_b \cap F \overset{F}{\leftrightsquigarrow} \mathbf{I}_r \cap F$. Furthermore, all tokens in $\mathbf{I}_b \cap F$ are $(F, \mathbf{I}_b \cap F)$-movable, and all tokens in $\mathbf{I}_r \cap F$ are $(F, \mathbf{I}_r \cap F)$-movable.*

Proof. Before proving the above lemma, observe that since F is obtained by deleting the vertices in $N[T, \mathsf{R}(\mathbf{I}_b)] = N[T, \mathsf{R}(\mathbf{I}_r)]$ from T, we have $\mathbf{I}_b \cap F = \mathbf{I}_b \setminus \mathsf{R}(\mathbf{I}_b)$ and $\mathbf{I}_r \cap F = \mathbf{I}_r \setminus \mathsf{R}(\mathbf{I}_r)$.

We first show the only-if-part of this lemma. Suppose that there exists a reconfiguration sequence $\mathcal{S} = \langle \mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_\ell \rangle$ ($\mathbf{I}_1 = \mathbf{I}_b$, $\mathbf{I}_\ell = \mathbf{I}_r$) such that $\mathbf{I}_b \underset{\mathcal{S}}{\overset{T}{\leftrightsquigarrow}} \mathbf{I}_r$. We want to show that there exists a reconfiguration sequence $\mathcal{S}'$ such that $\mathbf{I}_b \cap F \underset{\mathcal{S}'}{\overset{F}{\leftrightsquigarrow}} \mathbf{I}_r \cap F$. Indeed $\mathcal{S}' = \langle \mathbf{I}_1 \cap F, \mathbf{I}_2 \cap F, \ldots, \mathbf{I}_\ell \cap F \rangle$. To see this, note that:

- For each i ($1 \leq i \leq \ell$), $\mathbf{I}_i \in \mathcal{S}$ is independent, then $\mathbf{I}_i \cap F \in \mathcal{S}'$ is also independent.

- For two consecutive independent sets $\mathbf{I}_{i-1}$ and $\mathbf{I}_i$ in $\mathcal{S}$, let $\mathbf{I}_{i-1} \setminus \mathbf{I}_i = \{u\}$ and $\mathbf{I}_i \setminus \mathbf{I}_{i-1} = \{v\}$, i.e. $\mathbf{I}_{i-1} \overset{T}{\leftrightarrow} \mathbf{I}_i$. Since $u \notin \mathbf{I}_i$ and $v \notin \mathbf{I}_{i-1}$, neither u nor v are in $\mathsf{R}(\mathbf{I}_b) = \mathsf{R}(\mathbf{I}_r)$. Therefore, $u, v \in V(F)$, and hence $\{u, v\} \in E(F)$. It follows that $\mathbf{I}_{i-1} \cap F \overset{F}{\leftrightarrow} \mathbf{I}_i \cap F$.

In other words, $\mathcal{S}$ can be "restricted" to a reconfiguration sequence on F.

Next, we show the if-part of this lemma. Suppose that there exists a reconfiguration sequence $\mathcal{S}' = \langle \mathbf{J}_1, \mathbf{J}_2, \ldots, \mathbf{J}_k \rangle$ ($\mathbf{J}_1 = \mathbf{I}_b \cap F$, $\mathbf{J}_k = \mathbf{I}_r \cap F$) such that $\mathbf{I}_b \cap F \underset{\mathcal{S}'}{\overset{F}{\leftrightsquigarrow}} \mathbf{I}_r \cap F$. We want to show that there exists a reconfiguration sequence $\mathcal{S}$ such that $\mathbf{I}_b \underset{\mathcal{S}}{\overset{T}{\leftrightsquigarrow}} \mathbf{I}_r$. Indeed, $\mathcal{S} = \langle \mathbf{J}_1 \cup \mathsf{R}(\mathbf{I}_b), \mathbf{J}_2 \cup \mathsf{R}(\mathbf{I}_b), \ldots, \mathbf{J}_k \cup \mathsf{R}(\mathbf{I}_b) \rangle$. To see this, note that:

- Since F is obtained by deleting the vertices in $N[T, \mathsf{R}(\mathbf{I}_b)] = N[T, \mathsf{R}(\mathbf{I}_r)]$ from T, for every j ($1 \leq j \leq k$), $\mathbf{J}_j \cup \mathsf{R}(\mathbf{I}_b)$ is independent. Moreover, $\mathbf{J}_1 \cup \mathsf{R}(\mathbf{I}_b) = (\mathbf{I}_b \cap F) \cup \mathsf{R}(\mathbf{I}_b) = (\mathbf{I}_b \setminus \mathsf{R}(\mathbf{I}_b)) \cup \mathsf{R}(\mathbf{I}_b) = \mathbf{I}_b$. Similarly, $\mathbf{J}_k \cup \mathsf{R}(\mathbf{I}_b) = \mathbf{I}_r$.

- Since F is a subgraph of T, and note that rigid tokens can not be slid at all, it follows that if $\mathbf{J}_{j-1} \overset{F}{\leftrightarrow} \mathbf{J}_j$ then $\mathbf{J}_{j-1} \cup \mathsf{R}(\mathbf{I}_b) \overset{T}{\leftrightarrow} \mathbf{J}_j \cup \mathsf{R}(\mathbf{I}_b)$.

In other words, $\mathcal{S}'$ can be "extended" to a reconfiguration sequence on T.

We finally show that all tokens in $\mathbf{I}_b \cap F$ are $(F, \mathbf{I}_b \cap F)$-movable. A similar argument can be applied for $\mathbf{I}_r \cap F$. Note that each token t on a vertex v in $\mathbf{I}_b \cap F$ is $(T, \mathbf{I}_b)$-movable; otherwise $t \in \mathsf{R}(\mathbf{I}_b)$. Hence, there exists an independent set $\mathbf{I}'$ of T such that $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}'$ and $v \notin \mathbf{I}'$. As we have shown before, $\mathbf{I}_b \cap F \overset{F}{\leftrightsquigarrow} \mathbf{I}' \cap F$. Therefore, t is $(F, \mathbf{I}_b \cap F)$-movable. $\square$

Suppose that $\mathsf{R}(\mathbf{I}_b) = \mathsf{R}(\mathbf{I}_r)$ for two given independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of a tree T. Let F be the forest consisting of q trees $T_1, T_2, \ldots, T_q$, which is obtained from T by deleting the vertices in $N[T, \mathsf{R}(\mathbf{I}_b)] = N[T, \mathsf{R}(\mathbf{I}_r)]$. Since we can slide a token only along an edge of F, we clearly have $\mathbf{I}_b \cap F \overset{F}{\leftrightsquigarrow} \mathbf{I}_r \cap F$ if and only if $\mathbf{I}_b \cap T_j \overset{T_j}{\leftrightsquigarrow} \mathbf{I}_r \cap T_j$ for all $j \in \{1, 2, \ldots, q\}$. Furthermore, Lemma 3.5 implies that, for each $j \in \{1, 2, \ldots, q\}$, all tokens in $\mathbf{I}_b \cap T_j$ are $(T_j, \mathbf{I}_b \cap T_j)$-movable; similarly, all tokens in $\mathbf{I}_r \cap T_j$ are $(T_j, \mathbf{I}_r \cap T_j)$-movable.

We now complete our proof of the correctness of **Step 2** by showing that if there are no rigid tokens then $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$ if and only if $|\mathbf{I}_b| = |\mathbf{I}_r|$.

Lemma 3.6. *Let $\mathbf{I}_b$ and $\mathbf{I}_r$ be two independent sets of a tree T such that all tokens in $\mathbf{I}_b$ and $\mathbf{I}_r$ are $(T, \mathbf{I}_b)$-movable and $(T, \mathbf{I}_r)$-movable, respectively. Then, $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$ if and only if $|\mathbf{I}_b| = |\mathbf{I}_r|$.*

Before proving this lemma, we give some useful properties of a *safe* degree-1 vertex (see Definition 2.2). We claim that if v is a safe degree-1 vertex, then one of the closest tokens from v can be slid to v. Obviously, if a token is placed on v (i.e. $v \in \mathbf{I}$) then no extra sliding is needed.

Lemma 3.7. *Let* $\mathbf{I}$ *be an independent set of a tree* T *such that all tokens in* $\mathbf{I}$ *are* $(T, \mathbf{I})$*-movable, and let* $v \notin \mathbf{I}$ *be a safe degree-*1 *vertex of* T*. Then, there exists an independent set* $\mathbf{I}'$ *such that* $\mathbf{I}' \setminus \mathbf{I} = \{v\}$ *and* $\mathbf{I} \overset{T}{\leftrightsquigarrow} \mathbf{I}'$. [1]

Proof. Let $M = \{w \in \mathbf{I} \mid \mathsf{dist}(v, w) = \min_{x \in \mathbf{I}} \mathsf{dist}(v, x)\}$. Let w be an arbitrary vertex in M, and let $P = (p_0 = v, p_1, \ldots, p_\ell = w)$ be the unique path between v and w in T. (See Figure 3.3.)

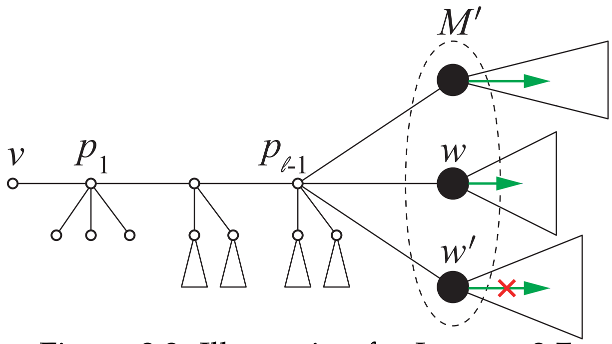

Figure 3.3: Illustration for Lemma 3.7.

If $\ell = 1$ and hence $p_1 \in \mathbf{I}$, then we can simply slide the token on p_1 to v.

We now consider the case $\ell \geq 2$. Since the token on w is closest to v, no token can be placed on the vertices $p_0, \ldots, p_{\ell-1}$ and the neighbors of $p_0, \ldots, p_{\ell-2}$. Let $M' = M \cap N(T, p_{\ell-1})$. Since $p_{\ell-1} \notin \mathbf{I}$, by Lemma 3.3 there is at most one vertex $w' \in M'$ such that the token on w' is $(T_{w'}^{p_{\ell-1}}, \mathbf{I} \cap T_{w'}^{p_{\ell-1}})$-rigid. We choose such a vertex w' if exists, otherwise choose an arbitrary vertex in M' and regard it as w'.

Before sliding the token on w' to v, we need to slide all tokens on the vertices w'' in $M' \setminus \{w'\}$ first. Since all tokens on the vertices w'' in $M' \setminus \{w'\}$ are $(T_{w''}^{p_{\ell-1}}, \mathbf{I} \cap T_{w''}^{p_{\ell-1}})$-movable, we can slide the tokens on w'' to some vertices in $T_{w''}^{p_{\ell-1}}$. Now, we can slide the token on w' to v along the path P. Finally, we reverse all the steps of sliding tokens on the vertices w'' above in order to get all tokens except the one in w' back to their original positions. In this way, we obtain an independent set $\mathbf{I}'$ such that $\mathbf{I}' \setminus \mathbf{I} = \{v\}$ and $\mathbf{I} \overset{T}{\leftrightsquigarrow} \mathbf{I}'$. □

We then prove that deleting a safe degree-1 vertex with a token does not affect the movability of the other tokens.

Lemma 3.8. *Let* v *be a safe degree-*1 *vertex of a tree* T*, and let* $\bar{T}$ *be the subtree of* T *obtained by deleting* v*, its unique neighbor* u*, and the resulting isolated vertices. Let* $\mathbf{I}$ *be an independent set of* T *such that* $v \in \mathbf{I}$ *and all tokens are* $(T, \mathbf{I})$*-movable. Then, all tokens in* $\mathbf{I} \setminus \{v\}$ *are* $(\bar{T}, \mathbf{I} \setminus \{v\})$*-movable.*

Proof (quoted from [8]). Since T_v^u consists of a single vertex v, the token on v is $(T_v^u, \mathbf{I} \cap T_v^u)$-rigid. Therefore, no token is placed on degree-1 neighbors of u other than v (see

[1] For any $v' \in \mathbf{I}'$, $v' \neq v$, we have $v' \in \mathbf{I}$

Figure 3.4), because otherwise it contradicts to Lemma 3.3; recall that all tokens in $\mathbf{I}$ are assumed to be $(T, \mathbf{I})$-movable.

Let $\bar{I} = \mathbf{I} \setminus \{v\}$. Suppose for a contradiction that there exists a token in $\bar{I}$ which is $(\bar{T}, \bar{I})$-rigid. Let $w_p \in \bar{I}$ be such a vertex closest to v, and let z be the vertex on the vw_p-path right before w_p.

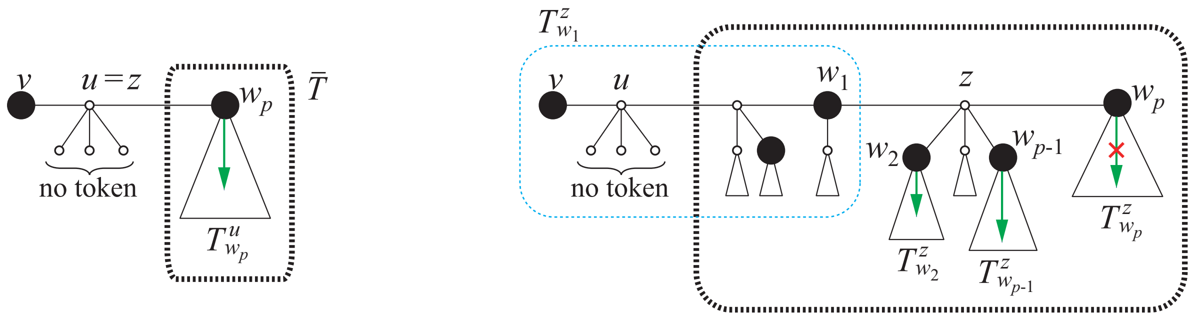

Figure 3.4: Illustration for Lemma 3.8.

Case (1): $z = u$. (See Figure 3.4(a).)

Recall that the token on v is $(T, \mathbf{I})$-movable, but is $(T^u_v, \mathbf{I} \cap T^u_v)$-rigid. Therefore, by Lemma 3.3 the token on w_p must be $(T^u_{w_p}, \mathbf{I} \cap T^u_{w_p})$-movable. However, this contradicts the assumption that w_p is $(\bar{T}, \bar{I})$-rigid, because $\bar{T} = T^u_{w_p}$ and $\bar{I} = \mathbf{I} \cap T^u_{w_p}$ in this case.

Case (2): $z \neq u$. (See Figure 3.4(b).)

Let w_1 be the neighbor of z on the vw_p-path other than w_p.
Let $N(T, z) = \{w_1, w_2, \dots, w_p\}$. We note that the subtree $T^z_{w_1}$ contains the deleted star $T \setminus \bar{T}$ centered at u, because only the neighbor w_1 of z is on the vz-path.

We first note that the token t_p on w_p is $(\bar{T}^z_{w_p}, \bar{I} \cap \bar{T}^z_{w_p})$-rigid, because otherwise t_p can be slid to some vertex in $\bar{T}^z_{w_p}$ and hence it is $(\bar{T}, \bar{I})$-movable. Since $\bar{T}^z_{w_p} = T^z_{w_p}$ and $\bar{I} \cap \bar{T}^z_{w_p} = \mathbf{I} \cap T^z_{w_p}$, the token t_p is also $(T^z_{w_p}, \mathbf{I} \cap T^z_{w_p})$-rigid.

For each $j \in \{2, 3, \dots, p-1\}$ with $w_j \in \mathbf{I}$, since t_p is $(T^z_{w_p}, \mathbf{I} \cap T^z_{w_p})$-rigid, by Lemma 3.3 each token t_j on w_j is $(T^z_{w_j}, \mathbf{I} \cap T^z_{w_j})$-movable. Then, since $T^z_{w_j} = \bar{T}^z_{w_j}$ and $\mathbf{I} \cap T^z_{w_j} = \bar{I} \cap \bar{T}^z_{w_j}$, the token t_j is $(\bar{T}^z_{w_j}, \bar{I} \cap \bar{T}^z_{w_j})$-movable. Therefore, if $w_1 \notin \bar{I}$ or the token t_1 on w_1 is $(\bar{T}^z_{w_1}, \bar{I} \cap \bar{T}^z_{w_1})$-movable, then we can slide t_p from w_p to z after sliding each token t_j in $\bar{I} \cap \{w_1, w_2, \dots, w_{p-1}\}$ to some vertex of the subtree $\bar{T}^z_{w_j}$. This contradicts the assumption that t_p is $(\bar{T}, \bar{I})$-rigid.

Therefore, we have $w_1 \in \bar{I}$ and a token t_1 on w_1 is $(\bar{T}^z_{w_1}, \bar{I} \cap \bar{T}^z_{w_1})$-rigid. However, since t_p is $(\bar{T}^z_{w_p}, \bar{I} \cap \bar{T}^z_{w_p})$-rigid, this implies that t_1 is $(\bar{T}, \bar{I})$-rigid. Since w_1 is on the vw_p-path in T, this contradicts the assumption that t_p is the $(\bar{T}, \bar{I})$-rigid token closest to v. $\square$

We are now ready to show the proof of Lemma 3.6.

Proof of Lemma 3.6. The only-if-part of this lemma is trivial. We now prove the if-part of this lemma.

Suppose that $|\mathbf{I}_b| = |\mathbf{I}_r|$. We claim that there is an independent set $\mathbf{I}^*$ such that $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$ and $\mathbf{I}_r \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$. Since a reconfiguration sequence is reversible, $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$ and $\mathbf{I}_r \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$ imply that $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}_r$. The following algorithm constructs such a set $\mathbf{I}^*$ described above.

Algorithm 2 Algorithm for constructing $\mathbf{I}^*$

Input: Two independent sets $\mathbf{I}_b$ and $\mathbf{I}_r$ of T; $|\mathbf{I}_b| = |\mathbf{I}_r|$.

Output: An independent set $\mathbf{I}^*$ such that $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$ and $\mathbf{I}_r \overset{T}{\leftrightsquigarrow} \mathbf{I}^*$.

1: $\mathbf{I}^* = \emptyset$.
2: **while** $|\mathbf{I}_b| = |\mathbf{I}_r| \neq 0$ **do**
3: Let v be a safe degree-1 vertex of T.
4: $\mathbf{I}^* \leftarrow \mathbf{I}^* \cup \{v\}$.
5: If $v \in \mathbf{I}_b$, let $\mathbf{I}'_b = \mathbf{I}_b$; otherwise let $\mathbf{I}'_b$ be such that $\mathbf{I}'_b \setminus \mathbf{I}_b = \{v\}$ and $\mathbf{I}_b \overset{T}{\leftrightsquigarrow} \mathbf{I}'_b$.
6: If $v \in \mathbf{I}_r$, let $\mathbf{I}'_r = \mathbf{I}_r$; otherwise let $\mathbf{I}'_r$ be such that $\mathbf{I}'_r \setminus \mathbf{I}_r = \{v\}$ and $\mathbf{I}_r \overset{T}{\leftrightsquigarrow} \mathbf{I}'_r$.
7: $\mathbf{I}_b \leftarrow \mathbf{I}'_b \setminus \{v\}$; $\mathbf{I}_r \leftarrow \mathbf{I}'_r \setminus \{v\}$.
8: Let T' be the tree obtained by deleting v, its unique neighbor u, and the resulting isolated vertices.
9: $T \leftarrow T'$.
10: **end while**
11: Return $\mathbf{I}^*$.

The correctness of lines 5 and 6 are followed from Lemma 3.7. Lemma 3.8 claims the correctness of lines 7, 8 and 9, which means that deleting a safe degree-1 vertex with a token does not affect the movability of the other tokens. Line 7 indicates that Algorithm 2 will finally stop. It is clear that in each loop of Algorithm 2, what we do is showing that a token from $\mathbf{I}_b$ (and $\mathbf{I}_r$) can be slid to a vertex $v \in \mathbf{I}^*$. Also, lines 4, 8 and 9 indicate that $\mathbf{I}^*$ is indeed independent. In line 9, we replace T by the subtree T'; so we need to ensure that a reconfiguration sequence in T' can be extended to a reconfiguration sequence in T. Indeed, this follows from the fact that the unique neigbor u of v are not in $V(T')$ and $u \notin \mathbf{I}'_b \cup \mathbf{I}'_r$.

Put everything together, the correctness of Algorithm 2 is now clear. □

We have shown that SLIDING TOKEN problem can be solved in polynomial time. In Algorithm 1, **Step 1** is the only step that takes $O(n^2)$ time. Indeed, **Step 1** can be improved to execute in linear time. To clarify this statement, we first give the following property of rigid tokens on a tree, which says that deleting movable tokens does not affect the rigidity of the other tokens.

Lemma 3.9. *Let* $\mathbf{I}$ *be an independent set of a tree* T*. Assume that the token on a vertex* $x \in \mathbf{I}$ *is* $(T, \mathbf{I})$*-movable. Then, for every vertex* $u \in I \setminus \{x\}$*, the token on* u *is* $(T, \mathbf{I})$*-rigid if and only if it is* $(T, \mathbf{I} \setminus \{x\})$*-rigid.*

Proof (quoted from [7]). The if-part is trivially true, because we cannot make a rigid token movable by adding another token. We thus show the only-if-part by contradiction.

Let $\mathbf{I}' = \mathbf{I} \setminus \{x\}$. Suppose that $u \in \mathbf{I}$ is the closest vertex to x such that its token is $(T, \mathbf{I})$-rigid but $(T, \mathbf{I}')$-movable. We assume that x is contained in a subtree T^u_v for a neighbor v of u. (See Figure 3.5.) Note that $x \neq v$ since $x, u \in \mathbf{I}$. Since the token t_u on u is $(T, \mathbf{I})$-rigid, by Lemma 3.1 the vertex $v \in N(T, u)$ has at least one neighbor $w \in \mathbf{I} \cap N(T^u_v, v)$ such that the token t_w on w is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid. Indeed, t_w is $(T, \mathbf{I})$-rigid, because t_u is assumed to be $(T, \mathbf{I})$-rigid. Thus, we know that $x \neq w$ since the token t_x on x is $(T, \mathbf{I})$-movable.

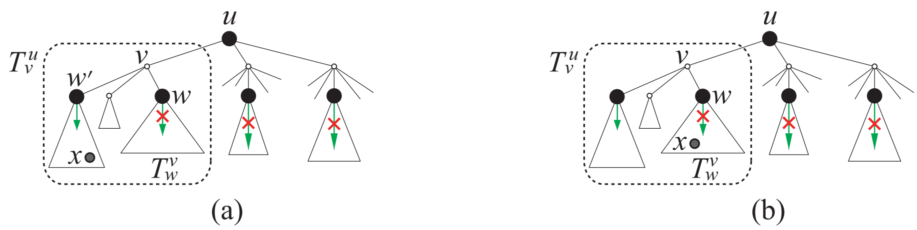

Figure 3.5: Illustration for Lemma 3.9.

First, consider the case where x is contained in a subtree $T^v_{w'}$ for some neighbor w' of v other than w. (See Figure 3.5(a).) Then, $\mathbf{I}' \cap T^v_w = \mathbf{I} \cap T^v_w$. Since t_w is $(T^v_w, \mathbf{I} \cap T^v_w)$-rigid, it is also $(T^v_w, \mathbf{I}' \cap T^v_w)$-rigid. Therefore, by Lemma 3.1 the token t_u is $(T, \mathbf{I}')$-rigid. This contradicts the assumption that t_u is $(T, \mathbf{I}')$-movable.

We thus consider the case where $x \in V(T^v_w) \setminus \{w\}$. (See Figure 3.5(b).) Recall that $\mathbf{I}'$ is obtained by deleting only x from $\mathbf{I}$. Then, since t_u is $(T, \mathbf{I})$-rigid but $(T, \mathbf{I}')$-movable, it must be slid from u to v. However, before executing this token-slide, we have to slide t_w to some vertex in $N(T^v_w, w)$. Thus, t_w is $(T^v_w, \mathbf{I}' \cap T^v_w)$-movable, and hence it is also $(T, \mathbf{I}')$-movable. Since t_w is $(T, \mathbf{I})$-rigid and w is strictly closer to $x \in V(T^v_w)$ than u, this contradicts the assumption that u is the closest vertex to x such that its token is $(T, \mathbf{I})$-rigid but $(T, \mathbf{I}')$-movable. □

Then, the following lemma proves that **Step 1** can be executed in $O(n)$ time, which then implies that SLIDING TOKEN for trees can be solved in linear time.

Lemma 3.10. *For an independent set* $\mathbf{I}$ *of a tree* T *with* n *vertices,* $\mathsf{R}(\mathbf{I})$ *can be computed in* $O(n)$ *time.*

Proof (quoted from [7]). Lemma 3.9 implies that the set $\mathsf{R}(\mathbf{I})$ of all $(T, \mathbf{I})$-rigid tokens in $\mathbf{I}$ can be found by removing all $(T, \mathbf{I})$-movable tokens in $\mathbf{I}$. Observe that, if $\mathbf{I}$ contains $(T, \mathbf{I})$-movable tokens, then at least one of them can be immediately slid to one of its neighbors. That is, there is a token on $u \in \mathbf{I}$ which has a neighbor $w \in N(T, u)$ such that $N(T, w) \cap \mathbf{I} = \{u\}$. Then, the following algorithm efficiently finds and removes such tokens iteratively.

Step A. Define and compute $\deg_{\mathbf{I}}(w) = |N(T, w) \cap \mathbf{I}|$ for all vertices $w \in V(T)$.

Step B. Define and compute $M = \{u \in \mathbf{I} \mid \exists w \in N(T,u) \text{ such that } \deg_{\mathbf{I}}(w) = 1\}$.

Step C. Repeat the following steps (i)–(iii) until $M = \emptyset$.

- **(i)** Select an arbitrary vertex $u \in M$, and remove it from M and $\mathbf{I}$.
- **(ii)** Update $\deg_{\mathbf{I}}(w) := \deg_{\mathbf{I}}(w) - 1$ for each neighbor $w \in N(T,u)$.
- **(iii)** If $\deg_{\mathbf{I}}(w)$ becomes one by the update (ii) above, then add the vertex $u' \in N(T,w) \cap \mathbf{I}$ into M.

Step D. Output $\mathbf{I}$ as the set $\mathsf{R}(\mathbf{I})$.

Clearly, Steps A, B and D can be done in $O(n)$ time. We now show that Step C takes only $O(n)$ time. Each vertex in $\mathbf{I}$ can be selected at most once as u at Step C-(i). For the selected vertex u, Step C-(ii) takes $O(\deg_T(u))$ time for updating $\deg_I(w)$ of its neighbors $w \in N(T,u)$. Each vertex in $V(T) \setminus \mathbf{I}$ can be selected at most once as w at Step C-(iii). For the selected vertex w, Step C-(iii) takes $O(\deg_T(w))$ time for finding $u' \in N(T,w) \cap \mathbf{I}$. Therefore, Step C takes $O\left(\sum_{v \in V(T)} \deg_T(v)\right) = O(n)$ time in total. $\square$

In summary, the following theorem is the main result of this chapter.

Theorem 3.2. *The* SLIDING TOKEN *problem can be solved in polynomial time for trees.*

Conclusion

In this thesis, we have shown that SLIDING TOKEN problem for trees can be solved in linear time [7] using a simple but non-trivial characterization of rigid tokens (Lemma 3.1). Indeed, we first presented our original quadratic-time algorithm (Theorem 3.1), and then showed our improvement to make it run in linear time (Lemmas 3.9 and 3.10). The author of this thesis is mainly contributed in proving Theorem 3.1 and the correctness of Algorithm 1.

The complexity status of SLIDING TOKEN remains open for chordal graphs and interval graphs. It is noted that these graphs have NO-instance such that all tokens are movable. (See Figure 3.6 for example.)

Figure 3.6: NO-instance for an interval graph such that all tokens are movable.

Interestingly, there is a subclass of chordal graphs called *block graphs* (also known as *completed Husimi trees*) which has a very similar structure to trees. Let $G = (V, E)$ be a graph. G is *connected* if any pair of vertices in G are joined by at least one path; otherwise, we say that G is *disconnected*. A vertex v of G is called a *cut vertex* if $G \setminus \{v\}$ is disconnected; otherwise, we say that v is a *non-cut vertex*. G is called a *clique* if for any $u, v \in V(G)$, $\{u, v\} \in E(G)$. If $|V(G)| = n$, we denote it by K_n and say that it is *a clique of size n*. A *block* of G is a maximal connected subgraph (i.e. a subgraph with as many edges as possible) with no cut vertex. G is called a *block graph* if G is connected and every block of G is a clique. Let $\mathbf{I}$ be an independent set of a block graph G. Imagine that a token is placed at each vertex of $\mathbf{I}$. Intuitively, we say that a token t is $(G, \mathbf{I})$-*confined* if there exists a block B of G such that t can not be slid to a vertex "outside" of B. In the near future, it is hoped that we can characterize this concept formally. This characterization might be the key point for solving SLIDING TOKEN problem for block graphs, which takes us one-step closer to clarify the complexity status of the problem on some bigger graph classes, such as chordal graphs.

Bibliography

[1] M. Bonamy and N. Bousquet. Reconfiguring Independent Sets in Cographs. *arXiv preprints*, arXiv:1406.1433, 2014.

[2] M. Bonamy, M. Johnson, I. Lignos, V. Patel, and D. Paulusma. Reconfiguration graphs for vertex colourings of chordal and chordal bipartite graphs. *Journal of Combinatorial Optimization*, 27(1):132–143, 2014.

[3] P. Bonsma. Independent Set Reconfiguration in Cographs. In D. Kratsch and I. Todinca, editors, *Graph-Theoretic Concepts in Computer Science - WG 2014*, volume 8747 of *Lecture Notes in Computer Science*, pages 105–116. Springer International Publishing, 2014.

[4] P. Bonsma and L. Cereceda. Finding Paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theoretical Computer Science*, 410 (50):5215–5226, 2009.

[5] P. Bonsma, M. Kamiski, and M. Wrochna. Reconfiguring Independent Sets in Claw-Free Graphs. In R. Ravi and I. Grtz, editors, *Algorithm Theory - SWAT 2014*, volume 8503 of *Lecture Notes in Computer Science*, pages 86–97. Springer International Publishing, 2014.

[6] L. Cereceda, J. van den Heuvel, and M. Johnson. Finding Paths Between 3-colorings. *J. Graph Theory*, 67(1):69–82, 2011.

[7] E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada. Linear-Time Algorithm for Sliding Tokens on Trees. *arXiv preprints*, arXiv:1406.6576, 2014.

[8] E. D. Demaine, M. L. Demaine, E. Fox-Epstein, D. A. Hoang, T. Ito, H. Ono, Y. Otachi, R. Uehara, and T. Yamada. Polynomial-Time Algorithm for Sliding Tokens on Trees. In H.-K. Ahn and C.-S. Shin, editors, *Algorithms and Computation - ISAAC 2014*, volume 8889 of *Lecture Notes in Computer Science*, pages 389–400. Springer International Publishing, 2014.

[9] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, 4th edition, 2010.

[10] P. Gopalan, P. Kolaitis, E. Maneva, and C. Papadimitriou. The Connectivity of Boolean Satisfiability: Computational and Structural Dichotomies. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming - ICALP 2006*, volume 4051 of *Lecture Notes in Computer Science*, pages 346–357. Springer Berlin Heidelberg, 2006.

[11] R. A. Hearn and E. 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.

[12] R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation*. AK Peters Series. Taylor & Francis, 2009.

[13] T. Ito, M. Kamiński, and E. Demaine. Reconfiguration of List Edge-Colorings in a Graph. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. Tóth, editors, *Algorithms and Data Structures - WADS 2009*, volume 5664 of *Lecture Notes in Computer Science*, pages 375–386. Springer Berlin Heidelberg, 2009.

[14] T. Ito, E. D. Demaine, N. J. Harvey, C. H. Papadimitriou, M. Sideri, R. Uehara, and Y. Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12-14):1054–1065, 2011.

[15] T. Ito, K. Kawamura, and X. Zhou. An Improved Sufficient Condition for Reconfiguration of List Edge-Colorings in a Tree. In M. Ogihara and J. Tarui, editors, *Theory and Applications of Models of Computation - TAMC 2011*, volume 6648 of *Lecture Notes in Computer Science*, pages 94–105. Springer Berlin Heidelberg, 2011.

[16] T. Ito, M. Kamiński, H. Ono, A. Suzuki, R. Uehara, and K. Yamanaka. On the Parameterized Complexity for Token Jumping on Graphs. In T. Gopal, M. Agrawal, A. Li, and S. Cooper, editors, *Theory and Applications of Models of Computation - TAMC 2014*, volume 8402 of *Lecture Notes in Computer Science*, pages 341–351. Springer International Publishing, 2014.

[17] M. Kamiński, P. Medvedev, and M. Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439(0):9–15, 2012.

[18] K. Makino, S. Tamaki, and M. Yamamoto. An exact algorithm for the Boolean connectivity problem for k-CNF. *Theoretical Computer Science*, 412(35):4613–4618, 2011.

[19] A. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the Parameterized Complexity of Reconfiguration Problems. In G. Gutin and S. Szeider, editors, *Parameterized and Exact Computation - IPEC 2013*, volume 8246 of *Lecture Notes in Computer Science*, pages 281–294. Springer International Publishing, 2013.

[20] A. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over Tree Decompositions. In M. Cygan and P. Heggernes, editors, *Parameterized and*

Exact Computation - IPEC 2014, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer International Publishing, 2014.

[21] J. van den Heuvel. The complexity of change. In S. R. Blackburn, S. Gerke, and M. Wildon, editors, *Surveys in Combinatorics 2013*, pages 127–160. Cambridge University Press, 2013.

[22] D. B. West. *Introduction to Graph Theory*. Featured Titles for Graph Theory Series. Prentice Hall, 2th edition, 2001.

[23] M. Wrochna. Reconfiguration in bounded bandwidth and treedepth. *arXiv preprints*, arXiv:1405.0847, 2014.